عنوان: **ASP.NET MVC #1**
نویسنده: وحید نصیری
تاریخ: ۰۸:۳۲:۰۰ ۱۳۹۱/۰۱/۰۳
آدرس: www.dotnettips.info
برچسب‌ها: MVC

چرا ASP.NET MVC ؟

با وجود فریم ورک پخته‌ای به نام ASP.NET web forms، اولین سؤالی که حین سوئیچ به ASP.NET MVC مطرح می‌شود این است: «برای چی؟». بنابراین تا به این سؤال پاسخ داده نشود، هر نوع بحث فنی در این مورد بی فایده است.

مزایای ASP.NET MVC نسبت به ASP.NET web forms

1) سادگی نوشتن آزمون‌های واحد

مهم‌ترین دلیل استفاده از ASP.NET MVC صرفنظر از تمام دلایل دیگر، بحث طراحی ویژه آن جهت ساده سازی تهیه آزمون‌های واحد است. مشکل اصلی نوشتن آزمون‌های واحد برای برنامه‌های ASP.NET web forms، درگیر شدن مستقیم با تمام جزئیات طول عمر یک صفحه است. به علاوه فایل‌های code behind هر چند به ظاهر کدهای منطق یک صفحه را از کدهای HTML مانند آن جدا می‌کنند اما در عمل حاوی ارجاعات مستقیمی به تک تک عناصر بصری موجود در صفحه هستند (حس غلط جدا سازی کدها از اجزای یک فرم). اگر قرار باشد برای این وب فرم‌ها و صفحات، آزمون واحد بنویسیم باید علاوه بر شبیه سازی چرخه طول عمر صفحه و همچنین رخدادهای رسیده، کار وهله سازی تک تک عناصر بصری را نیز عهده دار شویم. اینجا است که ASP.NET web forms گزینه‌ی مطلوبی برای این منظور نخواهد بود و اگر نوشتن آزمون واحد برای آن غیرممکن نباشد، به همین دلایل آنچنان مرسوم هم نیست.
البته شاید بپرسید که این مساله چه اهمیتی دارد؟ امکان نوشتن ساده‌تر آزمون‌های واحد مساوی است با امکان ساده‌تر اعمال تغییرات به یک پروژه بزرگ. تغییرات در پروژه‌های بزرگی که آزمون واحد ندارند واقعا مشکل است. یک قسمت را تغییر می‌دهید، 10 قسمت دیگر به هم می‌ریزند. اینجا است که مدام باید به کارفرما گفت: «نه!»، «نمیشه!» یا به عبارتی «نمی‌تونم پروژه رو جمع کنم!» چون نمی‌تونم سریع برآورد کنم که این تغییرات کدام قسمت‌ها را تحت تاثیر قرار می‌دهند، کجا به هم ریخت. من باید خودم سریع بتونم مشخص کنم با این تغییر جدید چه قسمت‌هایی به هم ریخته تا اینکه دو روز بعد زنگ بزنند: «باز جایی رو تغییر دادی، یکجای دیگر کار نمی‌کنه!»

2) دستیابی به کنترل بیشتر بر روی اجزای فریم ورک

در طراحی ASP.NET MVC همه‌جا interface ها قابل مشاهد هستند. همین مساله به معنای افزونه پذیری اکثر قطعات تشکیل دهنده ASP.NET MVC است؛ برخلاف ASP.NET web forms. برای مثال تابحال چندین routing engine، view engine و غیره توسط برنامه نویس‌های مستقل برای ASP.NET MVC طراحی شده‌اند که هیچکدام با ASP.NET web forms میسر نیست. برای مثال از view engine پیش فرض آن خوشتان نمی‌آید؟ عوضش کنید ! سیستم اعتبار سنجی توکار آن‌را دوست ندارید؟ آن‌را با یک نمونه بهتر تعویض کنید و الی آخر ...
به علاوه طراحی بر اساس interface ها یک مزیت دیگر را هم به همراه دارد و آن هم ساده سازی mocking (تقلید) آن‌ها است جهت ساده سازی نوشتن آزمون‌های واحد.

3) سرعت بیشتر اجرا

ASP.NET MVC یک سری از قابلیت‌های ذاتی ASP.NET web forms را مانند ViewState حذف کرده است. اگر وب را جستجو کنید، برنامه نویس‌های ASP.NET web forms مدام از این مساله شکایت دارند و راه حل‌های مختلفی را جهت حذف یا فشرده سازی آن ارائه می‌دهند. ViewState در ابتدای امر جهت شبیه سازی محیط دسکتاپ در وب درنظر گرفته شده بود و مهاجرت ساده‌تر برنامه نویس‌های VB6 به وب، اما واقعیت این است که اگر یک برنامه نویس ASP.NET web forms به اندازه آن توجهی نداشته باشد، ممکن است حجم آن در یک صفحه پیچیده تا 500 کیلوبایت یا بیشتر هم برسد. همین مساله بر روی سرعت دریافت و اجرا تاثیر گذار خواهد بود.

4) کنترل‌های ASP.NET web forms آنچنان آش دهن‌سوزی هم نیستند!

خوب، ViewState حذف شده، بنابراین اکثر کنترل‌های ASP.NET web forms هم کاربرد آنچنانی در ASP.NET MVC نخواهند داشت؛ اما واقعیت این است که اکثر اوقات اگر شروع به سفارشی سازی یک کنترل توکار ASP.NET web forms کنید تا مطابق نیازهای کاری شما رفتار کند، پس از مدتی به یک کنترل کاملا از نو بازنویسی شده خواهید رسید! بنابراین در ابتدای امر تا 80 درصد کار اینطور به نظر می‌رسد که به عجب سرعت بالایی در توسعه دست یافته‌ایم، اما هنگامیکه قرار است این 20 درصد پایانی را پر کنیم، به این نتیجه خواهیم رسید که این کنترل‌ها با این وضع ابتدایی که دارند قابل استفاده نیستند و نیاز به دستکاری قابل ملاحظه‌ای دارند تا نیازهای واقعی کاری را برآورده کنند.

5) کنترل کامل بر روی HTML نهایی تولیدی

اگر علاقمند به کار با jQuery باشید، مدام نیاز خواهید تا با ID کنترل‌ها و عناصر صفحه کار کنید. پیشتر ASP.NET web forms این ID را یک طرفه و به صورت مقدار منحصربفردی تولید می‌کرد که جهت کار با فریم ورک‌های جاوا اسکریپتی عموما مشکل ساز بود . البته ASP.NET web forms در نگارش‌های جدید خود مشکل عدم امکان مقدار دهی ClientId سفارشی را برای کنترل‌های وب خود برطرف کرده است و این مورد را می‌توان دستی هم تنظیم کرد ولی در کل باز هم آنچنان کنترلی رو خروجی HTML نهایی کنترل‌های تولیدی نیست مگر اینکه مانند مورد چهارم یاد شده یک کنترل را از صفر بازنویسی کنید!

همچنین اگر باز هم بیشتر با jQuery و ASP.NET web forms کار کرده باشید می‌دانید که jQuery آنچنان سنخیتی با ViewState و Postback وب فرم‌ها ندارد و همین مساله عموما مشکل‌زا است. علاوه بر آن اخیرا مایکروسافت توسعه ASP.NET Ajax خود را تقریبا در حالت تعلیق و واگذار شده به شرکت‌های ثالث درآورده است و توصیه آن‌ها استفاده از jQuery Ajax است. اینجا است که مدل ASP.NET MVC سازگاری کاملی را با jQuery Ajax دارد هم از لحاظ نبود ViewState و هم از جنبه‌ی کنترل کامل بر روی markup نهایی تولیدی.

یا برای مثال خروجی پیش فرض یک GridView، جدول HTML ایی است که این روزها همه‌جا علیه آن صحبت می‌شود. البته یک سری آداپتور CSS friendly برای اکثر این کنترل‌ها موجود است و ... باز هم دستکاری بیش از حد کنترل‌های پیش فرض جهت رسیدن به خروجی دلخواه. تمام این‌ها را در ASP.NET MVC می‌شود با معادل‌های بسیار باکیفیت افزونه‌های jQuery جایگزین کرد و از همه مهم‌تر چون ViewState و مفاهیمی مانند PostBack حذف شده، استفاده از این افزونه‌ها مشکل ساز نخواهد بود.

6) استفاده از امکانات جدید زبان‌های دات نتی

طراحی اصلی ASP.NET web forms مربوط است به دوران دات نت یک؛ زمانیکه نه Generics وجود داشت، نه LINQ و نه آنچنان مباحث TDD یا استفاده از ORMs متداول بود. برای مثال شاید ایجاد یک strongly typed web form الان کمی دور از ذهن به نظر برسد، زمانیکه اصل آن بر مبنای بکارگیری گسترده datatable و dataset بوده است (با توجه به امکانات زبان‌های دات نتی در آن دوران). بنابراین اگر علاقمند هستید که این امکانات جدید را بکاربگیرید، ASP.NET MVC برای استفاده از آن‌ها طراحی شده است!

7) از ASP.NET web forms ساده‌تر است

طراحی ASP.NET MVC بر اساس ایده Convention over configuration است. به این معنا که اجزای آن بر اساس یک سری قرار داد در کنار هم مشغول به کار هستند. مشخص است View باید کجا باشد، نام کنترلرها چگونه باید تعیین شوند و قرار داد مرتبط به آن چیست، مدل باید کجا قرار گیرد، قرار داد پردازش آدرس‌های صفحات سایت به چه نحوی است و الی آخر. خلاصه در بدو امر با یک فریم ورک حساب شده که شما را در مورد نحوه استفاده صحیح از آن راهنمایی می‌کند، مواجه هستید.

به همین ترتیب هر پروژه MVC دیگری را هم که مشاهده کنید، سریع می‌توانید تشخیص دهد قراردادهای بکارگرفته شده در آن چیست. بنابراین اگر قرار است ASP.NET را امروز شروع کنید و هیچ سابقه‌ای هم از وب فرم‌ها ندارید، یک راست با ASP.NET MVC شروع کنید.

8) محدود به پیاده سازی مایکروسافت نیست

پیاده سازی‌های مستقلی هم از ASP.NET MVC توسط اشخاص و گروه‌های خارج از مایکروسافت وجود دارد: ^ ، ^ ، ^ ، ^ و ...

و در پایان یکی دیگر از دلایل سوئیچ به ASP.NET MVC ، «یاد گرفتن یک چیز جدید است» یا به عبارتی فرا گرفتن یک روش دیگر برای حل مسایل، هیچگاه ضرری را به همراه نخواهد داشت که هیچ، بلکه باعث بازتر شدن میدان دید نیز خواهد گردید.

یک دیدگاه دیگر

ASP.NET MVC برای شما مناسب نخواهد بود اگر ...

1) با پلی‌مرفیزم مشکل دارید.

ASP.NET MVC پر است از interfaces، abstract classes، virtual methods و امثال آن. بنابراین اگر تازه کار هستید، ابتدا باید مفاهیم شیءگرایی را تکمیل کنید.

2) اگر نمی‌توانید فریم ورک خودتون رو بر پایه ASP.NET MVC بنا کنید!

ASP.NET MVC برخلاف وب فرم‌ها به همراه آنچنان تعداد بالایی کنترل و افزونه از پیش مهیا شده نیست. در بدو امر شما فقط یک سری html helper، url helper و ajax helper ساده را خواهید دید؛ این نقطه ضعف ASP.NET MVC نیست. عمدا به این نحو طراحی شده است. همانطور که عنوان شد اکثر اجزای این فریم ورک قابل تعویض است. بنابراین دست شما را باز گذاشته است تا با پیاده سازی این اینترفیس‌ها، امکانات جدیدی را خلق کنید. البته پس از این چندین و چند سال که از ارائه آن می‌گذرد، به اندازه کافی افزونه برای ASP.NET MVC طراحی شده است که به هیچ عنوان احساس کمبود نکنید یا اینکه نیازی هم نداشته باشید تا آنچنان فریم ورک خاصی را بر پایه ASP.NET MVC تهیه کنید. برای مثال پروژه MvcContrib موجود است یا شرکت telerik یک مجموعه سورس باز کامل مخصوص ASP.NET MVC را ارائه داده است و الی آخر.

3) اگر نمی‌توانید از کتابخانه‌های سورس باز استفاده کنید.

همانطور که عنوان شد ASP.NET MVC به همراه کوهی از کنترل‌ها ارائه نشده است. اکثر افزونه‌های آن سورس باز هستند و کار با آن‌ها هم دنیای خاص خودش را دارد. چگونه باید کتابخانه‌های مناسب را پیدا کرد، کجا سؤال پرسید، کجا باگ گزارش داد، چگونه مشارکت کرد و غیره. خلاصه منتظر یک بسته شکیل حاضر و آماده نباید بود. خود ASP.NET MVC هم تحت مجوز MSPL به صورت سورس باز در دسترس است .

و یک نکته تکمیلی

مایکروسافت مدتی است شروع کرده به پرورش و زمزمه ایده « یک ASP.NET واحد ». به عبارتی قصد دارند در یکی دو نگارش بعد، این دو (وب فرم و MVC) را یکی کنند. هم اکنون اگر مطالب وبلاگ‌ها را مطالعه کنید زیرساخت آن به نام ASP.NET Web API آماده شده است و در مرحله بتا است. نکته جالب اینجا است که این Web API امکان تعریف یکپارچه و مستقیم کنترلرهای MVC را در وب فرم‌ها میسر می‌کند . ولی باز هم نام آن Controller است یعنی جزئی از ASP.NET MVC و کسی می‌تواند از آن استفاده کند که با MVC مشکلی نداشته باشد. بنابراین یادگیری MVC هیچ ضرری نخواهد داشت و جای دوری نخواهد رفت!

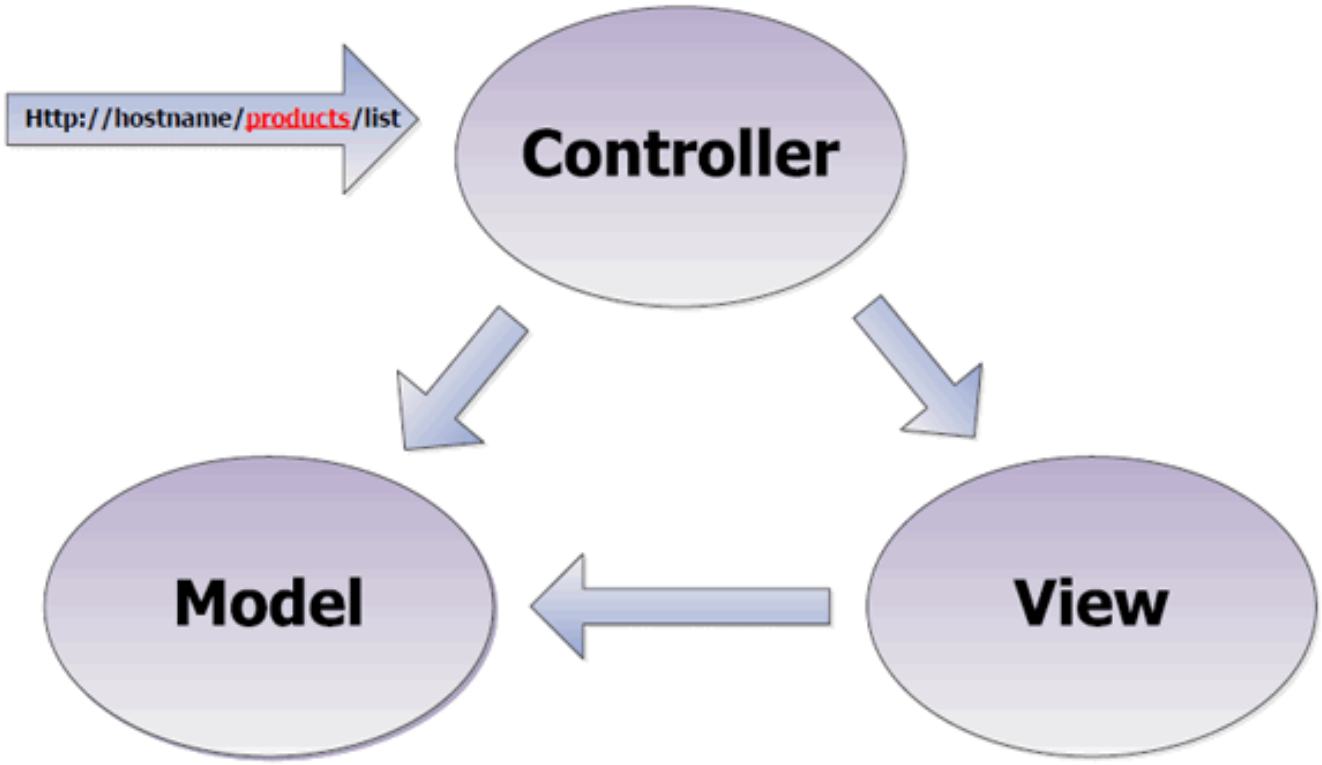
عنوان: **ASP.NET MVC #2**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۰۴ ۱۷:۴۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

MVC چیست و اساس کار آن چگونه است؟

الگوی MVC در سال‌های اول دهه 70 میلادی در شرکت زیراکس توسط خالقین زبان اسمال‌تاک که جزو اولین زبان‌های شیءگرا محسوب می‌شود، ارائه گردید. نام MVC از الگوی Model-View-Controller گرفته شده و چندین دهه است که در صنعت تولید نرم افزار مورد استفاده می‌باشد. هدف اصلی آن جدا سازی مسئولیت‌های اجزای تشکیل دهنده «لایه نمایشی» برنامه است.
این الگو در سال 2004 برای اولین بار در سکویی به نام Rails به کمک زبان روبی جهت ساخت یک فریم ورک وب MVC مورد استفاده قرار گرفت و پس از آن به سایر سکوها مانند جاوا، دات نت (در سال 2007)، PHP و غیره راه یافت.
تصاویری را از این تاریخچه در ادامه ملاحظه می‌کنید؛ از دکتر Trygve Reenskaug تا شرکت زیراکس و معرفی آن در Rails به عنوان اولین فریم ورک وب MVC.

شکل ۱

C در MVC معادل `Controller` است. کنترلر قسمتی است که کار دریافت ورودی‌های دنیای خارج را به عهده دارد؛ مانند پردازش یک درخواست HTTP ورودی.

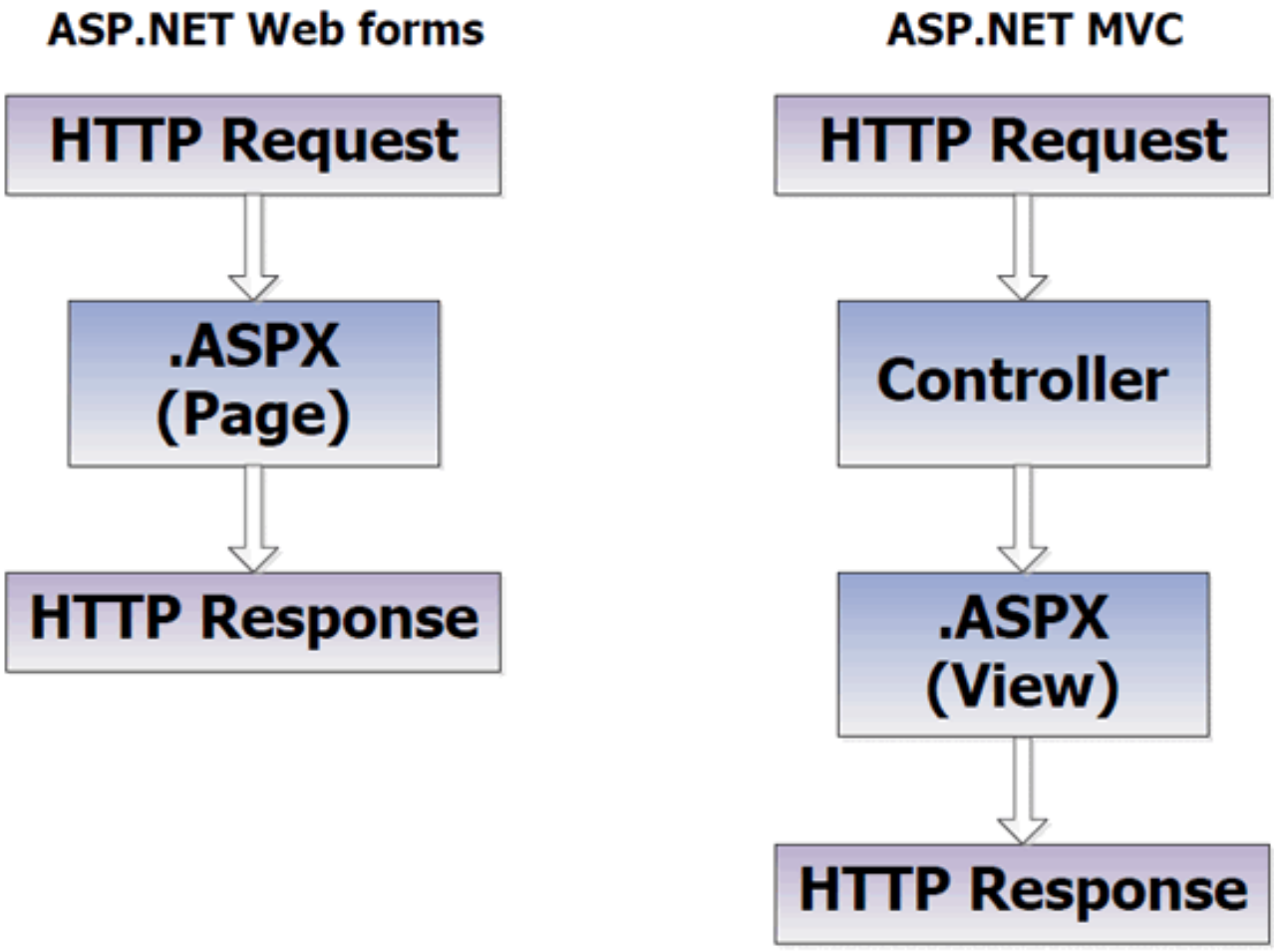


شکل ۲

زمانیکه کنترلر این درخواست را دریافت می‌کند، کار وهله سازی Model را عهده دار خواهد شد و حاوی اطلاعاتی است که نهایتا در اختیار کاربر قرار خواهد گرفت تا فرآیند پردازش درخواست رسیده را تکمیل نماید. برای مثال اگر کاربری جهت دریافت آخرین اخبار به سایت شما مراجعه کرده است، در اینجا کار تهیه لیست اخبار بر اساس مدل مرتبط به آن صورت خواهد گرفت. بنابراین کنترلرها، پایه اصلی و مدیر ارکستر الگوی MVC محسوب می‌شوند.
در ادامه، کنترلر یک View را جهت نمایش Model انتخاب خواهد کرد. View در الگوی MVC یک شیء ساده است. به آن می‌توان به شکل یک قالب که اطلاعاتی را از Model دریافت نموده و سپس آن‌ها را در مکان‌های مناسبی در صفحه قرار می‌دهد، نگاه کرد.
نتیجه استفاده از این الگو، ایزوله سازی سه جزء یاد شده از یکدیگر است. برای مثال View نمی‌داند و نیازی ندارد که بداند چگونه باید از لایه دسترسی به اطلاعات کوئری بگیرد. یا برای مثال کنترلر نیازی ندارد بداند که چگونه و در کجا باید خطایی را با رنگی مشخص نمایش دهد. به این ترتیب انجام تغییرات در لایه رابط کاربری برنامه در طول توسعه کلی سیستم، ساده‌تر خواهد شد.
همچنین در اینجا باید اشاره کرد که این الگو مشخص نمی‌کند که از چه نوع فناوری دسترسی به اطلاعاتی باید استفاده شود. می‌توان از بانک‌های اطلاعاتی، وب سرویس‌ها، صف‌ها و یا هر نوع دیگری از اطلاعات استفاده کرد. به علاوه در اینجا در مورد نحوه طراحی Model نیز قیدی قرار داده نشده است. این الگو تنها جهت ساخت بهتر و اصولی «رابط کاربری» طراحی شده است و بس.

تفاوت مهم پردازشی ASP.NET MVC با ASP.NET Web forms

اگر پیشتر با ASP.NET Web forms کار کرده باشید اکنون شاید این سؤال برایتان وجود داشته باشد که این سیستم جدید در مقایسه با نمونه قبلی، چگونه درخواست‌ها را پردازش می‌کند.

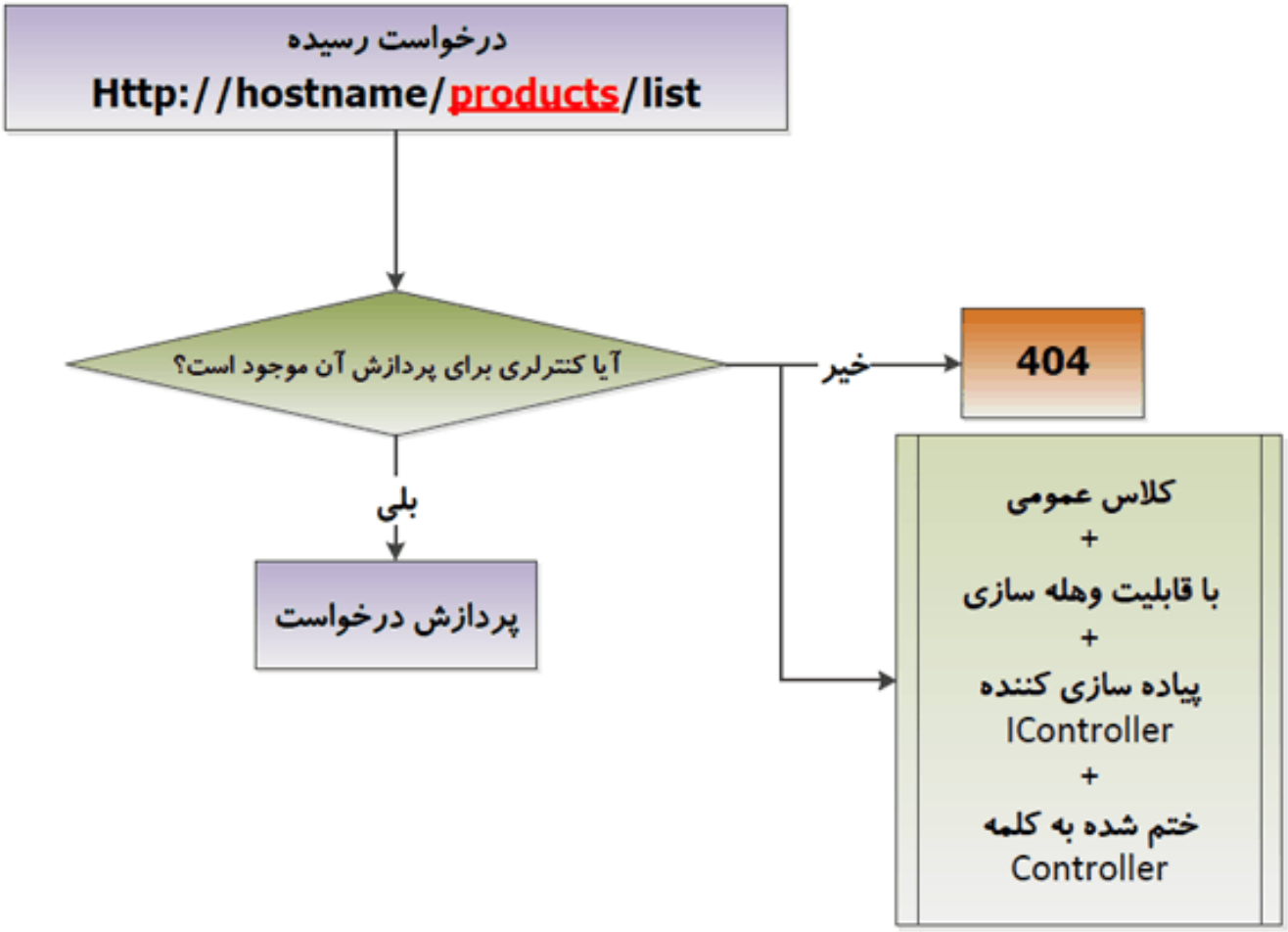


شکل ۳

همانطور که مشاهده می‌کنید، در وب فرم‌ها زمانیکه درخواستی دریافت می‌شود، این درخواست به یک فایل موجود در سیستم مثلا default.aspx ارسال می‌گردد. سپس ASP.NET یک کلاس وهله سازی شده معرف آن صفحه را ایجاد کرده و آن‌را اجرا می‌کند. در اینجا چرخه طول عمر صفحه مانند page_load و غیره رخ خواهد داد. جهت انجام این وهله سازی، View به فایل code behind خود گره خورده است و جدا سازی خاصی بین این دو وجود ندارد. منطق صفحه به markup آن که معادل است با یک فایل فیزیکی بر روی سیستم، کاملا مقید است. در ادامه، این پردازش صورت گرفته و HTML نهایی تولیدی به مرورگر کاربر ارسال خواهد شد.
در ASP.NET MVC این نحوه پردازش تغییر کرده است. در اینجا ابتدا درخواست رسیده به یک کنترلر هدایت می‌شود و این کنترلر چیزی نیست جز یک کلاس مجزا و مستقل از هر نوع فایل ASPX ایی در سیستم. سپس این کنترلر کار پردازش درخواست رسیده را شروع کرده، اطلاعات مورد نیاز را جمع آوری و سپس به View ایی که انتخاب می‌کند، جهت نمایش نهایی ارسال خواهد کرد. در اینجا View این اطلاعات را دریافت کرده و نهایتا در اختیار کاربر قرار خواهد داد.

آشنایی با قرارداد یافتن کنترلرهای مرتبط

تا اینجا دریافتیم که نحوه پردازش درخواست‌ها در ASP.NET MVC بر مبنای کلاس‌ها و متدها است و نه بر مبنای فایل‌های فیزیکی موجود در سیستم. اگر درخواستی به سیستم ارسال می‌شود، در ابتدا، این درخواست جهت پردازش، به یک متد عمومی موجود در یک کلاس کنترلر هدایت خواهد شد و نه به یک فایل فیزیکی ASPX (برخلاف وب فرم‌ها).



شکل ۴

همانطور که در تصویر مشاهده می‌کنید، در ابتدای پردازش یک درخواست، آدرسی به سیستم ارسال خواهد شد. بر مبنای این آدرس، نام کنترلر که در اینجا زیر آن خط قرمز کشیده شده است، استخراج می‌گردد (برای مثال در اینجا نام این کنترلرProducts است). سپس فریم ورک به دنبال کلاس این کنترلر خواهد گشت. اگر آن‌را در اسمبلی پروژه بیابد، از آن خواهد خواست تا درخواست رسیده را پردازش کند؛ در غیراینصورت پیغام 404 یا یافت نشد، به کاربر نمایش داده می‌شود.
اما فریم ورک چگونه این کلاس کنترلر درخواستی را پیدا می‌کند؟
در زمان اجرا، اسمبلی اصلی پروژه به همراه تمام اسمبلی‌هایی که به آن ارجاعی دارند جهت یافتن کلاسی با این مشخصات اسکن خواهند شد:
-1 این کلاس باید عمومی باشد.
-2 این کلاس نباید abstract باشد (تا بتوان آن‌را به صورت خودکار وهله سازی کرد).
-3 این کلاس باید اینترفیس استاندارد IController را پیاده سازی کرده باشد.
-4 و نام آن باید مختوم به کلمه Controller باشد (همان مبحث Convention over configuration یا کار کردن با یک سری قرار داد از پیش تعیین شده).

برای مثال در اینجا فریم ورک به دنبال کلاسی به نام ProductsController خواهد گشت.
شاید تعدادی از برنامه نویس‌های ASP.NET MVC تصور کنند که فریم ورک در پوشه‌ی استانداردی به نام Controllers به دنبال این کلاس خواهد گشت؛ اما در عمل زمانیکه برنامه کامپایل می‌شود، پوشه‌ای در این اسمبلی وجود نخواهد داشت و همه چیز از طریق Reflection مدیریت خواهد شد.

طبق تعریف `view` اطلاعاتی که کنترلر از `model` دریافت کرده را، از کنترلر مربوط دریافت و در مکان‌های مناسبی قرار می‌دهد. پس فلش که در شکل ابتدایی نشان داده شده است چه ارتباطی را بین `view` و `model` نشان می‌دهد؟

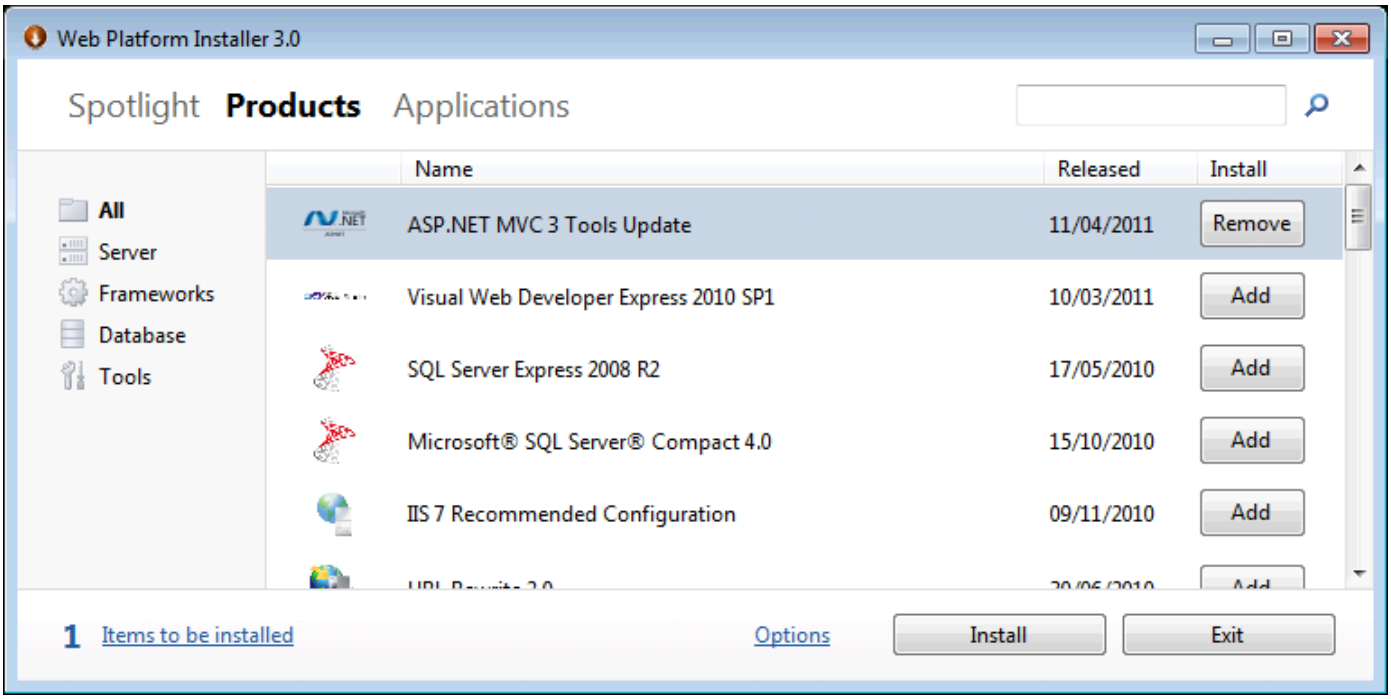
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۴ ۸:۴۸

برای توضیحات بیشتر مراجعه کنید به قسمت پنجم . یک قسمت کامل به آن اختصاص داده شده.

عنوان: **ASP.NET MVC #3**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۰۶ ۱۶:۳۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

تهیه پیش‌نیازهای شروع به کار با ASP.NET MVC

در زمان نگارش این مطلب، نگارش نهایی ASP.NET MVC 3 در دسترس است و همچنین نگارش بتای 4 آن نیز قابل دریافت و نصب می‌باشد. بنابراین فعلا اساس را بر مبنای نگارشی قرار خواهیم داد که در محیط کاری قابل استفاده باشد.
ASP.NET MVC 3 پس از ارائه Visual Studio 2010، منتشر شد و VS.NET به صورت پیش فرض به همراه ASP.NET MVC 2 است. ساده‌ترین روش نصب ASP.NET MVC 3 بر روی VS 2010 استفاده از برنامه رایگانی است به نام Web Platform Installer. این برنامه را از این آدرس می‌توان دریافت کرد: http://microsoft.com/web/downloads
پس از دریافت آن حداقل دو راه برای نصب ASP.NET MVC 3 وجود دارد. یا گزینه‌ی نصب ASP.NET MVC 3 Tools Update را انتخاب کنید و یا سرویس پک یک VS 2010 را از طریق این برنامه یا جداگانه (بسته کامل و مستقل) دریافت و نصب نمائید. VS 2010 SP1 نیز به همراه ASP.NET MVC 3 است؛ همچنین IIS Express را که نسخه ساده شده IIS 7.5 مخصوص توسعه دهنده‌ها است، می‌توان با این نگارش یکپارچه کرد.

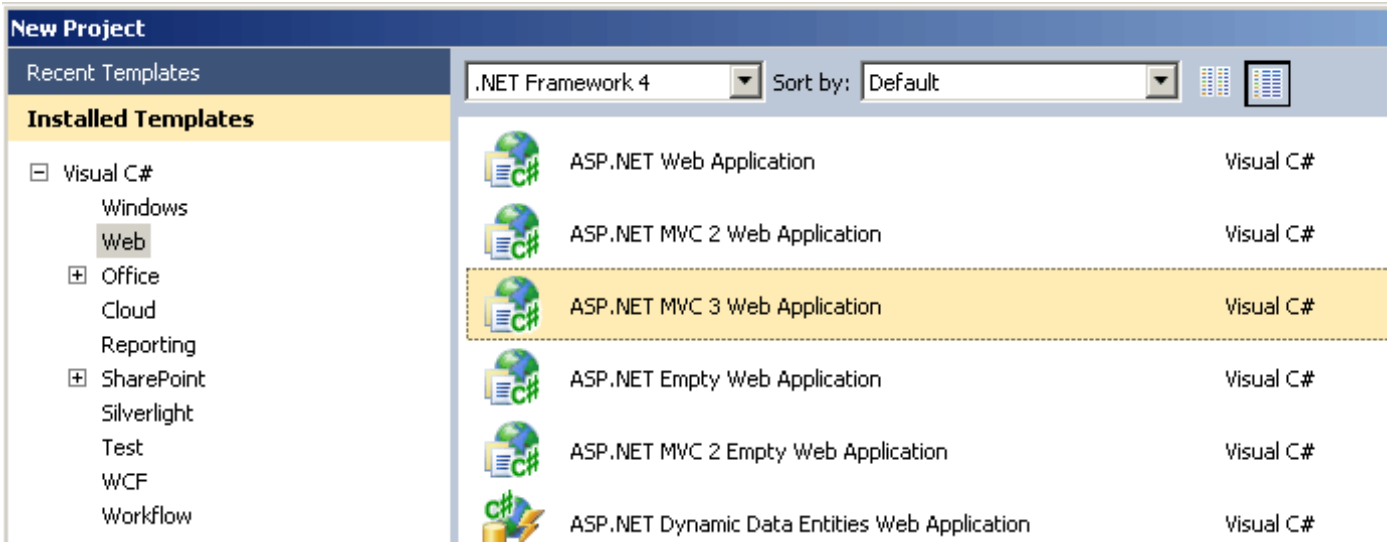


شکل ۱

بنابراین به صورت خلاصه بهترین کار این است که سرویس پک یک VS 2010 را یکبار نصب نمائید. اگر این نصب از طریق برنامه Web Platform Installer باشد، به صورت خودکار IIS Express را هم انتخاب و نصب خواهد کرد. اگر فقط SP1 را به صورت مستقل دریافت کرده‌اید، حاوی IIS Express نیست و باید جداگانه آن‌را دریافت و نصب نمائید (^). البته نصب IIS Express در اینجا یک گزینه اختیاری است و الزامی نیست.

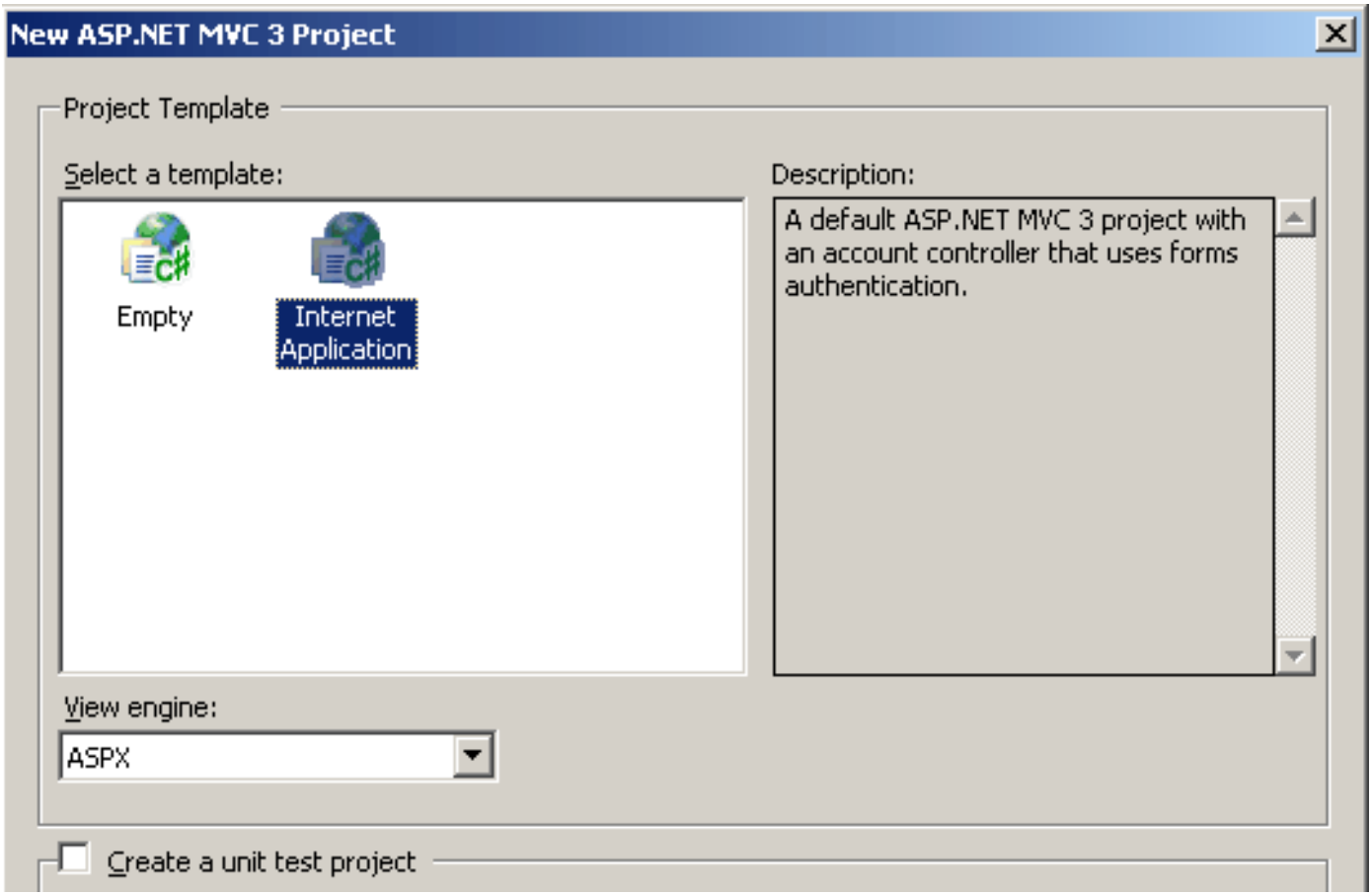
مروری بر ساختار یک پروژه ASP.NET MVC

پس از نصب پیش نیازها، امکان انتخاب یک پروژه وب ASP.NET MVC 3 در VS 2010 میسر خواهد شد:



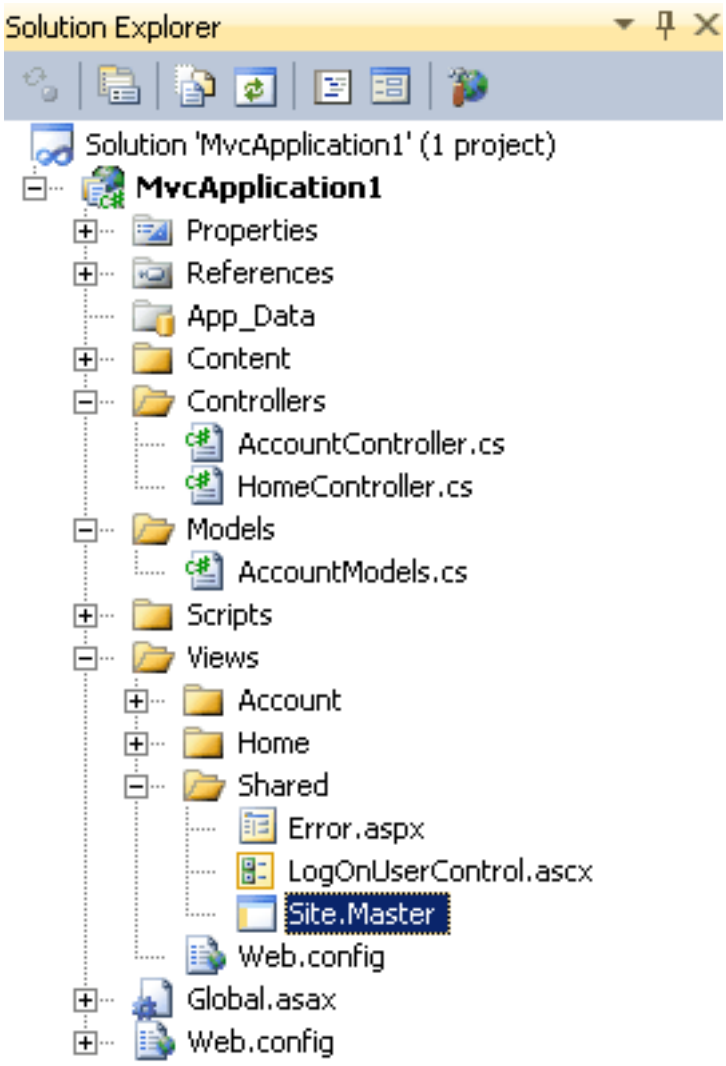
شکل ۲

در اینجا گزینه‌ی ASP.NET MVC 3 Web Application را انتخاب می‌کنیم. در صفحه بعدی که ظاهر می‌شود:



شکل ۳

حالت Internet Application به همراه یک سری مدل و کنترلر از پیش نوشته شده جهت مدیریت ورود به سایت و ثبت نام در سایت است و حالت Empty تنها به همراه ساختار پیش فرض پوشه‌های یک پروژه ASP.NET MVC است.
فعلا جهت توضیحات اولیه بیشتر، گزینه‌ی Internet Application و نوع View Engine را هم ASPX انتخاب می‌کنیم. کار View Engine، رندر یک View به شکل HTML و ارائه نهایی اطلاعات آن به کاربر است. این نوع‌های متفاوت هم فقط در Syntax تفاوت دارند (به آن templating language هم گفته می‌شود). نوع ASPX همان Syntax متداول قدیمی ASP.NET را تداعی می‌کند و نوع Razor به صورت اختصاصی برای ASP.NET MVC تهیه شده است.
باید در نظر داشت که گزینه مرجح از نگارش 3 به بعد، Razor است (البته این هم سلیقه‌ای است. اگر هیچکدام از این دو را هم نخواهید استفاده کنید مشکلی نیست! می‌شود کلا آن را عوض کرد). هدفم هم از انتخاب ASPX نمایش یک سری ریزه کاری است که شاید برای برنامه نویس‌های ASP.NET Web forms جالب باشد. این موارد را در حالت انتخاب Razor به این وضوح مشاهده نخواهید کرد و محیط خیلی ساده شده است.



شکل ۴

همانطور که ملاحظه می‌کنید این فریم ورک یک سری پوشه پیش فرض را توصیه می‌کند. بدیهی است که ضرورتی ندارد تا پوشه Models یا پوشه Controllers حتما در همین پروژه قرار داشته باشند؛ چون زمانیکه پروژه کامپایل شد، محل این پوشه بندی‌ها آنچنان اهمیتی ندارد.
نکته جالب در این تصویر، فایل Site.Master است. بله، این فایل شبیه به همان فایل master page موجود در ASP.NET Web form

است که قالب کلی سایت را به همراه داشته و سایر صفحات، قالب خود را از آن به ارث می‌برند. حتی تگ runat=server هم به وضوح در این فایل، در چندین جای آن قابل مشاهده است. تنها تفاوت آن نداشتن فایل code behind است. asp:ContentPlaceHolder نیز در آن تعریف شده است. خلاصه این محیط جدید به معنای دور ریختن تمام آنچیزی که در Web forms وجود دارد نیست. برای نمونه اگر فایل ChangePassword.aspx موجود در پوشه Account را باز کنید، باز هم همان asp:Content معروف به همراه تگ runat=server قابل مشاهده است. برای مثال این محتوای صفحه Error.aspx پیش فرض آن است:

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
 Inherits="System.Web.Mvc.ViewPage<System.Web.Mvc.HandleErrorInfo>" %>

<asp:Content ID="errorTitle" ContentPlaceHolderID="TitleContent" runat="server">
    Error
</asp:Content>

<asp:Content ID="errorContent" ContentPlaceHolderID="MainContent" runat="server">
    <h2>
        Sorry, an error occurred while processing your request.
    </h2>
</asp:Content>
```

اگر از قسمت Inherits آن صرفنظر کنیم، «هیچ» تفاوتی با ASP.NET Web forms ندارد؛ علت هم به این بر می‌گردد که موتوری که Web forms و MVC از آن استفاده می‌کنند، یکی است. هر دو بر فراز موتور ASP.NET معنا پیدا خواهند کرد.

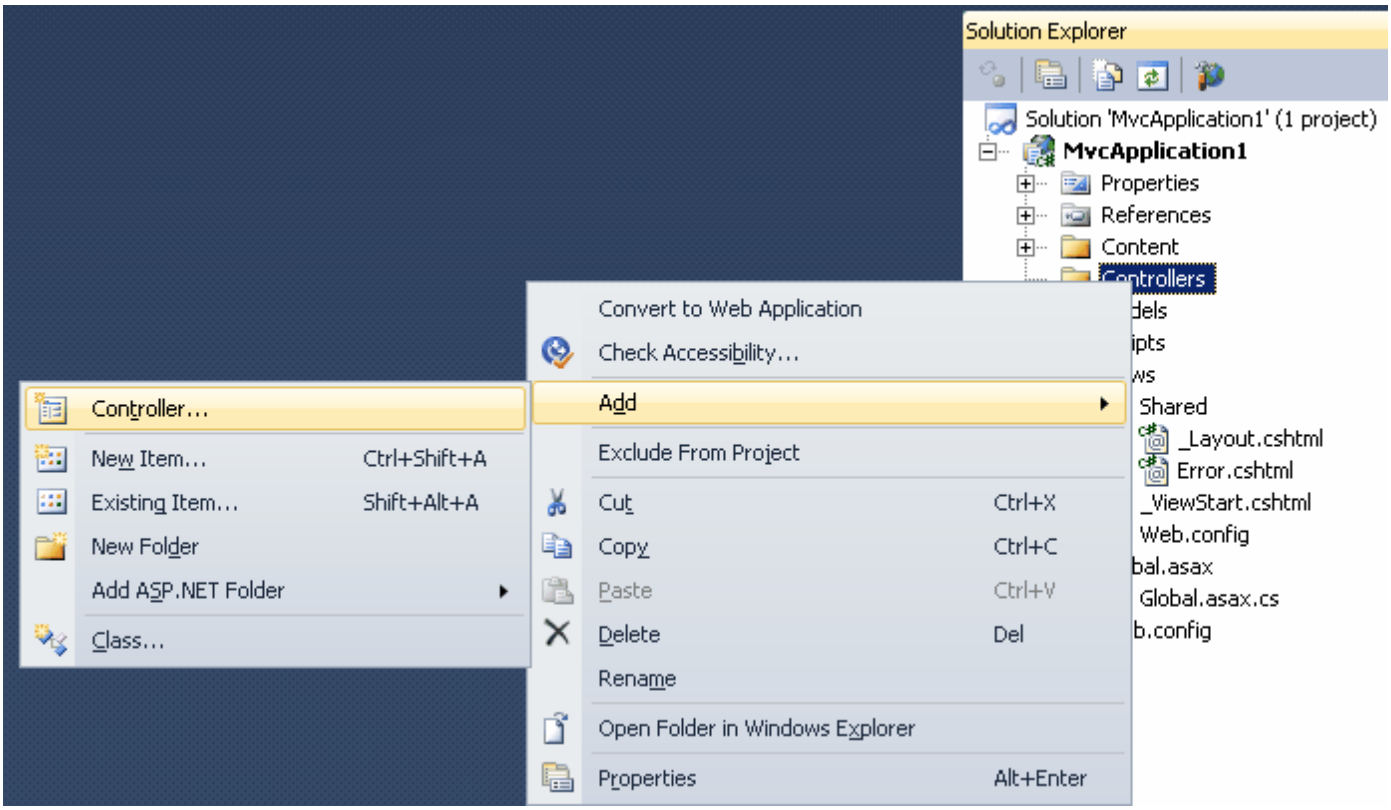
قرار دادهای پوشه‌های پیش فرض یک پروژه ASP.NET MVC

پوشه Controllers حاوی کلاس‌های کنترلری است که درخواست‌های رسیده را مدیریت می‌کنند.
پوشه Models حاوی کلاس‌هایی است که اشیاء تجاری و همچنین کار با اطلاعات را تعریف و مدیریت می‌کنند.
در پوشه Views، فایل‌های قالب‌های رابط کاربری که مسئول ارائه خروجی به کاربر هستند قرار می‌گیرند. همچنین مطابق قرارداد دیگری، اگر نام کنترلر ما مثلا ProductController باشد (با توجه به اینکه نام کلاس آن هم مطابق قرارداد، مختوم به کلمه Controller است)، فایل‌های Viewهای مرتبط با آن در پوشه Views/Product قرار خواهند گرفت.
در پوشه Scripts، فایل‌های جاوا اسکریپت مورد استفاده در سایت قرار خواهند گرفت.
پوشه Content محل قرارگیری فایل‌های CSS و تصاویر است.
پوشه App_Data جایی است که فایل‌هایی با قابلیت read/write در آن قرار می‌گیرند (و باید دقت داشت که فقط همینجا هم باید قرار گیرند و گرنه این نوشتن‌ها در مکان‌های متفرقه، ممکن است سبب ری استارت شدن برنامه شوند:(^)).

عنوان: **ASP.NET MVC #4**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۰۸ ۰۰:۴۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

بررسی نحوه ارتباطات بین اجزای مختلف الگوی MVC در ASP.NET MVC

اینبار برخلاف قسمت قبل، قالب پروژه خالی ASP.NET MVC را در VS.NET انتخاب کرده (ASP.NET MVC 3 Web Application) و بعد انتخاب قالب Empty نمایش داده شده) و سپس پروژه جدیدی را شروع می‌کنیم. View Engine را هم Razor در نظر خواهیم گرفت.
پس از ایجاد ساختار اولیه پروژه، بدون اعمال هیچ تغییری، برنامه را اجرا کنید. بلافاصله با پیغام The resource cannot be found یا 404 یافت نشد، مواجه خواهیم شد.
همانطور که در پایان قسمت دوم نیز ذکر شد، پردازش‌ها در ASP.NET MVC از کنترلرها شروع می‌شوند و نه از صفحات وب. بنابراین برای رفع این مشکل نیاز است تا یک کلاس کنترلر جدید را اضافه کنیم. به همین جهت بر روی پوشه استاندارد Controllers کلیک راست کرده، از منوی ظاهر شده قسمت Add، گزینه‌ی Controller را انتخاب کنید:



شکل ۱

در صفحه بعدی که ظاهر می‌شود، نام HomeController را وارد کنید (با توجه به اینکه مطابق قراردادهای ASP.NET MVC، نام کنترلر باید به کلمه Controller ختم شود). البته لازم به ذکر است که این مراحل را به همان شکل متداول مراجعه به منوی Project و انتخاب Add Class و سپس ارث بری از کلاس Controller نیز می‌توان انجام داد و طی این مراحل الزامی نیست. کلاسی که به صورت خودکار از طریق منوی Add Controller یاد شده ایجاد می‌شود، به شکل زیر است:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

سؤال:

در قسمت دوم عنوان شد که کنترلر باید کلاسی باشد که اینترفیس IController را پیاده سازی کرده است، اما در اینجا ارث بری از کلاس Controller را شاهد هستیم. جریان چیست؟!

سلسله مراتبی که بکارگرفته شده به صورت زیر است:

```
public abstract class ControllerBase : IController
public abstract class Controller : ControllerBase
```

ControllerBase، اینترفیس IController را پیاده سازی کرده و سپس کلاس Controller از کلاس ControllerBase مشتق شده است. شاید بپرسید که این همه پیچ و تاب برای چیست؟!

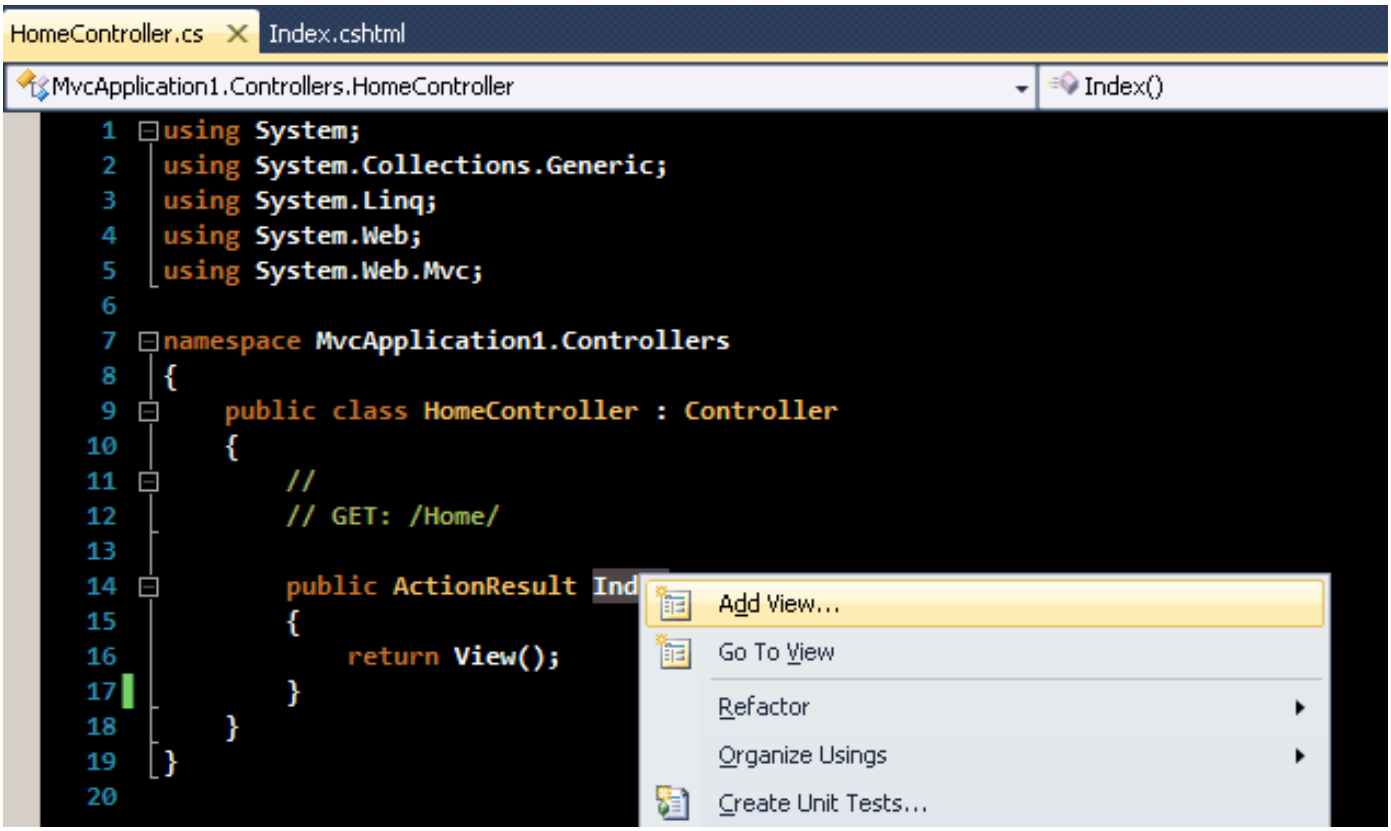
مشکلی که با اینترفیس خالص وجود دارد، عدم نگارش پذیری آن است. به این معنا که اگر متدی یا خاصیتی در نگارش بعدی به این اینترفیس اضافه شد، هیچکدام از پروژه‌های قدیمی دیگر کامپایل نخواهند شد و باید ابتدا این متد یا خاصیت جدید را نیز لحاظ کنند. اینجا است که کار کلاس‌های abstract شروع می‌شود. در یک کلاس abstract می‌توان پیاده سازی پیش فرضی را نیز ارائه داد. به این ترتیب مصرف کننده نهایی کلاس Controller متوجه این تغییرات نخواهد شد.

اگر برنامه نویس «من» باشم، شما رو وادار خواهم کرد که متدهای جدید اینترفیس تعریفی‌ام را پیاده سازی کنید! همینه که هست! اما اگر طراح مایکروسافت باشد، بلافاصله انبوهی از جماعت ایرادگیر که بالای 100 تا از کنترلرهای اون‌ها الان فقط در یک پروژه از کار افتاده، ممکن است جلوی دفتر مایکروسافت دست به خود سوزی بزنند! اینجا است که مایکروسافت مجبور است تا این پیچ و تاب‌ها را اعمال کند که اگر روزی متدی در اینترفیس IController بنابر نیازهای جدید درنظر گرفته شد، بتوان سریع یک پیاده سازی پیش فرض از آن‌را در کلاس‌های abstract یاد شده قرار داد (یکی از تفاوت‌های مهم کلاس‌های abstract با اینترفیس‌ها) تا جماعت ایرادگیر و نق‌زن متوجه تغییری نشوند و باز هم پروژه‌های قدیمی بدون مشکل کامپایل شوند. تابحال به فلسفه وجودی کلاس‌های abstract از این دیدگاه فکر کرده بودید؟!

بعد از این توضیحات و کارها، اگر اینبار برنامه را اجرا کنیم، خطای زیر نمایش داده می‌شود:

```
The view 'Index' or its master was not found or no view engine supports the searched locations. The
following locations were searched:
~/Views/Home/Index.aspx
~/Views/Home/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
~/Views/Home/Index.cshtml
~/Views/Home/Index.vbhtml
~/Views/Shared/Index.cshtml
~/Views/Shared/Index.vbhtml
```

همانطور که ملاحظه می‌کنید چون نام کنترلر ما Home است، فریم ورک در پوشه استاندارد Views در زیر پوشه‌ای به همان نام Home، به دنبال یک سری فایل می‌گردد. فایل‌های aspx مربوط به View Engine ایی به همین نام بوده و فایل‌های cshtml و vbhtml مربوط به View Engine دیگری به نام Razor هستند.
بنابراین نیاز است تا یکی از این فایل‌ها را در مکان‌های یاد شده ایجاد کنیم. برای این منظور حداقل دو راه وجود دارد. یا دستی اینکار را انجام دهیم یا اینکه از ابزار توکار خود VS.NET برای ایجاد یک View جدید استفاده کنیم.
برای ایجاد View ایی مرتبط با متد Index (در ASP.NET MVC نام دیگر متدهای قرار گرفته در یک کنترلر، Action نیز می‌باشد)، روی خود متد کلیک راست کرده و گزینه Add View را انتخاب کنید:



شکل ۲

در صفحه بعدی ظاهر شده، پیش فرض‌ها را پذیرفته و بر روی دکمه Add کلیک نمائید. اتفاقی که رخ خواهد داد شامل ایجاد فایل Index.cshtml، با محتوای زیر است (با توجه به اینکه زبان پروژه سی شارپ است و View Engine انتخابی Razor می‌باشد، cshtml تولید گردید و گرنه vbhtml ایجاد می‌شد):

```
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
```

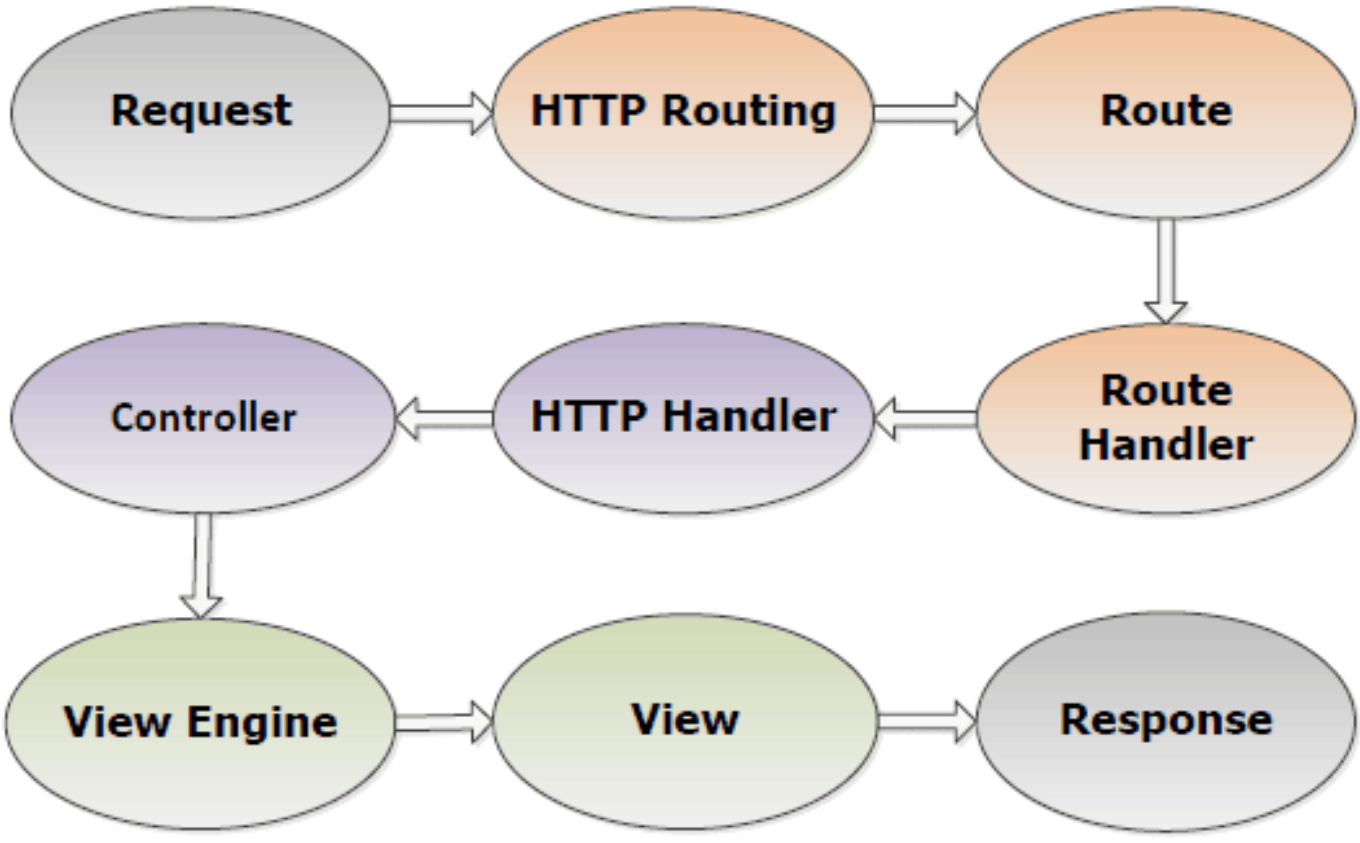
مجددا برنامه را اجرا کنید. اینبار بدون خطایی کلمه Index را در صفحه تولیدی می‌توان مشاهده کرد. نکته جالب این فایل‌های View جدید، عدم مشاهده ویژگی‌های runat=server و سایر موارد مشابه است.

چند سؤال مهم:

در حین ایجاد اولین کنترلر جهت نمایش صفحه پیش فرض برنامه، نام HomeController انتخاب شد. چرا مثلا نام TestController وارد نشد؟ برنامه از کجا متوجه شد که باید حتما این کنترلر را پردازش کند. نقش متد Index چیست؟ آیا حتما باید Index باشد و در اینجا نام دیگری را نمی‌توان وارد کرد؟ «قرارداد» پردازشی این‌ها کجا تنظیم می‌شود؟ فریم ورک، این سیم کشی‌ها را چگونه انجام می‌دهد؟

پاسخ به تمام این سؤال‌ها، در ویژگی «مسیریابی یا Routing» نهفته است. فایل Global.asax.cs برنامه را باز کنید. تعاریف مرتبط با مسیریابی پیش فرض را در متد RegisterRoutes آن می‌توان مشاهده کرد:

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter
defaults
            );
}
```



شکل ۳

پروسه هدایت یک درخواست HTTP به یک کنترلر، در اینجا مسیریابی یا Routing نامیده می‌شود. این قابلیت در فضای نام System.Web.Routing تعریف شده است و باید دقت داشت که جزو ASP.NET MVC نیست. این امکانات جزو ASP.NET Runtime است و به همراه دات نت 3.5 سرویس پک یک برای اولین بار ارائه شد. بنابراین جهت استفاده در ASP.NET Web forms نیز مهیا است. در ASP.NET MVC از این امکانات برای ارسال درخواست‌ها به کنترلرها استفاده می‌شود.

در متد routes.MapRoute فراخوانی شده‌ای که در کدهای بالا ملاحظه می‌کنید، کار نگاشت یک URL به Actionهای یک کنترلر صورت می‌گیرد (یا همان متدهای تعریف شده در کنترلرها). همچنین از این URLها پارامترهای این متدها یا اکشن‌ها نیز قابل استخراج است.
در متد MapRoute، اولین پارامتر تعریف شده، یک نام پیش فرض است و در ادامه اگر آدرسی را به فرم «یک چیزی اسلش یک چیزی اسلش یک چیزی» یافت، اولین قسمت آن‌را به عنوان نام کنترلر تفسیر خواهد کرد، دومین قسمت آن، نام متد عمومی موجود در کنترلر فرض شده و سومین قسمت به عنوان پارامتر ارسالی به این متد پردازش می‌شود.
برای مثال از آدرس زیر اینطور می‌توان دریافت که:

```
http://hostname/home/about
```

Home نام کنترلی است که فریم ورک به دنبال آن خواهد گشت تا این درخواست رسیده را پردازش کند و about نام متدی عمومی در این کلاس است که به صورت خودکار فراخوانی می‌گردد. در اینجا پارامتر id ایی هم وجود ندارد. در یک برنامه امکان تعریف چندین و چند مسیریابی وجود دارد.
پارامتر سوم متد routes.MapRoute، یک سری پیش فرض را تعریف می‌کند و این مورد همانجایی است که از اطلاعات آن جهت تعریف کنترلر پیش فرض استفاده کردیم. برای مثال به چهار آدرس زیر دقت نمائید:

```
http://localhost/
http://localhost/home
http://localhost/home/about
http://localhost/process/list
```

در حالت http://localhost/، هر سه مقدار پیش فرض مورد استفاده قرار خواهند گرفت چون سه جزئی ({controller}/{action}/{id}) که موتور مسیریابی به دنبال آن‌ها می‌گردد، در این آدرس وجود خارجی ندارد. بنابراین نام کنترلر پیش فرض در این حالت همان Home مشخص شده در پارامتر سوم متد routes.MapRoute خواهد بود و متد پیش فرض نیز Index و پارامتری هم به آن ارسال نخواهد شد.
در حالت http://localhost/home نام کنترلر مشخص است اما دو جزء دیگر ذکر نشده‌اند، بنابراین مقادیر آن‌ها از پیش فرض‌های صریح ذکر شده در متد routes.MapRoute گرفته می‌شود. یعنی نام متد اکشن مورد پردازش، Index خواهد بود به همراه هیچ آرگومان خاصی.
به علاوه باید خاطرنشان کرد که این مقادیر case sensitive یا حساس به بزرگی و کوچکی حروف نیستند. بنابراین مهم نیست که http://localhost/HoMe باشد یا http://localhost/HOMe یا هر ترکیب دیگری.
یا اگر آدرس رسیده http://localhost/process/list باشد، این مسیریابی پیش فرض تعریف شده قادر به پردازش آن می‌باشد. به این معنا که کنترلری به نام Process وهله سازی و سپس متدی به نام List در آن فراخوانی خواهند شد.

یک نکته:

ترتیب routes.MapRouteهای تعریف شده در اینجا مهم است و اگر اولین URL رسیده با الگوی تعریف شده مطابقت داشت، کار را تمام خواهد کرد و به سایر تعاریف نخواهد رسید. مثلا اگر در اینجا یک مسیریابی دیگر را به نام Process تعریف کنیم:

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
                "Process", // Route name
                "Process/{action}/{id}", // URL with parameters
                new { controller = "Process", action = "List", id = UrlParameter.Optional } //
Parameter defaults
            );

    routes.MapRoute(
```

```
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter
defaults
            );
}
```

آدرسی به فرم http://localhost/Process، به صورت خودکار به کنترلر Process و متد عمومی List آن نگاشت خواهد شد و کار به مسیریابی بعدی نخواهد رسید.

عنوان: ASP.NET MVC #5
نویسنده: وحید نصیری
تاریخ: ۰۹:۵۳:۰۰ ۱۳۹۱/۰۱/۰۹
آدرس: www.dotnettips.info
برچسب‌ها: MVC

بررسی نحوه انتقال اطلاعات از یک کنترلر به Viewهای مرتبط با آن

در ASP.NET Web forms در فایل code behind یک فرم مثلا می‌توان نوشت Label1.Text و سپس مقداری را به آن انتساب داد. اما اینجا به چه ترتیبی می‌توان شبیه به این نوع عملیات را انجام داد؟ با توجه به اینکه در کنترلرها هیچ نوع ارجاع مستقیمی به اشیاء رابط کاربری وجود ندارد و این دو از هم مجزا شده‌اند.
در پاسخ به این سؤال، همان مثال ساده قسمت قبل را ادامه می‌دهیم. یک پروژه جدید خالی ایجاد شده است به همراه HomeController ایی که به آن اضافه کرده‌ایم. همچنین مطابق روشی که ذکر شد، View ایی به نام Index را نیز به آن اضافه کرده‌ایم. سپس برای ارسال اطلاعات از یک کنترلر به View از یکی از روش‌های زیر می‌توان استفاده کرد:

الف) استفاده از اشیاء پویا

ViewBag یک شیء dynamic است که در دات نت 4 امکان تعریف آن میسر شده است. به این معنا که هر نوع خاصیت دلخواهی را می‌توان به این شیء انتساب داد و سپس این اطلاعات در View نیز قابل دسترسی و استخراج خواهد بود. مثلا اگر در اینجا به شیء ViewBag، خاصیت دلخواه Country را اضافه کنیم و سپس مقداری را نیز به آن انتساب دهیم:

```
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.Country = "Iran";
            return View();
        }
    }
}
```

این اطلاعات در View مرتبط با اکشنی به نام Index به نحو زیر قابل بازیابی خواهد بود (نحوه اضافه کردن View متناظر با یک اکشن یا متد را هم در قسمت قبل با تصویر مرور کردیم):

```
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
<p>
    Country : @ViewBag.Country
</p>
```

در این مثال، @ در View engine جاری که Razor نام دارد، به این معنا می‌باشد که این مقداری است که می‌خواهم دریافت کنی (ViewBag.Country) و سپس آن‌را در حین پردازش صفحه نمایش دهی.

ب) انتقال اطلاعات یک شیء کامل و غیر پویا به View

هر پروژه جدید MVC به همراه پوشه‌ای به نام Models است که در آن می‌توان تعاریف اشیاء تجاری برنامه را قرار داد. در پروژه جاری، یک کلاس ساده را به نام Employee به این پوشه اضافه می‌کنیم:

```
namespace MvcApplication1.Models
{
    public class Employee
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
    }
}
```

اکنون برای نمونه یک وهله از این شیء را در متد Index ایجاد کرده و سپس به view متناظر با آن ارسال می‌کنیم (در قسمت return View کد زیر مشخص است). بدیهی است این وهله سازی در عمل می‌تواند از طریق دسترسی به یک بانک اطلاعاتی یا یک وب سرویس و غیره باشد.

```
using System.Web.Mvc;
using MvcApplication1.Models;

namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.Country = "Iran";

            var employee = new Employee
            {
                 Email = "name@site.com",
                 FirstName = "Vahid",
                 LastName = "N."
            };

            return View(employee);
        }
    }
}
```

امضاهای متفاوت (overloads) متد کمکی View هم به شرح زیر هستند:

```
ViewResult View(Object model)
ViewResult View(string viewName, Object model)
ViewResult View(string viewName, string masterName, Object model)
```

اکنون برای دسترسی به اطلاعات این شیء employee در View متناظر با این متد، چندین روش وجود دارد:

```
@{
    ViewBag.Title = "Index";
}
<h2>
```

```
    Index</h2>
<div>
    Country: @ViewBag.Country <br />
    FirstName: @Model.FirstName
</div>
```

می‌توان از طریق شیء استاندارد دیگری به نام Model (که این هم یک شیء dynamic است مانند ViewBag قسمت قبل)، به خواص شیء یا مدل ارسالی به View جاری دسترسی پیدا کرد که یک نمونه از آن‌را در اینجا ملاحظه می‌کنید.
روش دوم، بر اساس **تعریف صریح نوع مدل** است به نحو زیر:

```
@model MvcApplication1.Models.Employee
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
<div>
    Country: @ViewBag.Country
    <br />
    FirstName: @Model.FirstName
</div>
```

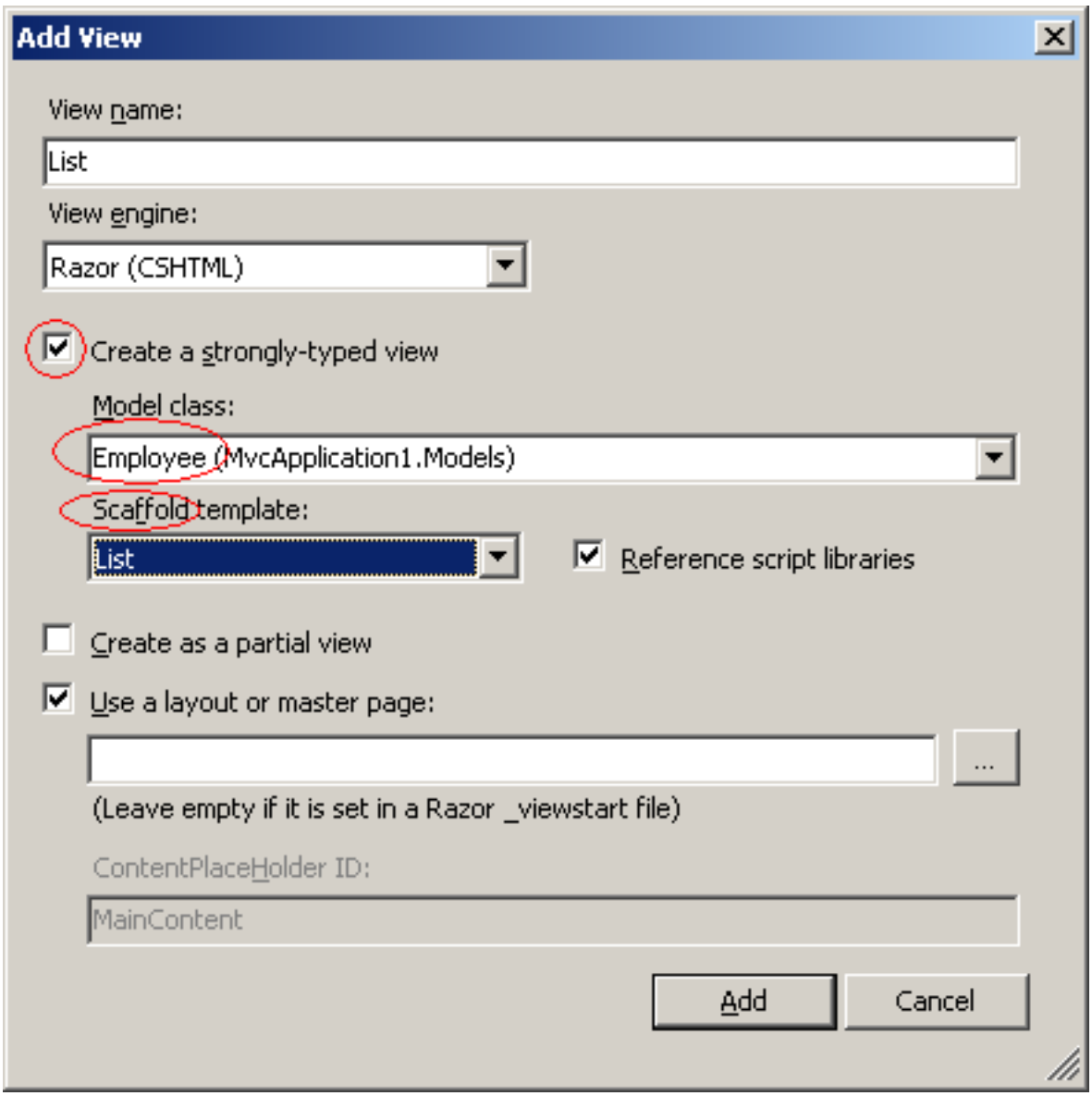
در اینجا در مقایسه با قبل، تنها یک سطر به اول فایل View اضافه شده است که در آن نوع شیء Model تعیین می‌گردد (کلمه model هم در اینجا با حروف کوچک شروع شده است). به این ترتیب اینبار اگر سعی کنیم به خواص این شیء دسترسی پیدا کنیم، Intellisense ویژوال استودیو ظاهر می‌شود. به این معنا که شیء Model بکارگرفته شده اینبار دیگر dynamic نیست و دقیقا می‌داند که چه خواصی را باید پیش از اجرای برنامه در اختیار استفاده کننده قرار دهد.
به این روش، روش **Strongly typed view** هم گفته می‌شود؛ چون View دقیقا می‌داند که چون نوعی را باید انتظار داشته باشد؛ تحت نظر کامپایلر قرار گرفته و همچنین Intellisense نیز برای آن مهیا خواهد بود.
به همین جهت این روش Strongly typed view، در بین تمام روش‌های مهیا، به عنوان روش توصیه شده و مرجح مطرح است.
به علاوه استفاده از Strongly typed views یک مزیت دیگر را هم به همراه دارد: فعال شدن یک code generator توکار در VS.NET به نام **scaffolding** . یک مثال ساده:
تا اینجا ما اطلاعات یک کارمند را نمایش دادیم. اگر بخواهیم یک لیست از کارمندها را نمایش دهیم چه باید کرد؟
روش کار با قبل تفاوتی نمی‌کند. اینبار در return View ما، یک شیء لیستی ارائه خواهد شد. در سمت View هم با یک حلقه foreach کار نمایش این اطلاعات صورت خواهد گرفت. راه ساده‌تری هم هست. اجازه دهیم تا خود VS.NET، کدهای مرتبط را برای ما تولید کند.
یک کلاس دیگر به پوشه مدل‌های برنامه اضافه کنید به نام Employees با محتوای زیر:

```
using System.Collections.Generic;

namespace MvcApplication1.Models
{
    public class Employees
    {
        public IList<Employee> CreateEmployees()
        {
            return new[]
                {
                    new Employee { Email = "name1@site.com", FirstName = "name1", LastName =
"LastName1" },
                    new Employee { Email = "name2@site.com", FirstName = "name2", LastName =
"LastName2" },
                    new Employee { Email = "name3@site.com", FirstName = "name3", LastName =
"LastName3" }
                };
        }
    }
}
```

سپس متد جدید زیر را به کنترلر Home اضافه کنید.

```
public ActionResult List()
{
    var employeesList = new Employees().CreateEmployees();
    return View(employeesList);
}
```

برای اضافه کردن View متناظر با آن، روی نام متد کلیک راست کرده و گزینه Add view را انتخاب کنید. در صفحه ظاهر شده:



شکل ۱

تیک مربوط به Create a strongly typed view را قرار دهید. سپس در قسمت Model class، کلاس Employee را انتخاب کنید (نه Employees جدید را، چون از آن می‌خواهیم به عنوان منبع داده لیست تولیدی استفاده کنیم). اگر این کلاس را مشاهده نمی‌کنید، به

این معنا است که هنوز برنامه را یکبار کامپایل نکرده‌اید تا VS.NET بتواند با اعمال Reflection بر روی اسمبلی برنامه آن‌را پیدا کند. سپس در قسمت Scaffold template گزینه List را انتخاب کنید تا Code generator توکار VS.NET فعال شود. اکنون بر روی دکمه Add کلیک نمائید تا View نهایی تولید شود. برای مشاهده نتیجه نهایی مسیر http://localhost/Home/List باید بررسی گردد.

ج) استفاده از ViewDataDictionary

ViewDataDictionary از نوع IDictionary با کلیدی رشته‌ای و مقداری از نوع object است. توسط آن شیءایی به نام ViewData در ASP.NET MVC به نحو زیر تعریف شده است:

```
public ViewDataDictionary ViewData { get; set; }
```

این روش در نگارش‌های اولیه ASP.NET MVC بیشتر مرسوم بود. برای مثال:

```
using System;
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewData["DateTime"] = "<br/>" + DateTime.Now;
            return View();
        }
    }
}
```

و سپس جهت استفاده از این ViewData تعریف شده با کلید دلخواهی به نام DateTime در View متناظر با اکشن Index خواهیم داشت:

```
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
<div>
    DateTime: @ViewData["DateTime"]
</div>
```

یک نکته امنیتی:

اگر به مقدار انتساب داده شده به شیء ViewDataDictionary دقت کنید، یک تگ br هم به آن اضافه شده است. برنامه را یکبار اجرا کنید. مشاهده خواهید کرد که این تگ به همین نحو نمایش داده می‌شود و نه به صورت یک سطر جدید HTML . چرا؟ چون Razor به صورت پیش فرض اطلاعات را encode شده (فراخوانی متد Html.Encode در پشت صحنه به صورت خودکار) در صفحه نمایش می‌دهد و این مساله از لحاظ امنیتی بسیار عالی است؛ زیرا جلوی بسیاری از حملات cross site scripting یا XSS را خواهد گرفت.

احتمالا الان این سؤال پیش خواهد آمد که اگر «عالمانه» بخواهیم این رفتار نیکوی پیش فرض را غیرفعال کنیم چه باید کرد؟

برای این منظور می‌توان نوشت:

```
@Html.Raw(myString)
```

و یا:

```
<div>@MvcHtmlString.Create("<h1>HTML</h1>")</div>
```

به این ترتیب خروجی Razor دیگر encode شده نخواهد بود.

د) استفاده از TempData

TempData نیز یک dictionary دیگر برای ذخیره سازی اطلاعات است و به نحو زیر در فریم ورک تعریف شده است:

```
public TempDataDictionary TempData { get; set; }
```

TempData در پشت صحنه از سشن‌های ASP.NET جهت ذخیره سازی اطلاعات استفاده می‌کند. بنابراین اطلاعات آن در سایر کنترلرها و View ها نیز در دسترس خواهد بود. البته TempData یک سری تفاوت هم با سشن معمولی ASP.NET دارد:
- بلافاصله پس از خوانده شدن، حذف خواهد شد.
- پس از پایان درخواست از بین خواهد رفت.

هر دو مورد هم به جهت بالابردن کارآیی برنامه‌های ASP.NET MVC و مصرف کمتر حافظه سرور درنظر گرفته شده‌اند.
البته کسانی که برای بار اول هست با ASP.NET مواجه می‌شوند، شاید سؤال بپرسند این مسایل چه اهمیتی دارد؟ پروتکل HTTP، ذاتا یک پروتکل «بدون حالت» است یا Stateless هم به آن گفته می‌شود. به این معنا که پس از ارائه یک صفحه وب توسط سرور، تمام اشیاء مرتبط با آن در سمت سرور تخریب خواهند شد. این مورد متفاوت است با برنامه‌های معمولی دسکتاپ که طول عمر یک شیء معمولی تعریف شده در سطح فرم به صورت یک فیلد، تا زمان باز بودن آن فرم، تعیین می‌گردد و به صورت خودکار از حافظه حذف نمی‌شود. این مساله دقیقا مشکل تمام تازه واردها به دنیای وب است که چرا اشیاء ما نیست و نابود شدند. در اینجا وب سرور قرار است به هزاران درخواست رسیده پاسخ دهد. اگر قرار باشد تمام این اشیاء را در سمت سرور نگهداری کند، خیلی زود با اتمام منابع مواجه می‌گردد. اما واقعیت این است که نیاز است یک سری از اطلاعات را در حافظه نگه داشت. به همین منظور یکی از چندین روش مدیریت حالت در ASP.NET استفاده از سشن‌ها است که در اینجا به نحو بسیار مطلوبی، با سربار حداقل توسط TempData مدیریت شده است.

یک مثال کاربردی در این زمینه:

فرض کنید در متد جاری کنترلر، ابتدا بررسی می‌کنیم که آیا ورودی دریافتی معتبر است یا خیر. در غیراینصورت، کاربر را به یک View دیگر از طریق کنترلری دیگر جهت نمایش خطاها هدایت خواهیم کرد.
همین «هدایت مرورگر به یک View دیگر» یعنی پاک شدن و تخریب اطلاعات کنترلر قبلی به صورت خودکار. بنابراین نیاز است این اطلاعات را در TempData قرار دهیم تا در کنترلری دیگر قابل استفاده باشد:

```
using System;
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult InsertData(string name)
        {
            // Check for input errors.
            if (string.IsNullOrWhiteSpace(name))
            {
                TempData["error"] = "name is required.";
                return RedirectToAction("ShowError");
            }
```

```
            // No errors
            // ...
            return View();
        }

        public ActionResult ShowError()
        {
            var error = TempData["error"] as string;
            if (!string.IsNullOrWhiteSpace(error))
            {
                ViewBag.Error = error;
            }
            return View();
        }
    }
}
```

در همان HomeController دو متد جدید به نام‌های InsertData و ShowError اضافه شده‌اند. در متد InsertData ابتدا بررسی می‌شود که آیا نامی وارد شده است یا خیر. اگر خیر توسط متد RedirectToAction، کاربر به اکشن یا متد ShowError هدایت خواهد شد.
برای انتقال اطلاعات خطایی که می‌خواهیم در حین این Redirect نمایش دهیم نیز از TempData استفاده شده است.
بدیهی است برای اجرا این مثال نیاز است دو View جدید برای متدهای InsertData و ShowError ایجاد شوند (کلیک راست روی نام متد و انتخاب گزینه Add view برای اضافه کردن View مرتبط با آن اکشن).
محتوای View مرتبط با متد افزودن اطلاعات فعلا مهم نیست، ولی View نمایش خطاها در ساده‌ترین حالت مثلا می‌تواند به صورت زیر باشد:

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

<h2>Error</h2>

@ViewBag.Error
```

برای آزمایش برنامه هم مطابق مسیریابی پیش فرض و با توجه به قرار داشتن در کنترلری به نام Home، مسیر http://localhost/Home/InsertData ابتدا باید بررسی شود. چون آرگومانی وارد نشده، بلافاصله صفحه به آدرس http://localhost/Home/ShowError به صورت خودکار هدایت خواهد شد.

نکته‌ای تکمیلی در مورد Strongly typed viewها:

عنوان شد که Strongly typed view روش مرجح بوده و بهتر است از آن استفاده شود، زیرا اطلاعات اشیاء و خواص تعریف شده در یک View تحت نظر کامپایلر قرار می‌گیرند که بسیار عالی است. یعنی اگر در View بنویسم @Model.FirstName1 :FirstName چون FirstName1 وجود خارجی ندارد، برنامه نباید کامپایل شود. یکبار این را بررسی کنید. برنامه بدون مشکل کامپایل می‌شود! اما تنها در زمان اجرا است که صفحه زرد رنگ معروف خطاهای ASP.NET ظاهر می‌شود که چنین خاصیتی وجود ندارد (این حالت پیش فرض است؛ یعنی کامپایل یک View در زمان اجرا). البته این باز هم خیلی بهتر است از ViewBag، چون اگر مثلا ViewBag.Country1 را وارد کنیم، در زمان اجرا تنها چیزی نمایش داده نخواهد شد؛ اما با روش Strongly typed view، حتما خطای Compilation Error به همراه نمایش محل مشکل نهایی، در صفحه ظاهر خواهد شد.
سؤال: آیا می‌شود پیش از اجرای برنامه هم این بررسی را انجام داد؟
پاسخ: بله. باید فایل پروژه را اندکی ویرایش کرده و مقدار MvcBuildViews را که به صورت پیش فرض false هست، true نمود. یا خارج از ویژوال استودیو با یک ادیتور متنی ساده مثلا فایل csproj را گشوده و این تغییر را انجام دهید. یا داخل ویژوال استودیو، بر روی نام پروژه کلیک راست کرده و سپس گزینه Unload Project را انتخاب کنید. مجددا بر روی این پروژه Unload شده کلیک راست نموده و گزینه edit را انتخاب نمائید. در صفحه باز شده، MvcBuildViews را یافته و آن‌را true کنید. سپس پروژه را Reload کنید.

اکنون اگر پروژه را کامپایل کنید، پیغام خطای زیر پیش از اجرای برنامه قابل مشاهده خواهد بود:

```
'MvcApplication1.Models.Employee' does not contain a definition for 'FirstName1'
and no extension method 'FirstName1' accepting a first argument of type
'MvcApplication1.Models.Employee'
could be found (are you missing a using directive or an assembly reference?)
d:\Prog\MvcApplication1\MvcApplication1\Views\Home\Index.cshtml 10 MvcApplication1
```

البته بدیهی است این تغییر، زمان Build پروژه را مقداری افزایش خواهد داد؛ اما امن‌ترین حالت ممکن برای جلوگیری از این نوع خطاهای تایپی است.
یا حداقل بهتر است یکبار پیش از ارائه نهایی برنامه این مورد فعال و بررسی شود.

و یک خبر خوب!

مجوز سورس کد ASP.NET MVC از MS-PL به Apache تغییر کرده و همچنین Razor و یک سری موارد دیگر هم سورس باز شده‌اند. این تغییرات به این معنا خواهند بود که پروژه از حالت فقط خواندنی MS-PL به حالت متداول یک پروژه سورس باز که شامل دریافت تغییرات و وصله‌ها از جامعه برنامه نویس‌ها است، تغییر کرده است (^ و ^).

عنوان: **رفع مشکل نصب به روز رسانی ASP.NET MVC 3**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۰ ۱۴:۲۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

در مورد نحوه نصب پیشنیازهای ASP.NET MVC 3 پیشتر توضیح داده شد. یک روش دیگر هم برای اینکار مهیا است؛ اگر به مشکل برخوردید حین نصب.
دریافت ASP.NET MVC 3 RTM (نگارش RTM یعنی نگارش نهایی ارائه شده به صنعت)
دریافت ASP.NET MVC 3 Tools Update (این هم شامل یک سری به روز رسانی است؛ مثلا قالب اینترانت به آن اضافه شده و غیره)

اگر حین نصب ASP.NET MVC 3 Tools Update، کارها خوب پیش نرفت و دست آخر زمانیکه فایل log خطا را باز کردید در پایان آن ذکر شده بود « Installation failed with error code: 0x80070643» باید این مراحل را طی کنید:
الف) فایل نصاب را با استفاده از برنامه zip-7 آنپک کنید (فایل‌های داخلی آن‌را استخراج کنید).
ب) فایل VS10-KB*.msp یا vs10-kb*.msi را یافته و بر روی آن کلیک راست کنید. گزینه install را از منوی باز شده انتخاب نمائید تا کار نصب شروع شود.
ج) در حین نصب این پیغام را دریافت خواهید کرد: «can't find vs_setup.msi». در این حالت بر روی دکمه browse کلیک کرده و این فایل vs_setup.msi را که در DVD نصب خود VS 2010 اصلی وجود دارد به آن معرفی کنید. علت هم به این بر می‌گردد که پروسه نصب VS 2010 شما از ابتدا ناقص بوده است و نیاز است یک سری فایل دیگر در ابتدا بر روی سیستم نصب گردند.
اکنون کار نصب بدون مشکل پیش خواهد رفت. لازم به ذکر است که این پیغام خطا را حین عملیات نصب معمولی MVC3 «دریافت نخواهید کرد».
د) سپس یکبار دیگر هم فایل setup.exe اصلی را بدون مراحل فوق اجرا کنید تا خیالتان از این بابت راحت شود و تمام موارد نصب نشده نیز نصب گردند (مهم!).

این مراحل باید مشکل را حل کنند، در غیراینصورت یک سری راه حل دیگر هم در اینجا ذکر شده است:
http://support.microsoft.com/kb/2531566
و خلاصه آن این است که فایل C:\Windows\Microsoft.NET\Framework\v4.0.30319\web.config را پیش از نصب rename کنید و اجازه دهید تا نصاب یک نمونه جدید را ایجاد کند.

عنوان: **ASP.NET MVC #6**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۱ ۰۰:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

آشنایی با انواع ActionResult

در قسمت چهارم، اولین متد یا اکشنی که به صورت خودکار توسط VS.NET به برنامه اضافه شد، اینچنین بود:

```
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

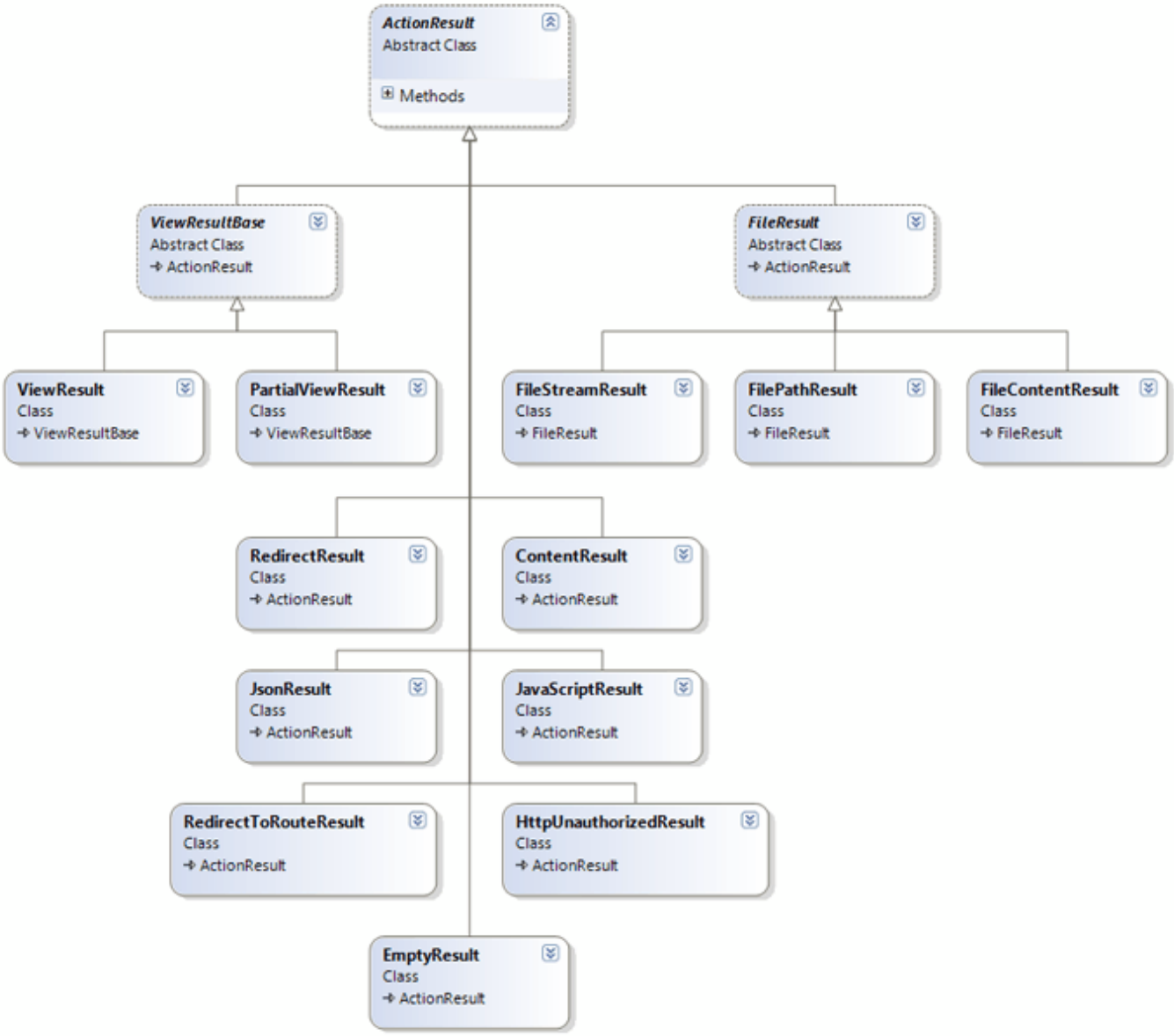
توضیحات تکمیلی مرتبط با خروجی از نوع ActionResult ایی را که مشاهده می‌کنید، در این قسمت ارائه خواهد شد.
رفتار یک کنترلر توسط متدهایی که در آن کلاس تعریف می‌شوند، مشخص می‌گردد. هر متد هم از طریق یک URL مجزا قابل دسترسی و فراخوانی خواهد بود. این متدها که به آن‌ها اکشن نیز گفته می‌شود باید عمومی بوده، استاتیک یا متد الحاقی (extension method) نباشند و همچنین دارای پارامترهایی از نوع ref و out نیز نباشند.
هر درخواست رسیده، به یک کنترلر و متدی عمومی در آن توسط سیستم مسیریابی، نگاشت خواهد شد. اگر علاقمند باشید که در یک کلاس کنترلر، متدی عمومی را از این سیستم خارج کنید، تنها کافی است آن‌را با ویژگی (attribute) به نام NonAction مزین کنید:

```
using System.Web.Mvc;

namespace MvcApplication2.Controllers
{
    public class HomeController : Controller
    {
        [NonAction]
        public string ShowData()
        {
            return "Text";
        }

        public ActionResult Index()
        {
            ViewBag.Message = string.Format("{0}/{1}/{2}",
                                            RouteData.Values["controller"],
                                            RouteData.Values["action"],
                                            RouteData.Values["id"]);
            return View();
        }

        public ActionResult Search(string data = "*")
        {
            // do something ...
            return View();
        }
    }
}
```

چند نکته در این مثال قابل ذکر است:
الف) در اینجا اگر شخصی آدرس http://localhost/home/showdata را درخواست نماید، با توجه به استفاده از ویژگی NonAction، با پیغام یافت نشد یا 404 مواجه می‌گردد.
ب) صرفنظر از پارامترهای یک متد و ساختار کلاس جاری، اطلاعات مسیریابی از طریق شیء RouteData.Values نیز در دسترس هستند که نمونه‌ای از آن‌را در اینجا بر اساس مقادیر پیش فرض تعاریف مسیریابی یک پروژه ASP.NET MVC ملاحظه می‌نمائید.
ج) در متد Search، از قابلیت امکان تعریف مقداری پیش فرض جهت آرگومان‌ها در سی شارپ 4 استفاده شده است. به این ترتیب اگر شخصی آدرس http://localhost/home/search را وارد کند، چون پارامتری را ذکر نکرده است، به صورت خودکار از مقدار پیش فرض آرگومان data استفاده می‌گردد.

انواع Action Results در ASP.NET MVC

در ASP.NET MVC بجای استفاده مستقیم از شیء Response، از شیء ActionResult جهت ارائه خروجی یک متد استفاده می‌شود و مهم‌ترین دلیل آن هم مشکل بودن نوشتن آزمون‌های واحد برای شیء Response است که وهله سازی آن مساوی است با به کار اندازی موتور ASP.NET و Http Runtime آن توسط یک وب سرور (بنابراین در ASP.NET MVC سعی کنید شیء Response را فراموش کنید).
سلسه مراتب ActionResultهای قابل استفاده در ASP.NET در تصویر زیر مشخص شده‌اند:



شکل ۱

و در مثال زیر تقریبا انواع و اقسام ActionResultهای مهم و کاربردی ASP.NET MVC را می‌توانید مشاهده کنید:

```
using System.Web.Mvc;

namespace MvcApplication2.Controllers
{
    public class ActionResultsController : Controller
    {
        //http://localhost/actionresults/welcome
        public string Welcome()
        {
            return "Hello, World";
        }

        //http://localhost/actionresults/index
        public ActionResult Index() // or ContentResult
        {
            return Content("Hello, World");
        }
```

```
        //http://localhost/actionresults/SendMail
        public void SendMail()
        {
        }

        public ActionResult SendMailCompleted() // or EmptyResult
        {
            // do whatever
            return new EmptyResult();
        }

        public ActionResult GetFile() // or FilePathResult
        {
            return File(Server.MapPath("~/content/site.css"), "text/css", "mySite.css");
        }

        public ActionResult UnauthorizedStatus() // or HttpStatusCodeResult/HttpUnauthorizedResult
        {
            return new HttpUnauthorizedResult("You need to login first.");
        }

        public ActionResult Status() // or HttpStatusCodeResult
        {
            return new HttpStatusCodeResult(501, "Server Error");
        }

        public ActionResult GetJavaScript() // or JavaScriptResult
        {
            return JavaScript("...JavaScript...");
        }

        public ActionResult GetJson() // or JsonResult
        {
            var obj = new { prop1 = 1, prop2 = "data" };
            return Json(obj, JsonRequestBehavior.AllowGet);
        }

        public ActionResult RedirectTo() // or RedirectResult
        {
            return RedirectPermanent("http://www.site.com");
            //return RedirectToAction("Home", "Index");
        }

        public ActionResult ShowView() // or ViewResult
        {
            return View();
        }
    }
}
```

چند نکته در این مثال وجود دارد:

1) مثلا متد GetJavaScript را درنظر بگیرید. در این متد خاص، چه بنویسید public ActionResult GetJavaScript یا بنویسید public JavaScriptResult GetJavaScript تفاوتی نمی‌کند. در سایر موارد هم به همین ترتیب است. علت را در تصویر سلسله مراتبی ActionResultها می‌توان جستجو کرد. تمام این کلاس‌ها نوعی ActionResult هستند و از یک کلاس پایه به ارث رسیده‌اند.

2) مثلا ContentResult شبیه به همان Response.Write سابق ASP.NET عمل می‌کند. علت وجودی آن هم عدم وابستگی مستقیم به شیء Response و ساده‌تر سازی نوشتن آزمون‌های واحد برای این نوع اکشن متدها است.

3) منهای متد آخری که نمایش داده شده (ShowView)، هیچکدام از متدهای دیگر نیازی به View متناظر ندارند. یعنی نیازی نیست تا روی متد کلیک راست کرده و Add view را انتخاب کنیم. چون در همین متد کنترلر، کار Response به پایان می‌رسد و مرحله بعدی ندارد. مثلا در حالت return File، یک فایل به درون مرورگر کاربر Flush خواهد شد و تمام.

4) متد Welcome و متد Index در اینجا به یک صورت تفسیر می‌شوند. به این معنا که اگر خروجی متد تعریف شده در یک کنترلر از نوع ActionResult نباشد، به صورت پیش فرض درون یک ContentResult محصور خواهد شد.

5) اگر خروجی متدی در اینجا از نوع void باشد، با ActionResult ایی به نام EmptyResult یکسان خواهد بود. بنابراین با متدهای SendMail و SendMailCompleted به یک نحو رفتار می‌گردد.

6) return Json یاد شده که خروجی‌اش از نوع JsonResultاست در پیاده سازی‌های Ajax ایی کاربرد دارد.

7) جهت بازگرداندن حالت وضعیت 403 یا غیرمجاز می‌توان از return new HttpUnauthorizedResult استفاده کرد.

8) یا جهت اعلام مشکلی در سمت سرور به کمک return new HttpStatusCodeResultکد ویژه‌ای را می‌توان به کاربر نمایش داد.
9) به کمک return RedirectToAction می‌توان به یک کنترلر و متدی خاص در آن، کاربر را هدایت کرد.

و خلاصه اینکه تمام کارهایی را که پیشتر در ASP.NET Web forms ، مستقیما به کمک شیء Response انجام می‌دادید (Response.Write، Response.End، Response.Redirect و غیره)، اینبار به کمک یکی از ActionResultهای یاد شده انجام دهید تا بتوان بدون نیاز به راه اندازی یک وب سرور، برای متدهای کنترلرها آزمون واحد نوشت. برای مثال:

```
[TestMethod]
public void TestMethod1()
{
    // Arrange
    var controller = new ActionResultsController();

    // Act
    var result = controller.Index() as ContentResult;

    // Assert
    Assert.NotNull(result);
    Assert.AreEqual( "Hello, World", result.Content);
}
```

عنوان: **ASP.NET MVC #7**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۲ ۱۰:۵۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

آشنایی با Razor Views

قبل از اینکه بحث جاری ASP.NET MVC را بتوانیم ادامه دهیم و مثلا مباحث دریافت اطلاعات از کاربر، کار با فرم‌ها و امثال آن‌را بررسی کنیم، نیاز است حداقل به دستور زبان یکی از View Engineهای ASP.NET MVC آشنا باشیم.
MVC3 موتور View جدیدی را به نام Razor معرفی کرده است که به عنوان روش برگزیده ایجاد Viewها در این سیستم به شمار می‌رود و فوق العاده نسبت به ASPX view engine سابق، زیباتر، ساده‌تر و فشرده‌تر طراحی شده است و یکی از اهداف آن تلفیق code و markup می‌باشد. در این حالت دیگر پسوند فایل‌های Viewها همانند سابق ASPX نخواهد بود و به cshtml و یا vbhtml تغییر یافته است. همچنین برخلاف web forms view engine از System.Web.Page مشتق نشده است. و باید دقت داشت که Razor یک زبان برنامه نویسی جدید نیست. در اینجا از مخلوط زبان‌های سی شارپ و یا ویژوال بیسیک به همراه تگ‌های html استفاده می‌شود.
البته این را هم باید عنوان کرد که این مسایل سلیقه‌ای است. اگر با web forms view engine راحت هستید، با همان کار کنید. اگر با هیچکدام از این‌ها راحت نیستید (!) نمونه‌های دیگر هم وجود دارند ، مثلا:

Spark
NHaml
SharpDOM
SharpTiles
Wing Beats
string-template-view-engine-mvc
Bellevue
Brail
Hasic
NDjango

Razor Views یک سری قابلیت جالب را هم به همراه دارند:
1) امکان کامپایل آن‌ها به درون یک DLL وجود دارد. مزیت: استفاده مجدد از کد، عدم نیاز به وجود صریح فایل cshtml یا vbhtml بر روی دیسک سخت.
2) آزمون پذیری: از آنجائیکه Razor viewها به صورت یک کلاس کامپایل می‌شوند و همچنین از System.Web.Page مشتق نخواهند شد، امکان بررسی HTML نهایی تولیدی آن‌هابدون نیاز به راه اندازی یک وب سرور وجود دارد.
3) IntelliSense ویژوال استودیو به خوبی آن‌را پوشش می‌دهد.
4) با توجه به مواردی که ذکر شد، یک اتفاق جالب هم رخ داده است: امکان استفاده از Razor engine خارج از ASP.NET MVC هم وجود دارد. برای مثال یک سرویس ویندوز NT طراحی کرده‌اید که قرار است ایمیل فرمت شده‌ای به همراه اطلاعات مدل‌های شما را در فواصل زمانی مشخص ارسال کند؟ می‌توانید برای طراحی آن از Razor engine استفاده کنید و تهیه خروجی نهایی HTML آن نیازی به راه اندازی وب سرور و وهله سازی HttpContext ندارد.

ساختار پروژه مثال جاری

در ادامه مرور سریعی خواهیم داشت بر دستور زبان Razor engine و جهت نمایش این قابلیت‌ها، یک مثال ساده را در ابتدا با مشخصات زیر ایجاد خواهیم کرد:
الف) یک ASP.NET MVC 3 project empty را ایجاد کنید و نوع View engine را هم در ابتدای کار Razor انتخاب نمائید.

ب) دو کلاس زیر را به پوشه مدل‌های برنامه اضافه کنید:

```
namespace MvcApplication3.Models
{
    public class Product
    {
        public Product(string productNumber, string name, decimal price)
        {
            Name = name;
            Price = price;
            ProductNumber = productNumber;
        }
        public string ProductNumber { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
    }
}
```

```
using System.Collections.Generic;

namespace MvcApplication3.Models
{
    public class Products : List<Product>
    {
        public Products()
        {
            this.Add(new Product("D123", "Super Fast Bike", 1000M));
            this.Add(new Product("A356", "Durable Helmet", 123.45M));
            this.Add(new Product("M924", "Soft Bike Seat", 34.99M));
        }
    }
}
```

کلاس Products صرفا یک منبع داده تشکیل شده در حافظه است. بدیهی است هر نوع ORM ایی که یک ToList را بتواند در اختیار شما قرار دهد، توانایی تشکیل لیست جنریکی از محصولات را نیز خواهد داشت و تفاوتی نمی‌کند که کدامیک مورد استفاده قرار گیرد.

ج) سپس یک کنترلر جدید به نام ProductsController را به پوشه Controllers برنامه اضافه می‌کنیم:

```
using System.Web.Mvc;
using MvcApplication3.Models;

namespace MvcApplication3.Controllers
{
    public class ProductsController : Controller
    {
        public ActionResult Index()
        {
            var products = new Products();
            return View(products);
        }
    }
}
```

د) بر روی نام متد Index کلیک راست کرده، گزینه Add view را جهت افزودن View متناظر آن، انتخاب کنید. البته می‌شود همانند قسمت پنجم گزینه Create a strongly typed view را انتخاب کرد و سپس Product را به عنوان کلاس مدل انتخاب نمود و در آخر خیلی سریع یک لیست از محصولات را نمایش داد، اما فعلا از این قسمت صرفنظر نمائید، چون می‌خواهیم آن را دستی ایجاد کرده و توضیحات و نکات بیشتری را بررسی کنیم.

ه) برای اینکه حین اجرای برنامه در VS.NET هربار نخواهیم که آدرس کنترلر Products را دستی در مرورگر وارد کنیم، فایل

Global.asax.cs را گشوده و سپس در متد RegisterRoutes، در سطر Parameter defaults، مقدار پیش فرض کنترلر را مساوی Products قرار دهید.

مرجع سریع Razor

ابتدا کدهای View متد Index را به شکل زیر وارد نمائید:

```
@model List<MvcApplication3.Models.Product>
@{
    ViewBag.Title = "Index";
    var number = 12;
    var data = "some text...";
    <h2>line1: @data</h2>

    @:line-2: @data <br />
    <text>line-3:</text> @data
}
<br />
site@(data)
<br />
@@name
<br />
@(number/10)
<br />
First product: @Model.First().Name
<br />
@if (@number>10)
{
  <span>@data</span>
}
else
{
  <text>Plain Text</text>
}
<br />
@foreach (var item in Model)
{
    <li>@item.Name, $@item.Price </li>
}

@*
    A Razor Comment
*@

<br />
@("First product: " + Model.First().Name)
<br />
<img src="@(number).jpg" />
```

در ادامه توضیحات مرتبط با این کدها ارائه خواهد شد:

1) نحوه معرفی یک قطعه کد

```
@model List<MvcApplication3.Models.Product>
@{
    ViewBag.Title = "Index";
    var number = 12;
    var data = "some text...";
    <h2>line1: @data</h2>

    @:line-2: @data <br />
    <text>line-3:</text> @data
}
```

این کدها متعلق به Viewایی است که در قسمت (د) بررسی ساختار پروژه مثال جاری، ایجاد کردیم. در ابتدای آن هم نوع model

مشخص شده تا بتوان ساده‌تر به اطلاعات شیء Model به کمک IntelliSense دسترسی داشت.
برای ایجاد یک قطعه کد در Viewایی از نوع Razor به این نحو عمل می‌شود:

```
@{  ...Code Block.... }
```

در اینجا مجاز هستیم کدهای سی شارپ را وارد کنیم. یک نکته جالب را هم باید درنظر داشت: امکان نوشتن تگ‌های html هم در این بین وجود دارد (بدون اینکه مجبور باشیم قطعه کد شروع شده را خاتمه دهیم، به حالت html معمولی سوئیچ کرده و دوباره یک قطعه کد دیگر را شروع نمائیم). مانند line1 مثال فوق. اگر کمی پایین‌تر از این سطر مثلا بنویسیم line2 (به عنوان یک برچسب) کامپایلر ایراد خواهد گرفت، زیرا این مورد نه متغیر است و نه از پیش تعریف شده است. به عبارتی نباید فراموش کنیم که اینجا قرار است کد نوشته شود. برای رفع این مشکل دو راه حل وجود دارد که در سطرهای دو و سه ملاحظه می‌کنید. یا باید از تگی به نام text برای معرفی یک برچسب در این میان استفاده کرد (سطر سه) یا اگر قرار است اطلاعاتی به شکل یک متن معمولی پردازش شود ابتدای آن مانند سطر دوم باید یک @: قرار گیرد.
کمی پایین‌تر از قطعه کد معرفی شده در بالا بنویسید:

```
<br />
site@data
```

اکنون اگر خروجی این View را در مرورگر بررسی کنید، دقیقا همین site@data خواهد بود. چون در این حالت Razor تصور خواهد کرد که قصد داشته‌اید یک آدرس ایمیل را وارد کنید. برای این حالت خاص باید نوشت:

```
<br />
site@(data)
```

به این ترتیب data از متغیر data تعریف شده در code block قبلی برنامه دریافت و نمایش داده خواهد شد.
شبیه به همین حالت در مثال زیر هم وجود دارد:

```
<img src="@(number).jpg" />
```

در اینجا اگر پرانتزها را حذف کنیم، Razor فرض را بر این خواهد گذاشت که شیء number دارای خاصیت jpg است. بنابراین باید به نحو صریحی، بازه کاری را مشخص نمائیم.

بکار گیری این علامت @ یک نکته جنبی دیگر را هم به همراه دارد. فرض کنید در صفحه قصد دارید آدرس توئیتری شخصی را وارد کنید. مثلا:

```
<br />
@name
```

در این حالت View کامپایل نخواهد شد و Razor تصور خواهد کرد که قرار است اطلاعات متغیری به نام name را نمایش دهید. برای نمایش این اطلاعات به همین شکل، یک @ دیگر به ابتدای سطر اضافه کنید:

```
<br />
@@name
```

2) نحوه معرفی عبارات

عبارات پس از علامت @ معرفی می‌شوند و به صورت پیش فرض Html Encoded هستند (در قسمت 5 در اینباره بیشتر توضیح داده شد):

```
First product: @Model.First().Name
```

در این مثال با توجه به اینکه نوع مدل در ابتدای View مشخص شده است، شیء Model به لیستی از Products اشاره می‌کند.

یک نکته:
مشخص سازی حد و مرز صریح یک متغیر در مثال زیر نیز کاربرد دارد:

```
<br />
@number/10
```

اگر خروجی این مثال را بررسی کنید مساوی 10/12 خواهد بود و محاسبه‌ای انجام نخواهد شد. برای حل این مشکل باز هم از پرانتز می‌توان کمک گرفت:

```
<br />
@(number/10)
```

3) نحوه معرفی عبارات شرطی

```
@if (@number>10)
{
  <span>@data</span>
}
else
{
  <text>Plain Text</text>
}
```

یک عبارت شرطی در اینجا با @if شروع می‌شود و سپس نکاتی که در «نحوه معرفی یک قطعه کد» بیان شد، در مورد عبارات داخل {} صادق خواهد بود. یعنی در اینجا نیز می‌توان عبارات سی شارپ مخلوط با تگ‌های html را نوشت.
یک نکته: عبارت شرطی زیر نادرست است. حتما باید سطرهای کدهای سی شارپ بین {} محصور شوند؛ حتی اگر یک سطر باشند:

```
@if( i < 1 ) int myVar=0;
```

4) نحوه استفاده از حلقه foreach

```
@foreach (var item in Model)
{
    <li>@item.Name, $@item.Price </li>
}
```

حلقه foreach نیز مانند عبارات شرطی با یک @ شروع شده و داخل {} بدنه آن نکات «نحوه معرفی یک قطعه کد» برقرار هستند (امکان تلفیق code و markup با هم).
کسانی که پیشتر با web forms کار کرده باشند، احتمالا الان خواهند گفت که این یک پس رفت است و بازگشت به دوران ASP کلاسیک دهه نود! ما به ندرت داخل صفحات aspx وب فرم‌ها کد می‌نوشتیم. مثلا پیشتر یک GridView وجود داشت و یک دیتاسورس که به آن متصل می‌شد؛ مابقی خودکار بود و ما هیچ وقت حلقه‌ای ننوشتیم. در اینجا هم این مساله با نوشتن برای مثال «html helpers» قابل کنترل است که در قسمت‌های بعدی به آن پرداخته خواهد شد. به عبارتی قرار نیست به این نحو با Viewهای Razor رفتار کنیم. این قسمت فقط یک آشنایی کلی با Syntax است.

5) امکان تعریف فضای نام در ابتدای View

```
@using namespace;
```

6) نحوه نوشتن توضیحات سمت سرور:

```
@*
    A Razor Comment / Server side Comment
*@
```

7) نحوه معرفی عبارات چند جزئی:

```
@("First product: " + Model.First().Name)
```

همانطور که ملاحظه می‌کنید، ذکر یک پرانتز برای معرفی عبارات چندجزئی کفایت می‌کند.

استفاده از موتور Razor خارج از ASP.NET MVC

پیشتر مطلبی را در مورد « تهیه قالب برای ایمیل‌های ارسالی یک برنامه ASP.Net » در این سایت مطالعه کرده‌اید. اولین سؤالی هم که در ذیل آن مطلب مطرح شده این است: «در برنامه‌های ویندوز چطور؟» پاسخ این است که کل آن مثال بر مبنای HttpContext.Current.Server.Execute کار می‌کند. یعنی باید مراحل وهله سازی HttpContext و شیء Server توسط یک وب سرور و درخواست رسیده طی شود و ... شبیه سازی آن آنچنان مرسوم و کار ساده‌ای نیست.
اما این مشکل با Razor وجود ندارد. به عبارتی در اینجا برای رندر کردن یک Razor View به html نهایی، نیازی به HttpContext نیست. بنابراین از این امکانات مثلا در یک سرویس ویندوز ان تی یا یک برنامه کنسول، WPF، WinForms و غیره هم می‌توان استفاده کرد.
برای اینکه بتوان از Razor خارج از ASP.NET MVC استفاده کرد، نیاز به اندکی کدنویسی هست مثلا استفاده از کامپایلر سی شارپ یا وی بی و کامپایل پویای کد و یک سری ست آپ دیگر. پروژه‌ای به نام RazorEngine این کپسوله سازی رو انجام داده و از اینجا http://razorengine.codeplex.com/ قابل دریافت است.

عنوان: **ASP.NET MVC #8**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۳ ۲۲:۱۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

معرفی HTML Helpers

یک HTML Helper تنها یک متد است که رشته‌ای را بر می‌گرداند و این رشته می‌تواند حاوی هر نوع محتوای دلخواهی باشد. برای مثال می‌توان از HTML Helpers برای رندر تگ‌های HTML، مانند img و input استفاده کرد. یا به کمک HTML Helpers می‌توان ساختارهای پیچیده‌تری مانند نمایش لیستی از اطلاعات دریافت شده از بانک اطلاعاتی را پیاده سازی کرد. به این ترتیب حجم کدهای تکراری تولید رابط کاربری در Viewهای برنامه‌های ASP.NET MVC به شدت کاهش خواهد یافت، به همراه قابلیت استفاده مجدد از متدهای الحاقی HTML Helpers در برنامه‌های دیگر.
HTML Helpers در ASP.NET MVC معادل کنترل‌های ASP.NET Web forms هستند اما نسبت به آن‌ها بسیار سبک‌تر می‌باشند؛ برای مثال به همراه ViewState و همچنین Event model نیستند.
ASP.NET MVC به همراه تعدادی متد HTML Helper توکار است و برای دسترسی به آن‌ها شیء Html که وهله‌ای از کلاس توکار HtmlHelper می‌باشد، در تمام Viewها قابل استفاده است.

نحوه ایجاد یک HTML Helper سفارشی

از دات نت سه و نیم به بعد امکان توسعه اشیاء توکار فریم ورک، به کمک متدهای الحاقی (extension methods) میسر شده است. برای نوشتن یک HTML Helper نیز باید همین شیوه عمل کرد و کلاس HtmlHelper را توسعه داد. در ادامه قصد داریم یک HTML Helper را جهت رندر تگ label در صفحه ایجاد کنیم. برای این منظور پوشه‌ی جدیدی به نام Helper را به پروژه اضافه نمائید (جهت نظم بیشتر). سپس کلاس زیر را به آن اضافه کنید:

```
using System;
using System.Web.Mvc;

namespace MvcApplication4.Helpers
{
    public static class LabelExtensions
    {
        public static string MyLabel(this HtmlHelper helper, string target, string text)
        {
            return string.Format("<label for='{0}'>{1}</label>", target, text);
        }
    }
}
```

همانطور که ملاحظه می‌کنید متد Label به شکل یک متد الحاقی توسعه دهنده کلاس HtmlHelper که تنها یک رشته را بر می‌گرداند، تعریف شده است. اکنون برای استفاده از این متد در View دلخواهی خواهیم داشت:

```
@using MvcApplication4.Helpers
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

@Html.MyLabel("firstName", "First Name:")
```

ابتدا فضای نام مرتبط با متد الحاقی باید پیوست شود و سپس از طریق شیء Html می‌توان به این متد الحاقی دسترسی پیدا کرد. اگر برنامه را اجرا کنید، این خروجی را مشاهده خواهیم کرد. چرا؟

```
Index
<label for='firstName'>First Name:</label>
```

علت این است که Razor، اطلاعات را Html encoded به مرورگر تحویل می‌دهد. برای تغییر این رویه باید اندکی متد الحاقی تعریف شده را تغییر داد:

```
using System.Web.Mvc;

namespace MvcApplication4.Helpers
{
    public static class LabelExtensions
    {
        public static MvcHtmlString MyLabel(this HtmlHelper helper, string target, string text)
        {
            return MvcHtmlString.Create(string.Format("<label for='{0}'>{1}</label>", target, text));
        }
    }
}
```

تنها تغییر صورت گرفته، استفاده از MvcHtmlString بجای string معمولی است تا Razor آن‌را encode نکند.

تعریف HTML Helpers سفارشی به صورت عمومی:

می‌توان فضای نام MvcApplication4.Helpers این مثال را عمومی کرد. یعنی بجای اینکه بخواهیم در هر View آن‌را ابتدا تعریف کنیم، یکبار آن‌را همانند تعاریف اصلی یک برنامه ASP.NET MVC، عمومی معرفی می‌کنیم. برای این منظور فایل web.config موجود در پوشه Views را باز کنید (و نه فایل web.config قرار گرفته در ریشه اصلی برنامه). سپس فضای نام مورد نظر را در قسمت namespaces صفحات اضافه نمائید:

```
<pages pageBaseType="System.Web.Mvc.WebViewPage">
      <namespaces>
        <add namespace="System.Web.Mvc" />
        <add namespace="System.Web.Mvc.Ajax" />
        <add namespace="System.Web.Mvc.Html" />
        <add namespace="System.Web.Routing" />
        <add namespace="MvcApplication4.Helpers"/>
      </namespaces>
```

به این ترتیب متدهای الحاقی تعریف شده در فضای نام MvcApplication4.Helpers، در تمام Viewهای برنامه در دسترس خواهند بود.

استفاده از کلاس TagBuilder برای تولید HTML Helpers سفارشی:

```
using System.Web.Mvc;

namespace MvcApplication4.Helpers
{
```

```
    public static class LabelExtensions
    {
        public static MvcHtmlString MyNewLabel(this HtmlHelper helper, string target, string text)
        {
            var labelTag = new TagBuilder("label");
            labelTag.MergeAttribute("for", target);
            labelTag.InnerHtml = text;
            return MvcHtmlString.Create(labelTag.ToString());
        }
    }
}
```

در فضای نام System.Web.Mvc، کلاسی وجود دارد به نام TagBuilder که کار تولید تگ‌های HTML، مقدار دهی ویژگی‌ها و خواص آن‌ها را بسیار ساده می‌کند و روش توصیه شده‌ای است برای تولید متدهای HTML Helper. یک نمونه از کاربرد آن‌را برای بازنویسی متد MyLabel ذکر شده در اینجا ملاحظه می‌کنید.
شبیه به همین کلاس، کلاس دیگری به نام HtmlTextWriter در فضای نام System.Web.UI برای انجام اینگونه کارها وجود دارد.

نوشتن HTML Helpers ویژه، به کمک امکانات Razor

نوع دیگری از این متدهای کمکی، Declarative HTML Helpers نام دارند. از این جهت هم Declarative نامیده شده‌اند که مستقیما درون فایل‌های cshtml یا vbhtml به کمک امکانات Razor قابل تعریف هستند. تولید این نوع متدهای کمکی به این شکل نسبت به مثلا روش TagBuilder ساده‌تر است، چون توسط Razor به سادگی و به نحو طبیعی‌تری می‌توان تگ‌های HTML و کدهای مورد نظر را با هم ترکیب کرد (این رفتار طبیعی و روان، یکی از اهداف Razor است).
به عنوان مثال، تعاریف همان کلاس‌های Product و Products قسمت قبل (قسمت هفتم) را در نظر بگیرید. با همان کنترلر و Viewایی که ذکر شد.
سپس برای تعریف این نوع خاص از HTML Helpers/Razor Helpers باید به این نحو عمل کرد:
الف) در ریشه پروژه یا سایت، پوشه‌ی جدیدی به نام App_Code ایجاد کنید (دقیقا به همین نام. این پوشه، جزو پوشه‌های ویژه ASP.NET است).
ب) بر روی این پوشه کلیک راست کرده و گزینه Add|New Item را انتخاب کنید.
ج) در صفحه باز شده، MVC 3 Partial Page/Razor را یافته و مثلا نام ProductsList.cshtml را وارد کرده و این فایل را اضافه کنید.
د) محتوای این فایل جدید را به نحو زیر تغییر دهید:

```
@using MvcApplication4.Models
@helper GetProductsList(List<Product> products)
{
    <ul>
        @foreach (var item in products)
        {
            <li>@item.Name ($@item.Price)</li>
        }
    </ul>
}
```

در اینجا نحوه تعریف یک helper method مخصوص Razor را مشاهده می‌کنید که با کلمه @helper شروع شده است. مابقی آن هم ترکیب آشنای code و markup هستند که به کمک امکانات Razor به این شکل روان میسر شده است.
اکنون اگر Viewایی بخواهد از این اطلاعات استفاده کند تنها کافی است به نحو زیر عمل نماید:

```
@model List<MvcApplication4.Models.Product>
@{
    ViewBag.Title = "Index";
}
```

```
<h2>Index</h2>
@ProductsList.GetProductsList(@Model)
```

ابتدا نام فایل ذکر شده بعد نام متد کمکی تعریف شده در آن. Model هم در اینجا به لیستی از محصولات اشاره می‌کند.
همچنین چون در پوشه app_code قرار گرفته، تمام Viewها به اطلاعات آن دسترسی خواهند داشت. علت هم این است که ASP.NET به صورت خودکار محتوای این پوشه ویژه را همواره کامپایل می‌کند و در اختیار برنامه قرار می‌دهد.
به علاوه در این فایل ProductsList.cshtml، باز هم می‌توان متدهای helper دیگری را اضافه کرد و از این بابت محدودیتی ندارد.
همچنین می‌توان این متد helper را مستقیما داخل یک View هم تعریف کرد. بدیهی است در این حالت قابلیت استفاده مجدد از آن‌را به همراه داشتن Viewهایی تمیز و کم حجم، از دست خواهیم داد.

جهت تکمیل بحث

Turn your Razor helpers into reusable libraries

عنوان: ASP.NET MVC #9
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۵ ۱۳:۳۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

مروری بر HTML Helpers استاندارد مهیا در ASP.NET MVC

یکی از اهداف وجودی Server controls در ASP.NET Web forms، رندر خودکار HTML است. برای مثال Menu control، TreeView control، GridView و امثال آن کار تولید تگ‌های table، tr و بسیاری موارد دیگر را در پشت صحنه برای ما انجام می‌دهند. اما در ASP.NET MVC، هدف رسیدن به یک markup ساده و تمیز است که 100 درصد بر روی اجزای آن کنترل داشته باشیم و این مورد به صورت ضمنی به این معنا است که در اینجا تمام این HTMLها را باید خودمان تولید کنیم. البته در عمل خیر. یک نمونه از آن‌را در قسمت قبل مشاهده کردیم که چطور می‌توان منطق تولید تگ‌های HTML را کپسوله سازی کرد و بارها مورد استفاده قرار داد. به علاوه فریم ورک ASP.NET MVC نیز به همراه تعدادی HTML helper توکار ارائه شده است مانند CheckBox، ActionLink، RenderPartial و غیره که کار تولید تگ‌های HTML ضروری و پایه را برای ما ساده می‌کنند.
یک مثال:

```
@Html.ActionLink("About us", "Index", "About")
```

در اینجا از متدی به نام ActionLink استفاده شده است. شیء Html هم وهله‌ای از کلاس HtmlHelper است که در تمام Viewها قابل دسترسی می‌باشد.
در این متد، اولین پارامتر، متن نمایش داده شده به کاربر را مشخص می‌کند، پارامتر سوم، نام کنترلری است که مورد استفاده قرار می‌گیرد و پارامتر دوم، نام متد یا اکشنی در آن است که فراخوانی خواهد شد (البته هر کدام از این HtmlHelperها به همراه تعداد قابل توجهی overload هم هستند).
زمانیکه این صفحه را رندر کنیم، به خروجی زیر خواهیم رسید:

```
<a href="/About">About us</a>
```

در این لینک نهایی خبری از متد Index ایی که معرفی کردیم، نیست. چرا؟
متد ActionLink بر اساس تعاریف پیش فرض مسیریابی برنامه، سعی می‌کند بهترین خروجی را ارائه دهد. مطابق تعاریف پیش فرض برنامه، متد Index، اکشن پیش فرض کنترلرهای برنامه است. بنابراین ضرورتی به ذکر آن ندیده است.

مثالی دیگر:
همان کلاس‌های Product و Products قسمت هفتم را در نظر بگیرید (قسمت بررسی «ساختار پروژه مثال جاری» در آن مثال).
همچنین به اطلاعات «نوشتن HTML Helpers ویژه، به کمک امکانات Razor» قسمت هشتم هم نیاز داریم.
اینبار می‌خواهیم بجای نمایش لیست ساده‌ای از محصولات، ابتدا نام آن‌ها را به صورت لینک‌هایی در صفحه نمایش دهیم. در ادامه پس از کلیک کاربر روی یک نام، توضیحات بیشتری از محصول انتخابی را در صفحه‌ای دیگر ارائه نمائیم. کدهای View ما اینبار به شکل زیر تغییر می‌کنند:

```
@using MvcApplication5.Models
@model MvcApplication5.Models.Products
@{
    ViewBag.Title = "Index";
}
@helper GetProductsList(List<Product> products)
    {
```

```
    <ul>
        @foreach (var item in products)
        {
            <li>@Html.ActionLink(item.Name, "Details", new { id = item.ProductNumber })</li>
        }
    </ul>
}
<h2>Index</h2>

@GetProductsList(@Model)
```

توضیحات:

ابتدا یک helper method را تعریف کرده‌ایم و به کمک Html.ActionLink، از نام و شماره محصول، جهت تولید لینک‌های نمایش جزئیات هر یک از محصولات کمک گرفته‌ایم. بنابراین در کنترلر خود نیاز به متد جدیدی به نام Details خواهیم داشت که پارامتری از نوع ProductNumber را دریافت می‌کند. سپس جزئیات این محصول را یافته و در View متناظر با خودش ارائه خواهد داد. پارامتر سومی که در متد ActionLink بکارگرفته شده در اینجا مشاهده می‌کنید، یک anonymously typed object است و توسط آن خواصی را تعریف خواهیم کرد که توسط تعاریف مسیریابی تعریف شده در فایل Global.asax.cs، قابل تفسیر و تبدیل به لینک‌های مرتبط و صحیحی باشد.

اکنون اگر این مثال را اجرا کنیم، اولین لینک تولیدی آن به این شکل خواهد بود:

```
http://localhost/Home/Details/D123
```

در اینجا به یک نکته مهم هم باید دقت داشت؛ نام کنترلر به صورت خودکار به این لینک اضافه شده است. بنابراین بهتر است از ایجاد دستی این نوع لینک‌ها خودداری کرده و کار را به متدهای استاندارد فریم ورک واگذار نمود تا بهترین خروجی را دریافت کنیم.

البته اگر الان بر روی این لینک کلیک نمائیم، با پیغام 404 مواجه خواهیم شد. برای تکمیل این مثال، متد Details را به کنترلر تعریف شده اضافه خواهیم کرد:

```
using System.Linq;
using System.Web.Mvc;
using MvcApplication5.Models;

namespace MvcApplication5.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            var products = new Products();
            return View(products);
        }

        public ActionResult Details(string id)
        {
            var product = new Products().FirstOrDefault(x => x.ProductNumber == id);
            if (product == null)
                return View("Error");
            return View(product);
        }
    }
}
```

در متد Details، ابتدا ProductNumber دریافت شده و سپس شیء محصول متناظر با آن، به View این متد، بازگشت داده می‌شود. اگر بر اساس ورودی دریافتی، محصولی یافت نشد، کاربر را به View ایی به نام Error که در پوشه Views/Shared قرار گرفته است، هدایت می‌کنیم.

برای اضافه کردن این View هم بر روی متد کلیک راست کرده و گزینه Add view را انتخاب کنید. چون یک شیء strongly typed از نوع Product را قرار است به View ارسال کنیم (مانند مثال قسمت پنجم)، می‌توان در صفحه باز شده تیک Create a strongly typed view را گذاشت و سپس Model class را از نوع Product انتخاب کرد و در قسمت Scaffold template هم Details را انتخاب نمود. به این ترتیب Code generator توکار VS.NET قسمتی از کار تولید View را برای ما انجام داده و بدیهی است اکنون سفارشی سازی این View تولیدی که قسمت عمده‌ای از آن تولید شده است، کار ساده‌ای می‌باشد:

```
@model MvcApplication5.Models.Product

@{
    ViewBag.Title = "Details";
}

<h2>Details</h2>

<fieldset>
    <legend>Product</legend>

    <div class="display-label">ProductNumber</div>
    <div class="display-field">@Model.ProductNumber</div>

    <div class="display-label">Name</div>
    <div class="display-field">@Model.Name</div>

    <div class="display-label">Price</div>
    <div class="display-field">@String.Format("{0:F}", Model.Price)</div>
</fieldset>
<p>
    @Html.ActionLink("Edit", "Edit", new { /* id=Model.PrimaryKey */ }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

در اینجا کدهای مرتبط با View نمایش جزئیات محصول را مشاهده می‌کنید که توسط VS.NET به صورت خودکار از روی مدل انتخابی تولید شده است.
اکنون یکبار دیگر برنامه را اجرا کرده و بر روی لینک نمایش جزئیات محصولات کلیک نمائید تا بتوان این اطلاعات را در صفحه‌ی بعدی مشاهده نمود.

یک نکته:
اگر سعی کنیم متد helper GetProductsList@ فوق را در پوشه App_Code، همانند قسمت قبل قرار دهیم، به متد Html.ActionLink دسترسی نخواهیم داشت. چرا؟
پیغام خطایی که ارائه می‌شود این است:

```
'System.Web.WebPages.Html.HtmlHelper' does not contain a definition for 'ActionLink'
```

به این معنا که در وهله‌ای از شیء System.Web.WebPages.Html.HtmlHelper، به دنبال متد ActionLink می‌گردد. در حالیکه ActionLink مورد نظر به کلاس System.Web.Mvc.HtmlHelper مرتبط می‌شود.
یک راه حل آن به صورت زیر است. به هر متد helper یک آرگومان WebViewPage page را اضافه می‌کنیم (به همراه دو فضای نامی که به ابتدای فایل اضافه می‌شوند)

```
@using System.Web.Mvc
@using System.Web.Mvc.Html

@using MvcApplication5.Models

@helper GetProductsList(WebViewPage page, List<Product> products)
    {
    <ul>
```

```
        @foreach (var item in products)
        {
            <li> @page.Html.ActionLink(item.Name, "Details", new { id = item.ProductNumber })</li>
        }
    </ul>
}
```

سپس برای استفاده از آن در یک View خواهیم داشت:

```
@MyHelpers.GetProductsList(this, @Model)
```

متد ActionLink و عبارات فارسی

متد ActionLink آدرس‌های وبی را که تولید می‌کند، URL encoded هستند. برای نمونه اگر رشته‌ای که قرار است به عنوان پارامتر به اکشن متد ما ارسال شود، مساوی Hello World است، آن‌را به صورت Hello%20World در صفحه درج می‌کند. البته این مورد مشکلی را در سمت متدهای کنترلرها ایجاد نمی‌کند، چون کار URL decoding خودکار است. اما ... اگر مقداری که قرار است ارسال شود مثلا «مقدار یک» باشد، آدرس تولیدی این شکل را خواهد داشت:

```
http://localhost/Home/Details/%D9%85%D9%82%D8%AF%D8%A7%D8%B1%20%D9%8A%D9%83
```

و اگر این URL encoding انجام نشود، فقط اولین قسمت قبل از فاصله به متد ارسال می‌گردد.
مرورگرهایی مثل فایرفاکس و کروم، مشکلی با نمایش این لینک به شکل اصلی فارسی آن ندارند (حین نمایش، URL decoding را اعمال می‌کنند). اما اگر مرورگر مثلا IE8 باشد، کاربر دقیقا به همین شکل آدرس‌ها را در نوار آدرس مرورگر خود مشاهده خواهد کرد که آنچنان زیبا نیستند.
حل این مشکل، یک نکته کوچک را به همراه دارد. اگر href تولیدی به شکل زیر باشد:

```
<li><a href="/Home/Details/مقدار یک">Super Fast Bike</a></li>
```

IE حین نمایش نهایی آن، آن‌را فارسی نشان خواهد داد. حتی زمانیکه کاربر بر روی آن کلیک کند، به صورت خودکار کاراکترهایی را که لازم است encode نماید، به نحو صحیحی در URL نهایی قابل مشاهده در نوار آدرس‌ها ظاهر خواهد کرد. برای مثال %20 را به صورت خودکار اضافه می‌کند و نگرانی از این لحاظ وجود نخواهد داشت که الان بین دو کلمه فاصله‌ای وجود دارد یا خیر (مرورگرهای دیگر هم دقیقا همین رفتار را در مورد لینک‌های داخل صفحه دارند).
خلاصه این توضیحات متد کمکی زیر است:

```
@helper EmitCleanUnicodeUrl(MvcHtmlString data)
{
    @Html.Raw(HttpUtility.UrlDecode(data.ToString()))
}
```

و برای نمونه نحوه استفاده از آن به شکل زیر خواهد بود:

```
@helper GetProductsList(List<Product> products)
    {
    <ul>
        @foreach (var item in products)
        {
            <li>@EmitCleanUnicodeUrl(@Html.ActionLink(item.Name, "Details", new { id =
item.ProductNumber }))</li>
        }
    </ul>
}
```

ضمن اینکه باید درنظر داشت کلا این نوع طراحی مشکل دارد! برای مثال فرض کنید که در این مثال، جزئیات، نمایش دهنده مطلب ارسالی در یک بلاگ است. یعنی یک سری عنوان و جزئیات متناظر با آن‌ها در دیتابیس وجود دارند. اگر آدرس مطالب به این شکل باشد http://site/blog/details/text، به این معنا است که این text مساوی است با primary key جدول بانک اطلاعاتی. یعنی وبلاگ نویس سایت شما فقط یکبار در طول عمر این برنامه می‌تواند بگوید «سال نو مبارک!». دفعه‌ی بعد به علت تکراری بودن، مجاز به ارسال پیام تبریک دیگری نخواهد بود! به همین جهت بهتر است طراحی را به این شکل تغییر دهید http://site/blog/details/id/text. در اینجا id همان primary key خواهد بود. Text هم عنوان مطلب. Id به جهت خوشایند بانک اطلاعاتی و Text هم برای خوشایند موتورهای جستجو در این URL قرار دارند. مطابق تعاریف مسیریابی برنامه، Text فقط حالت تزئینی داشته و پردازش نخواهد شد.

از این نوع ترفندها زیاد به کار برده می‌شوند. برای نمونه به URL مطالب انجمن‌های معروف اینترنتی دقت کنید. عموما یک عدد را به همراه text مشاهده می‌کنید. عدد در برنامه پردازش می‌شود، متن هم برای موتورهای جستجو درنظر گرفته شده است.

عنوان: **ASP.NET MVC #10**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۷ ۰۰:۳۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

آشنایی با روش‌های مختلف ارسال اطلاعات یک درخواست به کنترلر

تا اینجا با روش‌های مختلف ارسال اطلاعات از یک کنترلر به View متناظر آن آشنا شدیم. اما حالت عکس آن چطور؟ مثلا در ASP.NET Web forms، دوبار بر روی یک دکمه کلیک می‌کردیم و در روال رویدادگردان کلیک آن، همانند برنامه‌های ویندوزی، دسترسی به اطلاعات اشیاء قرار گرفته بر روی فرم را داشتیم. در ASP.NET MVC که کلا مفهوم Events را حذف کرده و وب را همانگونه که هست ارائه می‌دهد و به علاوه کنترلرهای آن، ارجاع مستقیمی را به هیچکدام از اشیاء بصری در خود ندارند (برای مثال کنترلر و متدی در آن نمی‌دانند که الان بر روی View آن، یک گرید قرار دارد یا یک دکمه یا اصلا هیچی)، چگونه می‌توان اطلاعاتی را از کاربر دریافت کرد؟
در اینجا حداقل سه روش برای دریافت اطلاعات از کاربر وجود دارد:
الف) استفاده از اشیاء Context مانند HttpContext، Request، RouteData و غیره
ب) به کمک پارامترهای اکشن متدها
ج) با استفاده از ویژگی جدیدی به نام Data Model Binding

یک مثال کاربردی

قصد داریم یک صفحه لاگین ساده را طراحی کنیم تا بتوانیم هر سه حالت ذکر شده فوق را در عمل بررسی نمائیم. بحث HTML Helpers استاندارد ASP.NET MVC را هم که در قسمت قبل شروع کردیم، لابلای توضیحات قسمت جاری و قسمت‌های بعدی با مثال‌های کاربردی دنبال خواهند شد.
بنابراین یک پروژه جدید خالی ASP.NET MVC را شروع کرده و مدلی را به نام Account با محتوای زیر به پوشه Models برنامه اضافه کنید:

```
namespace MvcApplication6.Models
{
    public class Account
    {
        public string Name { get; set; }
        public string Password { get; set; }
    }
}
```

یک کنترلر جدید را هم به نام LoginController به پوشه کنترلرهای برنامه اضافه کنید. بر روی متد Index پیش فرض آن کلیک راست نمائید و یک View خالی را اضافه نمائید.
در ادامه به فایل Global.asax.cs مراجعه کرده و نام کنترلر پیش‌فرض را به Login تغییر دهید تا به محض شروع برنامه در VS.NET، صفحه لاگین ظاهر شود.
کدهای کامل کنترلر لاگین را در ادامه ملاحظه می‌کنید:

```
using System.Web.Mvc;
using MvcApplication6.Models;

namespace MvcApplication6.Controllers
{
    public class LoginController : Controller
    {
        [HttpGet]
        public ActionResult Index()
```

```
        {
            return View(); //Shows the login page
        }

        [HttpPost]
        public ActionResult LoginResult()
        {
            string name = Request.Form["name"];
            string password = Request.Form["password"];

            if (name == "Vahid" && password == "123")
                ViewBag.Message = "Succeeded";
            else
                ViewBag.Message = "Failed";

            return View("Result");
        }

        [HttpPost]
        [ActionName("LoginResultWithParams")]
        public ActionResult LoginResult(string name, string password)
        {
            if (name == "Vahid" && password == "123")
                ViewBag.Message = "Succeeded";
            else
                ViewBag.Message = "Failed";

            return View("Result");
        }

        [HttpPost]
        public ActionResult Login(Account account)
        {
            if (account.Name == "Vahid" && account.Password == "123")
                ViewBag.Message = "Succeeded";
            else
                ViewBag.Message = "Failed";

            return View("Result");
        }
    }
}
```

همچنین Viewهای متناظر با این کنترلر هم به شرح زیر هستند:

فایل index.cshtml به نحو زیر تعریف خواهد شد:

```
@model MvcApplication6.Models.Account
@{
    ViewBag.Title = "Index";
}
<h2>
    Login</h2>
@using (Html.BeginForm(actionName: "LoginResult", controllerName: "Login"))
{
    <fieldset>
        <legend>Test LoginResult()</legend>
        <p>
            Name: @Html.TextBoxFor(m => m.Name)</p>
        <p>
            Password: @Html.PasswordFor(m => m.Password)</p>
        <input type="submit" value="Login" />
    </fieldset>
}
@using (Html.BeginForm(actionName: "LoginResultWithParams", controllerName: "Login"))
{
    <fieldset>
        <legend>Test LoginResult(string name, string password)</legend>
        <p>
            Name: @Html.TextBoxFor(m => m.Name)</p>
        <p>
            Password: @Html.PasswordFor(m => m.Password)</p>
        <input type="submit" value="Login" />
    </fieldset>
}
```

```
@using (Html.BeginForm(actionName: "Login", controllerName: "Login"))
{
    <fieldset>
        <legend>Test Login(Account acc)</legend>
        <p>
            Name: @Html.TextBoxFor(m => m.Name)</p>
        <p>
            Password: @Html.PasswordFor(m => m.Password)</p>
        <input type="submit" value="Login" />
    </fieldset>
}
```

و فایل result.cshtml هم محتوای زیر را دارد:

```
@{
    ViewBag.Title = "Result";
}
<fieldset>
    <legend>Login Result</legend>
    <p>
        @ViewBag.Message</p>
</fieldset>
```

توضیحاتی در مورد View لاگین برنامه:

در View صفحه لاگین سه فرم را مشاهده می‌کنید. در برنامه‌های ASP.NET Web forms در هر صفحه، تنها یک فرم را می‌توان تعریف کرد؛ اما در ASP.NET MVC این محدودیت برداشته شده است.

تعریف یک فرم هم با متد کمکی Html.BeginForm انجام می‌شود. در اینجا برای مثال می‌شود یک فرم را به کنترلری خاص و متدی مشخص در آن نگاشت نمائیم.

از عبارت using هم برای درج خودکار تگ بسته شدن فرم، در حین dispose شیء MvcForm کمک گرفته شده است.

برای نمونه خروجی HTML اولین فرم تعریف شده به صورت زیر است:

```
<form action="/Login/LoginResult" method="post">
    <fieldset>
        <legend>Test LoginResult()</legend>
        <p>
            Name: <input id="Name" name="Name" type="text" value="" /></p>
        <p>
            Password: <input id="Password" name="Password" type="password" /></p>
        <input type="submit" value="Login" />
    </fieldset>
</form>
```

توسط متدهای کمکی Html.TextBoxFor و Html.PasswordFor یک TextBox و یک PasswordBox به صفحه اضافه می‌شوند، اما این For آن‌ها و همچنین lambda expression ایی که بکارگرفته شده برای چیست؟

متدهای کمکی Html.TextBox و Html.Password از نگارش‌های اولیه ASP.NET MVC وجود داشتند. این متدها نام خاصیت‌ها و پارامترهایی را که قرار است به آن‌ها بایند شوند، به صورت رشته می‌پذیرند. اما با توجه به اینکه در اینجا می‌توان یک strongly typed view را تعریف کرد، تیم ASP.NET MVC بهتر دیده است که این رشته‌ها را حذف کرده و از قابلیتی به نام Static reflection استفاده کند (^ و ^).

با این توضیحات، اطلاعات سه فرم تعریف شده در View لاگین برنامه، به سه متد متفاوت قرار گرفته در کنترلری به نام Login ارسال خواهند شد. همچنین با توجه به مشخص بودن نوع model که در ابتدای فایل تعریف شده، خاصیت‌هایی را که قرار است اطلاعات ارسالی به آن‌ها بایند شوند نیز به نحو strongly typed تعریف شده‌اند و تحت نظر کامپایلر خواهند بود.

توضیحاتی در مورد نحوه عملکرد کنترلر لاگین برنامه:

در این کنترلر صرفنظر از محتوای متدهای آن‌ها، دو نکته جدید را می‌توان مشاهده کرد. استفاده از ویژگی‌های HttpGet، HttpPost و ActionName. در اینجا به کمک ویژگی‌های HttpGet و HttpPost در مورد نحوه دسترسی به این متدها، محدودیت قائل شده‌ایم. به این معنا که تنها در حالت Post است که متد LoginResult در دسترس خواهد بود و اگر شخصی نام این متدها را مستقیما در مرورگر وارد کند (یا همان HttpGet پیش فرض که نیازی هم به ذکر صریح آن نیست)، با پیغام «یافت نشد» مواجه می‌گردد.
البته در نگارش‌های اولیه ASP.NET MVC از ویژگی دیگری به نام AcceptVerbs برای مشخص سازی نوع محدودیت فراخوانی یک اکشن متد استفاده می‌شد که هنوز هم معتبر است. برای مثال:

```
[AcceptVerbs(HttpVerbs.Get)]
```

یک نکته امنیتی:
همیشه متدهای Delete خود را به HttpPost محدود کنید. به این علت که ممکن است در طی مثلا یک ایمیل، آدرسی به شکل http://localhost/blog/delete/10 برای شما ارسال شود و همچنین سشن کار با قسمت مدیریتی بلاگ شما نیز در همان حال فعال باشد. URL ایی به این شکل، در حالت پیش فرض، محدودیت اجرایی HttpGet را دارد. بنابراین احتمال اجرا شدن آن بالا است. اما زمانیکه متد delete را به HttpPost محدود کردید، دیگر این نوع حملات جواب نخواهند داد و حتما نیاز خواهد بود تا اطلاعاتی به سرور Post شود و نه یک Get ساده (مثلا کلیک بر روی یک لینک معمولی)، کار حذف را انجام دهد.

توسط ActionName می‌توان نام دیگری را صرفنظر از نام متد تعریف شده در کنترلر، به آن متد انتساب داد که توسط فریم ورک در حین پردازش نهایی مورد استفاده قرار خواهد گرفت. برای مثال در اینجا به متد LoginResult دوم، نام LoginResultWithParams را انتساب داده‌ایم که در فرم دوم تعریف شده در View لاگین برنامه مورد استفاده قرار گرفته است. وجود این ActionName هم در مثال فوق ضروری است. از آنجائیکه دو متد هم نام را معرفی کرده‌ایم و فریم ورک نمی‌داند که کدامیک را باید پردازش کند. در این حالت (بدون وجود ActionName معرفی شده)، برنامه با خطای زیر مواجه می‌گردد:

```
The current request for action 'LoginResult' on controller type 'LoginController' is ambiguous between
the following action methods:
System.Web.Mvc.ActionResult LoginResult() on type MvcApplication6.Controllers.LoginController
System.Web.Mvc.ActionResult LoginResult(System.String, System.String) on type
MvcApplication6.Controllers.LoginController
```

برای اینکه بتوانید نحوه نگاشت فرم‌ها به متدها را بهتر درک کنید، بر روی چهار return View موجود در کنترلر لاگین برنامه، چهار breakpoint را تعریف کنید. سپس برنامه را در حالت دیباگ اجرا نمائید و تک تک فرم‌ها را یکبار با کلیک بر روی دکمه لاگین، به سرور ارسال نمائید.

بررسی سه روش دریافت اطلاعات از کاربر در ASP.NET MVC

الف) استفاده از اشیاء Context

در ویژوال استودیو، در کنترلر لاگین برنامه، بر روی کلمه Controller کلیک راست کرده و گزینه Go to definition را انتخاب کنید. در اینجا بهتر می‌توان به خواصی که در یک کنترلر به آن‌ها دسترسی داریم، نگاهی انداخت:

```
public HttpContextBase HttpContext { get; }
public HttpRequestBase Request { get; }
```

```
public HttpResponseBase Response { get; }
public RouteData RouteData { get; }
```

در بین این خواص و اشیاء مهیا، Request و RouteData بیشتر مد نظر ما هستند. در مورد RouteData در قسمت ششم این سری، توضیحاتی ارائه شد. اگر مجددا Go to definition مربوط به HttpRequestBase خاصیت Request را بررسی کنیم، موارد ذیل جالب توجه خواهند بود:

```
public virtual NameValueCollection QueryString { get; } // GET variables
public NameValueCollection Form { get; } // POST variables
public HttpCookieCollection Cookies { get; }
public NameValueCollection Headers { get; }
public string HttpMethod { get; }
```

توسط خاصیت Form شیء Request می‌توان به مقادیر ارسالی به سرور در یک کنترلر دسترسی یافت که نمونه‌ای از آن‌را در اولین متد LoginResult می‌توانید مشاهده کنید. این روش در ASP.NET Web forms هم کار می‌کند. جهت اطلاع این روش با ASP کلاسیک دهه نود هم سازگار است!
البته این روش آنچنان مرسوم نیست؛ چون NameValueCollection مورد استفاده، ایندکسی عددی یا رشته‌ای را می‌پذیرد که هر دو با پیشرفت‌هایی که در زبان‌های دات نتی صورت گرفته‌اند، دیگر آنچنان مطلوب و روش مرجح به حساب نمی‌آیند. اما ... هنوز هم قابل استفاده است.
به علاوه اگر دقت کرده باشید در اینجا HttpContextBase داریم بجای HttpContext. تمام این کلاس‌های پایه هم به جهت سهولت انجام آزمون‌های واحد در ASP.NET MVC ایجاد شده‌اند. کار کردن مستقیم با HttpContext مشکل بوده و نیاز به شبیه سازی فرآیندهای رخ داده در یک وب سرور را دارد. اما این کلاس‌های پایه جدید، مشکلات یاد شده را به همراه ندارند.

ب) استفاده از پارامترهای اکشن متدها

نکته‌ای در مورد نامگذاری پارامترهای یک اکشن متد به صورت توکار اعمال می‌شود که باید به آن دقت داشت:
اگر نام یک پارامتر، با نام کلید یکی از رکوردهای موجود در مجموعه‌های زیر یکی باشد، آنگاه به صورت خودکار اطلاعات دریافتی به این پارامتر نگاشت خواهد شد (پارامتر هم نام، به صورت خودکار مقدار دهی می‌شود). این مجموعه‌ها شامل موارد زیرهستند:

```
Request.Form
Request.QueryString
Request.Files
RouteData.Values
```

برای نمونه در متدی که با نام LoginResultWithParams مشخص شده، چون نام‌های دو پارامتر آن، با نام‌های بکارگرفته شده در Html.TextBoxFor و Html.PasswordFor یکی هستند، با مقادیر ارسالی آن‌ها مقدار دهی شده و سپس در متد قابل استفاده خواهند بود. در پشت صحنه هم از همان رکوردهای موجود در Request.Form (یا سایر موارد ذکر شده)، استفاده می‌شود. در اینجا هر رکورد مثلا مجموعه Request.Form، کلیدی مساوی نام ارسالی به سرور را داشته و مقدار آن هم، مقداری است که کاربر وارد کرده است.
اگر همانندی یافت نشد، آن پارامتر با نال مقدار دهی می‌گردد. بنابراین اگر برای مثال یک پارامتر از نوع int را معرفی کرده باشید و چون نوع int، نال نمی‌پذیرد، یک استثناء بروز خواهد کرد. برای حل این مشکل هم می‌توان از Nullable types استفاده نمود (مثلا بجای int id نوشت int? id تا مشکلی جهت انتساب مقدار نال وجود نداشته باشد).
همچنین باید دقت داشت که این بررسی تطابق‌های بین نام عناصر HTML و نام پارامترهای متدها، case insensitive است و به کوچکی و بزرگی حروف حساس نیست. برای مثال، پارامتر معرفی شده در متد LoginResult مساوی string name است، اما نام

خاصیت تعریف شده در کلاس Account مساوی Name بود.

ج) استفاده از ویژگی جدیدی به نام Data Model Binding

در ASP.NET MVC چون می‌توان با یک Strongly typed view کار کرد، خود فریم ورک این قابلیت را دارد که اطلاعات ارسالی یکی فرم را به صورت خودکار به یک وهله از یک شیء نگاشت کند. در اینجا model binder وارد عمل می‌شود، مقادیر ارسالی را استخراج کرده (اطلاعات دریافتی از Form یا کوئری استرینگ‌ها یا اطلاعات مسیریابی و غیره) و به خاصیت‌های یک شیء نگاشت می‌کند. بدیهی است در اینجا این خواص باید عمومی باشند و هم نام عناصر HTML ارسالی به سرور. همچنین model binder پیش فرض ASP.NET MVC را نیز می‌توان کاملا تعویض کرد و محدود به استفاده از model binder توکار آن نیستیم.
وجود این Model binder، کار با ORMها را بسیار لذت بخش می‌کند؛ از آنجائیکه خود فریم ورک ASP.NET MVC می‌تواند عناصر شیءایی را که قرار است به بانک اطلاعاتی اضافه شود، یا در آن به روز شود، به صورت خودکار ایجاد کرده یا به روز رسانی نماید.
نحوه کار با model binder را در متد Login کنترلر فوق می‌توانید مشاهده کنید. بر روی return View آن یک breakpoint قرار دهید. فرم سوم را به سرور ارسال کنید و سپس در VS.NET خواص شیء ساخته شده را در حین دیباگ برنامه، بررسی نمائید.
بنابراین تفاوتی نمی‌کند که از چندین پارامتر استفاده کنید یا اینکه کلا یک شیء را به عنوان پارامتر معرفی نمائید. فریم ورک سعی می‌کند اندکی هوش به خرج داده و مقادیر ارسالی به سرور را به پارامترهای تعریفی، حتی به خواص اشیاء این پارامترهای تعریف شده، نگاشت کند.

در ASP.NET MVC سه نوع Model binder وجود دارند:
1) Model binder پیش فرض که توضیحات آن به همراه مثالی ارائه شد.
2) Form collection model binder که در ادامه توضیحات آن‌را مشاهده خواهید نمود.
3) HTTP posted file base model binder که توضیحات آن به قسمت بعدی موکول می‌شود.

یک نکته:
اولین متد LoginResult کنترلر را به نحو زیر نیز می‌توان بازنویسی کرد:

```
[HttpPost]
[ActionName("LoginResultWithFormCollection")]
public ActionResult LoginResult(FormCollection collection)
{
	string name = collection["name"];
	string password = collection["password"];

	if (name == "Vahid" && password == "123")
		ViewBag.Message = "Succeeded";
	else
		ViewBag.Message = "Failed";

	return View("Result");
}
```

در اینجا FormCollection به صورت خودکار بر اساس مقادیر ارسالی به سرور توسط فریم ورک تشکیل می‌شود (FormCollection هم یک نوع model binder ساده است) و اساسا یک NameValueCollection می‌باشد.
بدیهی است در این حالت باید نگاشت مقادیر دریافتی، به متغیرهای متناظر با آن‌ها، دستی انجام شود (مانند مثال فوق) یا اینکه می‌توان از متد UpdateModel کلاس Controller هم استفاده کرد:

```
[HttpPost]
public ActionResult LoginResultUpdateFormCollection(FormCollection collection)
{
	var account = new Account();
	this.UpdateModel(account, collection.ToValueProvider());

	if (account.Name == "Vahid" && account.Password == "123")
		ViewBag.Message = "Succeeded";
	else
```

```
                ViewBag.Message = "Failed";

        return View("Result");
}
```

متد توکار UpdateModel، به صورت خودکار اطلاعات FormCollection دریافتی را به شیء مورد نظر، نگاشت می‌کند.
همچنین باید عنوان کرد که متد UpdateModel، در پشت صحنه از اطلاعات Model binder پیش فرض و هر نوع Model binder سفارشی که ایجاد کنیم استفاده می‌کند. به این ترتیب زمانیکه از این متد استفاده می‌کنیم، اصلا نیازی به استفاده از FormCollection نیست و متد بدون آرگومان زیر هم به خوبی کار خواهد کرد:

```
[HttpPost]
public ActionResult LoginResultUpdateModel()
{
        var account = new Account();
         this.UpdateModel(account);

         if (account.Name == "Vahid" && account.Password == "123")
                ViewBag.Message = "Succeeded";
         else
                ViewBag.Message = "Failed";

         return View("Result");
}
```

استفاده از model binderها همینجا به پایان نمی‌رسد. نکات تکمیلی آن‌ها در قسمت بعدی بررسی خواهند شد.

عنوان: **سایت‌های مهمی که از ASP.NET MVC استفاده می‌کنند**
نویسنده: وحید نصیری
تاریخ: ۰۹:۴۶:۰۰ ۱۳۹۱/۰۱/۱۸
آدرس: www.dotnettips.info
برچسب‌ها: MVC

عموما استفاده وسیع از نگارش‌های مختلف ASP.NET مربوط به اینترانت‌های شرکت‌های خصوصی و دولتی است. برنامه‌هایی که هیچ وقت رنگ آسمان را هم نخواهند دید و کسی از آمار یا وجود آن‌ها مطلع نخواهد شد. اما در این بین هستند سایت‌های عمومی که از این فناوری‌ها استفاده می‌کنند. مهم‌ترین و پرترافیک‌ترین سایت‌هایی که در حال حاضر از ASP.NET MVC کمک می‌گیرند شامل موارد زیر هستند:

http://www.bing.com

http://www.msnbc.msn.com

http://stackoverflow.com

جالب اینجا است که اخیرا سایت msnbc استفاده وسیعی از RavenDB را هم شروع کرده است .

سایر منابع:

وضعیت استفاده کلی از ASP.NET در سایت‌های عمومی دنیا

Big websites using ASP.NET MVC

What platform and software stack is Bing running on

Showcase of "Live" ASP.NET MVC Sites

Live examples of asp.net mvc driven sites / applications

Using the ASP.NET MVC Framework on live sites

سؤال: چگونه تشخیص دهیم یک سایت از ASP.NET MVC استفاده می‌کند؟

ابتدا افزونه Server Spy را نصب کنید. این افزونه می‌تواند وب سروری را که یک سایت هم اکنون مورد استفاده قرار داده، تشخیص دهد. اگر IIS بود، یعنی این سایت از یکی از مشتقات ASP یا ASP.NET استفاده می‌کند. اگر پسوند صفحات به asp ختم شده بود، ASP کلاسیک دهه نود است. در غیراینصورت یا Web forms است یا MVC. در این حالت به سورس صفحه مراجعه کنید. اگر از ViewState خبری نبود یعنی ASP.NET MVC است.

البته این روش در 90 درصد موارد جواب می‌دهد. می‌شود هدر ارسالی وب سرور را کلا تغییر داد. یعنی ضرورتی ندارد که یک سایت استفاده کننده از IIS حتما اعلام کند که از این وب سرور خاص استفاده می‌کند. یا در ASP.NET Web forms می‌شود ViewState را با ترفندهایی حذف کرد. اما ... این مسایل همه گیر نیست و روش ذکر شده شناسایی، در اکثر موارد جواب می‌دهد.

عنوان: **ASP.NET MVC #11**
نویسنده: وحید نصیری
تاریخ: ۰۹:۵۲:۰۰ ۱۳۹۱/۰۱/۱۹
آدرس: www.dotnettips.info
برچسب‌ها: MVC

بررسی نکات تکمیلی Model binder در ASP.NET MVC

یک برنامه خالی جدید ASP.NET MVC را شروع کنید و سپس مدل زیر را به پوشه Models آن اضافه نمائید:

```
using System;

namespace MvcApplication7.Models
{
    public class User
    {
        public int Id { set; get; }
        public string Name { set; get; }
        public string Password { set; get; }
        public DateTime AddDate { set; get; }
        public bool IsAdmin { set; get; }
    }
}
```

از این مدل چند مقصود ذیل دنبال می‌شوند:
استفاده از Id به عنوان primary key برای edit و update رکوردها. استفاده از DateTime برای اینکه اگر کاربری اطلاعات بی ربطی را وارد کرد چگونه باید این مشکل را در حالت model binding خودکار تشخیص داد و استفاده از IsAdmin برای یادآوری یک نکته امنیتی بسیار مهم که اگر حین model binding خودکار به آن توجه نشود، سایت را با مشکلات حاد امنیتی مواجه خواهد کرد. سیستم پیشرفته است. می‌تواند به صورت خودکار ورودی‌های کاربر را تبدیل به یک شیء حاضر و آماده کند ... اما باید حین استفاده از این قابلیت دلپذیر به یک سری نکات امنیتی هم دقت داشت تا سایت ما به نحو دلپذیری هک نشود!

در ادامه یک کنترلر جدید به نام UserController را به پوشه کنترلرهای پروژه اضافه نمائید. همچنین نام کنترلر پیش فرض تعریف شده در قسمت مسیریابی فایل Global.asax.cs را هم به User تغییر دهید تا در هربار اجرای برنامه در VS.NET، نیازی به تایپ آدرس‌های مرتبط با UserController نداشته باشیم.
یک منبع داده تشکیل شده در حافظه را هم برای نمایش لیستی از کاربران، به نحو زیر به پروژه اضافه خواهیم کرد:

```
using System;
using System.Collections.Generic;

namespace MvcApplication7.Models
{
    public class Users
    {
        public IList<User> CreateInMemoryDataSource()
        {
            return new[]
            {
                new User { Id = 1, Name = "User1", Password = "123", IsAdmin = false, AddDate =
DateTime.Now },
                new User { Id = 2, Name = "User2", Password = "456", IsAdmin = false, AddDate =
DateTime.Now },
                new User { Id = 3, Name = "User3", Password = "789", IsAdmin = true, AddDate =
DateTime.Now }
            };
        }
    }
}
```

در اینجا فعلا هدف آشنایی با زیر ساخت‌های ASP.NET MVC است و درک صحیح نحوه کارکرد آن. مهم نیست از EF استفاده می‌کنید یا NH یا حتی ADO.NET کلاسیک و یا از Micro ORM هایی که پس از ارائه دات نت 4 مرسوم شده‌اند. تهیه یک ToList یا Insert و Update با این فریم ورک‌ها خارج از بحث جاری هستند.

سورس کامل کنترلر User به شرح زیر است:

```
using System;
using System.Linq;
using System.Web.Mvc;
using MvcApplication7.Models;

namespace MvcApplication7.Controllers
{
    public class UserController : Controller
    {
        [HttpGet]
        public ActionResult Index()
        {
            var usersList = new Users().CreateInMemoryDataSource();
            return View(usersList); // Shows the Index view.
        }

        [HttpGet]
        public ActionResult Details(int id)
        {
            var user = new Users().CreateInMemoryDataSource().FirstOrDefault(x => x.Id == id);
            if (user == null)
                return View("Error");
            return View(user); // Shows the Details view.
        }

        [HttpGet]
        public ActionResult Create()
        {
            var user = new User { AddDate = DateTime.Now };
            return View(user); // Shows the Create view.
        }

        [HttpPost]
        public ActionResult Create(User user)
        {
            if (this.ModelState.IsValid)
            {
                // todo: Add record
                return RedirectToAction("Index");
            }
            return View(user); // Shows the Create view again.
        }

        [HttpGet]
        public ActionResult Edit(int id)
        {
            var user = new Users().CreateInMemoryDataSource().FirstOrDefault(x => x.Id == id);
            if (user == null)
                return View("Error");
            return View(user); // Shows the Edit view.
        }

        [HttpPost]
        public ActionResult Edit(User user)
        {
            if (this.ModelState.IsValid)
            {
                // todo: Edit record
                return RedirectToAction("Index");
            }
            return View(user); // Shows the Edit view again.
        }

        [HttpPost]
        public ActionResult Delete(int id)
        {
```

```
            // todo: Delete record
            return RedirectToAction("Index");
        }
    }
}
```

توضیحات:

ایجاد خودکار فرم‌های ورود اطلاعات

در قسمت قبل برای توضیح دادن نحوه ایجاد فرم‌ها در ASP.NET MVC و همچنین نحوه نگاشت اطلاعات آن‌ها به اکشن متدهای کنترلرها، فرم‌های مورد نظر را دستی ایجاد کردیم.
اما باید درنظر داشت که برای ایجاد Viewها می‌توان از ابزار توکار خود VS.NET نیز استفاده کرد و سپس اطلاعات و فرم‌های تولیدی را سفارشی نمود. این سریع‌ترین راه ممکن است زمانیکه مدل مورد استفاده کاملا مشخص است و می‌خواهیم Strongly typed views را ایجاد کنیم.
برای نمونه بر روی متد Index کلیک راست کرده و گزینه Add view را انتخاب کنید. در اینجا گزینه‌ی create a strongly typed view را انتخاب کرده و سپس از لیست مدل‌ها، User را انتخاب نمائید. Scaffold template را هم بر روی حالت List قرار دهید. برای متد Details هم به همین نحو عمل نمائید.
برای ایجاد View متناظر با متد Create در حالت HttpGet، تمام مراحل یکی است. فقط Scaffold template انتخابی را بر روی Create قرار دهید تا فرم ورود اطلاعات، به صورت خودکار تولید شود.
متد Create در حالت HttpPost نیازی به View اضافی ندارد. چون صرفا قرار است اطلاعاتی را از سرور دریافت و ثبت کند.
برای ایجاد View متناظر با متد Edit در حالت HttpGet، باز هم مراحل مانند قبل است با این تفاوت که Scaffold template انتخابی را بر روی گزینه Edit قرار دهید تا فرم ویرایش اطلاعات کاربر به صورت خودکار به پروژه اضافه شود.
متد Edit در حالت HttpPost نیازی به View اضافی ندارد و کارش تنها دریافت اطلاعات از سرور و به روز رسانی بانک اطلاعاتی است.
به همین ترتیب متد Delete نیز، نیازی به View خاصی ندارد. در اینجا بر اساس primary key دریافتی، می‌توان یک کاربر را یافته و حذف کرد.

سفارشی سازی Viewهای خودکار تولیدی

با کمک امکانات Scaffolding نامبرده شده، حجم قابل توجهی کد را در اندک زمانی می‌توان تولید کرد. بدیهی است حتما نیاز به سفارشی سازی کدهای تولیدی وجود خواهد داشت. مثلا شاید نیازی نباشد فیلد پسود کاربر، در حین نمایش لیست کاربران، نمایش داده شود. می‌شود کلا این ستون را حذف کرد و از این نوع مسایل.
یک مورد دیگر را هم در Viewهای تولیدی حتما نیاز است که ویرایش کنیم. آن هم مرتبط است به لینک حذف اطلاعات یک کاربر در صفحه Index.cshtml:

```
@Html.ActionLink("Delete", "Delete", new { id=item.Id }
```

در قسمت قبل هم عنوان شد که اعمال حذف باید بر اساس HttpPost محدود شوند تا بتوان میزان امنیت برنامه را بهبود داد. متد Delete هم در کنترلر فوق تنها به حالت HttpPost محدود شده است. بنابراین ActionLink پیش فرض را حذف کرده و بجای آن فرم و دکمه زیر را قرار می‌دهیم تا اطلاعات به سرور Post شوند:

```
@using (Html.BeginForm(actionName: "Delete", controllerName: "User", routeValues: new { id = item.Id
}))
{
```

```
    <input type="submit" value="Delete"
              onclick="return confirm ('Do you want to delete this record?');" />
}
```

در اینجا نحوه ایجاد یک فرم، که id رکورد متناظر را به سرور ارسال می‌کند، مشاهده می‌کنید.

علت وجود دو متد، به ازای هر Edit یا Create

به ازای هر کدام از متدهای Edit و Create دو متد HttpGet و HttpPost را ایجاد کرده‌ایم. کار متدهای HttpGet نمایش Viewهای متناظر به کاربر هستند. بنابراین وجود آن‌ها ضروری است. در این حالت چون از دو Verb متفاوت استفاده شده، می‌توان متدهای هم نامی را بدون مشکل استفاده کرد. به هر کدام از افعال Get و Post و امثال آن، یک Http Verb گفته می‌شود.

بررسی معتبر بودن اطلاعات دریافتی

کلاس پایه Controller که کنترلرهای برنامه از آن مشتق می‌شوند، شامل یک سری خواص و متدهای توکار نیز هست. برای مثال توسط خاصیت this.ModelState.IsValid می‌توان بررسی کرد که آیا Model دریافتی معتبر است یا خیر. برای بررسی این مورد، یک breakpoint را بر روی سطر this.ModelState.IsValid در متد Create قرار دهید. سپس به صفحه ایجاد کاربر جدید مراجعه کرده و مثلا بجای تاریخ روز، abcd را وارد کنید. سپس فرم را به سرور ارسال نمائید. در این حالت مقدار خاصیت this.ModelState.IsValid مساوی false می‌باشد که حتما باید به آن پیش از ثبت اطلاعات دقت داشت.

شبیه سازی عملکرد ViewState در ASP.NET MVC

در متدهای Create و Edit در حالت Post، اگر اطلاعات Model معتبر نباشند، مجددا شیء User دریافتی، به View بازگشت داده می‌شود. چرا؟
صفحات وب، زمانیکه به سرور ارسال می‌شوند، تمام اطلاعات کنترل‌های خود را از دست خواهد داد (صفحه پاک می‌شود، چون مجددا یک صفحه خالی از سرور دریافت خواهد شد). برای رفع این مشکل در ASP.NET Web forms، از مفهومی به نام ViewState کمک می‌گیرند. کار ViewState ذخیره موقت اطلاعات فرم جاری است برای استفاده مجدد پس از Postback. به این معنا که پس از ارسال فرم به سرور، اگر کاربری در textbox اول مقدار abc را وارد کرده بود، پس از نمایش مجدد فرم، مقدار abc را در همان textbox مشاهده خواهد کرد (شبیه سازی برنامه‌های دسکتاپ در محیط وب). بدیهی است وجود ViewState برای ذخیره سازی این نوع اطلاعات، حجم صفحه را بالا می‌برد (بسته به پیچیدگی صفحه ممکن است به چند صد کیلوبایت هم برسد).
در ASP.NET MVC بجای استفاده از ترفندی به نام ViewState، مجددا اطلاعات همان مدل متناظر با View را بازگشت می‌دهند. در این حالت پس از ارسال صفحه به سرور و نمایش مجدد صفحه ورود اطلاعات، تمام کنترل‌ها با همان مقادیر قبلی وارد شده توسط کاربر قابل مشاهده خواهند بود (مدل مشخص است، View ما هم از نوع strongly typed می‌باشد. در این حالت فریم ورک می‌داند که اطلاعات را چگونه به کنترل‌های قرار گرفته در صفحه نگاشت کند).
در مثال فوق، اگر اطلاعات وارد شده صحیح باشند، کاربر به صفحه Index هدایت خواهد شد. در غیراینصورت مجددا همان View جاری با همان اطلاعات model قبلی که کاربر تکمیل کرده است به او برای تصحیح، نمایش داده می‌شود. این مساله هم جهت بالا بردن سهولت کاربری برنامه بسیار مهم است. تصور کنید که یک فرم خالی با پیغام «تاریخ وارد شده معتبر نیست» مجدا به کاربر نمایش داده شود و از او درخواست کنیم که تمام اطلاعات دیگر را نیز از صفر وارد کند چون اطلاعات صفحه پس از ارسال به سرور پاک شده‌اند؛ که ... اصلا قابل قبول نیست و فوق‌العاده برنامه را غیرحرفه‌ای نمایش می‌دهد.

خطاهای نمایش داده شده به کاربر

به صورت پیش فرض خطایی که به کاربر نمایش داده می‌شود، استثنایی است که توسط فریم ورک صادر شده است. برای مثال نتوانسته است abcd را به یک تاریخ معتبر تبدیل کند. می‌توان توسط this.ModelState.AddModelError خطایی را نیز در اینجا اضافه کرد و پیغام بهتری را به کاربر نمایش داد. یا توسط یک سری data annotations هم کار اعتبار سنجی را سفارشی کرد که

بحث آن به صورت جداگانه در یک قسمت مستقل بررسی خواهد شد.
ولی به صورت خلاصه اگر به فرم‌های تولید شده توسط VS.NET دقت کنید، در ابتدای هر فرم داریم:

```
@Html.ValidationSummary(true)
```

در اینجا خطاهای عمومی در سطح مدل نمایش داده می‌شوند. برای اضافه کردن این نوع خطاها، در متد AddModelError، مقدار key را خالی وارد کنید:

```
ModelState.AddModelError(string.Empty, "There is something wrong with model.");
```

همچنین در این فرم‌ها داریم:

```
@Html.EditorFor(model => model.AddDate)
@Html.ValidationMessageFor(model => model.AddDate)
```

EditorFor سعی می‌کند اندکی هوش به خرج دهد. یعنی اگر خاصیت دریافتی مثلا از نوع bool بود، خودش یک checkbox را در صفحه نمایش می‌دهد. همچنین بر اساس متادیتا یک خاصیت نیز می‌تواند تصمیم گیری را انجام دهد. این متادیتا منظور attributes و data annotations ایی است که به خواص یک مدل اعمال می‌شود. مثلا اگر ویژگی HiddenInput را به یک خاصیت اعمال کنیم، به شکل یک فیلد مخفی در صفحه ظاهر خواهد شد.
یا متد Html.DisplayFor، اطلاعات را به صورت فقط خواندنی نمایش می‌دهد. اصطلاحا به این نوع متدها، Templated Helpers هم گفته می‌شود. بحث بیشتر درباره‌ای این موارد به قسمتی مجزا و مستقل موکول می‌گردد. برای نمونه کل فرم ادیت برنامه را حذف کنید و بجای آن بنویسید Html.EditorForModel و سپس برنامه را اجرا کنید. یک فرم کامل خودکار ویرایش اطلاعات را مشاهده خواهید کرد (و البته نکات سفارشی سازی آن به یک قسمت کامل نیاز دارند).
در اینجا متد ValidationMessageFor کار نمایش خطاهای اعتبارسنجی مرتبط با یک خاصیت مشخص را انجام می‌دهد. بنابراین اگر قصد ارائه خطایی سفارشی و مخصوص یک فیلد مشخص را داشتید، در متد AddModelError، مقدار پارامتر اول یا همان key را مساوی نام خاصیت مورد نظر قرار دهید.

مقابله با مشکل امنیتی Mass Assignment در حین کار با Model binders

استفاده از Model binders بسیار لذت بخش است. یک شیء را به عنوان پارامتر اکشن متد خود معرفی می‌کنیم. فریم ورک هم در ادامه سعی می‌کند تا اطلاعات فرم را به خواص این شیء نگاشت کند. بدیهی است این روش نسبت به روش ASP.NET Web forms که باید به ازای تک تک کنترل‌های موجود در صفحه یکبار کار دریافت اطلاعات و مقدار دهی خواص یک شیء را انجام داد، بسیار ساده‌تر و سریعتر است.
اما اگر همین سیستم پیشرفته جدید ناآگاهانه مورد استفاده قرار گیرد می‌تواند منشاء حملات ناگواری شود که به نام «Mass Assignment» شهرت یافته‌اند.
همان صفحه ویرایش اطلاعات را درنظر بگیرید. چک باکس IsAdmin قرار است در قسمت مدیریتی برنامه تنظیم شود. اگر کاربری نیاز داشته باشد اطلاعات خودش را ویرایش کند، مثلا پسوردش را تغییر دهد، با یک صفحه ساده کلمه عبور قبلی را وارد کنید و دوبار کلمه عبور جدید را نیز وارد نمائید، مواجه خواهد شد. خوب ... اگر همین کاربر صفحه را جعل کند و فیلد چک باکس IsAdmin را به صفحه اضافه کند چه اتفاقی خواهد افتاد؟ بله ... مشکل هم همینجا است. در اینصورت کاربر عادی می‌تواند دسترسی خودش را تا سطح ادمین بالا ببرد، چون model binder اطلاعات IsAdmin را از کاربر دریافت کرده و به صورت خودکار به model ارائه شده، نگاشت کرده است.
برای مقابله با این نوع حملات چندین روش وجود دارند:

الف) ایجاد لیست سفید

به کمک ویژگی Bind می‌توان لیستی از خواص را جهت به روز رسانی به model binder معرفی کرد. مابقی ندید گرفته خواهند شد:

```
public ActionResult Edit([Bind(Include = "Name, Password")] User user)
```

در اینجا تنها خواص Name و Password توسط model binder به خواص شیء User نگاشت می‌شوند.
به علاوه همانطور که در قسمت قبل نیز ذکر شد، متد edit را به شکل زیر نیز می‌توان بازنویسی کرد. در اینجا متدهای توکار UpdateModel و TryUpdateModel نیز لیست سفید خواص مورد نظر را می‌پذیرند (اعمال دستی model binding):

```
[HttpPost]
public ActionResult Edit()
{
    var user = new User();
    if(TryUpdateModel(user, includeProperties: new[] { "Name", "Password" }))
    {
                // todo: Edit record
                return RedirectToAction("Index");
     }
     return View(user); // Shows the Edit view again.
}
```

ب) ایجاد لیست سیاه

به همین ترتیب می‌توان تنها خواصی را معرفی کرد که باید صرفنظر شوند:

```
public ActionResult Edit([Bind(Exclude = "IsAdmin")] User user)
```

در اینجا از خاصیت IsAdmin صرف نظر گردیده و از مقدار ارسالی آن توسط کاربر استفاده نخواهد شد.
و یا می‌توان پارامتر excludeProperties متد TryUpdateModel را نیز مقدار دهی کرد.

لازم به ذکر است که ویژگی Bind را به کل یک کلاس هم می‌توان اعمال کرد. برای مثال:

```
using System;
using System.Web.Mvc;

namespace MvcApplication7.Models
{
    [Bind(Exclude = "IsAdmin")]
    public class User
    {
        public int Id { set; get; }
        public string Name { set; get; }
        public string Password { set; get; }
        public DateTime AddDate { set; get; }
        public bool IsAdmin { set; get; }
    }
}
```

این مورد اثر سراسری داشته و قابل بازنویسی نیست. به عبارتی حتی اگر در متدی خاصیت IsAdmin را مجددا الحاق کنیم، تاثیری

نخواهد داشت.

یا می‌توان از ویژگی ReadOnly هم استفاده کرد:

```
using System;
using System.ComponentModel;

namespace MvcApplication7.Models
{
    public class User
    {
        public int Id { set; get; }
        public string Name { set; get; }
        public string Password { set; get; }
        public DateTime AddDate { set; get; }

        [ReadOnly(true)]
        public bool IsAdmin { set; get; }
    }
}
```

در این حالت هم خاصیت IsAdmin هیچگاه توسط model binder به روز و مقدار دهی نخواهد شد.

ج) استفاده از ViewModels

این راه حلی است که بیشتر مورد توجه معماران نرم افزار است و البته کسانی که پیشتر با الگوی MVVM کار کرده باشند این نام برایشان آشنا است؛ اما در اینجا مفهوم متفاوتی دارد. در الگوی MVVM، کلاس‌های ViewModel شبیه به کنترلرها در MVC هستند یا به عبارتی همانند رهبر یک اکستر عمل می‌کنند. اما در الگوی MVC خیر. در اینجا فقط مدل یک View هستند و نه بیشتر. هدف هم این است که بین Domain Model و View Model تفاوت قائل شد.

کار View model در الگوی MVC، شکل دادن به چندین domain model و همچنین اطلاعات اضافی دیگری که نیاز هستند، جهت استفاده نهایی توسط یک View می‌باشد. به این ترتیب View با یک شیء سر و کار خواهد داشت و همچنین منطق شکل دهی به اطلاعات مورد نیازش هم از داخل View حذف شده و به خواص View model در زمان تشکیل آن منتقل می‌شود.

مشخصات یک View model خوب به شرح زیر است:

الف) رابطه بین یک View و View model آن، رابطه‌ای یک به یک است. به ازای هر View، بهتر است یک کلاس View model وجود داشته باشد.

ب) View ساختار View model را دیکته می‌کند و نه کنترلر.

ج) View modelها صرفا یک سری کلاس POCO (کلاس‌هایی تشکیل شده از خاصیت، خاصیت، خاصیت ....) هستند که هیچ منطقی در آن‌ها قرار نمی‌گیرد.

د) View model باید حاوی تمام اطلاعاتی باشد که View جهت رندر نیاز دارد و نه بیشتر و الزامی هم ندارد که این اطلاعات مستقیما به domain models مرتبط شوند. برای مثال اگر قرار است firstName+LastName در View نمایش داده شود، کار این جمع زدن باید حین تهیه View Model انجام شود و نه داخل View. یا اگر قرار است اطلاعات عددی با سه رقم جدا کننده به کاربر نمایش داده شوند، وظیفه View Model است که یک خاصیت اضافی را برای تهیه این مورد تدارک ببیند. یا مثلا اگر یک فرم ثبت نام داریم و در این فرم لیستی وجود دارد که تنها Id عنصر انتخابی آن در Model اصلی مورد استفاده قرار می‌گیرد، تهیه اطلاعات این لیست هم کار ViewModel است و نه اینکه مدام به Model اصلی بخواهیم خاصیت اضافه کنیم.

ViewModel چگونه پیاده سازی می‌شود؟

اکثر مقالات را که مطالعه کنید، این روش را توصیه می‌کنند:

```
public class MyViewModel
{
    public SomeDomainModel1 Model1 { get; set; }
    public SomeDomainModel2 Model2 { get; set; }
    ...
}
```

یعنی اینکه View ما به اطلاعات مثلا دو Model نیاز دارد. این‌ها را به این شکل محصور و کپسوله می‌کنیم. اگر View، واقعا به تمام فیلدهای این کلاس‌ها نیاز داشته باشد، این روش صحیح است. در غیر اینصورت، این روش نادرست است (و متاسفانه همه جا هم دقیقا به این شکل تبلیغ می‌شود).
ViewModel محصور کننده یک یا چند مدل نیست. در اینجا حس غلط کار کردن با یک ViewModel را داریم. ViewModel فقط باید ارائه کننده اطلاعاتی باشد که یک View نیاز دارد و نه بیشتر و نه تمام خواص تمام کلاس‌های تعریف شده. به عبارتی این نوع تعریف صحیح است:

```
public class MyViewModel
{
    public string SomeExtraField1 { get; set; }
    public string SomeExtraField2 { get; set; }
    public IEnumerable<SelectListItem> StateSelectList { get; set; }
    // ...
    public string PersonFullName { set; set; }
}
```

در اینجا، View متناظری، قرار است نام کامل یک شخص را به علاوه یک سری اطلاعات اضافی که در domain model نیست، نمایش دهد. مثلا نمایش نام استان‌ها که نهایتا Id انتخابی آن قرار است در برنامه استفاده شود.
خلاصه علت وجودی ViewModel این موارد است:
الف) Model برنامه را مستقیما در معرض استفاده قرار ندهیم (عدم رعایت این نکته به مشکلات امنیتی حادی هم حین به روز رسانی اطلاعات ممکن است ختم شود که پیشتر توضیح داده شد).
ب) فیلدهای نمایشی اضافی مورد نیاز یک View را داخل Model برنامه تعریف نکنیم (مثلا تعاریف عناصر یک دراپ داون لیست، جایش اینجا نیست. مدل فقط نیاز به Id عنصر انتخابی آن دارد).

با این توضیحات، اگر View به روز رسانی اطلاعات کلمه عبور کاربر، تنها به اطلاعات id آن کاربر و کلمه عبور او نیاز دارد، فقط باید همین اطلاعات را در اختیار View قرار داد و نه بیشتر:

```
namespace MvcApplication7.Models
{
    public class UserViewModel
    {
        public int Id { set; get; }
        public string Password { set; get; }
    }
}
```

به این ترتیب دیگر خاصیت IsAdming اضافه‌ای وجود ندارد که بخواهد مورد حمله واقع شود.

استفاده از model binding برای آپلود فایل به سرور

برای آپلود فایل به سرور تنها کافی است یک اکشن متد به شکل زیر را تعریف کنیم. HttpPostedFileBase نیز یکی دیگر از model binderهای توکار ASP.NET MVC است:

```
[HttpGet]
public ActionResult Upload()
{
       return View(); // Shows the upload page
}
```

```
[HttpPost]
public ActionResult Upload(System.Web.HttpPostedFileBase file)
{
       string filename = Server.MapPath("~/files/somename.ext");
       file.SaveAs(filename);
       return RedirectToAction("Index");
}
```

View متناظر هم می‌تواند به شکل زیر باشد:

```
@{
    ViewBag.Title = "Upload";
}
<h2>
    Upload</h2>
@using (Html.BeginForm(actionName: "Upload", controllerName: "User",
                       method: FormMethod.Post,
                       htmlAttributes: new { enctype = "multipart/form-data" }))
{
    <text>Upload a photo:</text> <input type="file" name="photo" />
    <input type="submit" value="Upload" />
}
```

اگر دقت کرده باشید در طراحی ASP.NET MVC از anonymously typed objects زیاد استفاده می‌شود. در اینجا هم برای معرفی enctype فرم آپلود، مورد استفاده قرار گرفته است. به عبارتی هر جایی که مشخص نبوده چه تعداد ویژگی یا کلا چه ویژگی‌ها و خاصیت‌هایی را می‌توان تنظیم کرد، اجازه تعریف آن‌ها را به صورت anonymously typed objects میسر کرده‌اند. یک نمونه دیگر آن در متد routes.MapRoute فایل Global.asax.cs است که پارامتر سوم دریافت مقدار پیش فرض‌ها نیز anonymously typed object است. یا نمونه دیگر آن‌را در همین قسمت در جایی که لینک delete را به فرم تبدیل کردیم مشاهده نمودید. مقدار routeValues هم یک anonymously typed object معرفی شد.

سفارشی سازی model binder پیش فرض ASP.NET MVC

در همین مثال فرض کنید تاریخ را به صورت شمسی از کاربر دریافت می‌کنیم. خاصیت تعریف شده هم DateTime میلادی است. به عبارتی model binder حین تبدیل رشته تاریخ شمسی دریافتی به تاریخ میلادی با شکست مواجه شده و نهایتا خاصیت this.ModelState.IsValid مقدارش false خواهد بود. برای حل این مشکل چکار باید کرد؟
برای این منظور باید نحوه پردازش یک نوع خاص را سفارشی کرد. ابتدا با پیاده سازی اینترفیس IModelBinder شروع می‌کنیم. توسط bindingContext.ValueProvider می‌توان به مقداری که کاربر وارد کرده در میانه راه دسترسی یافت. آن‌را تبدیل کرده و نمونه صحیح را بازگشت داد.
نمونه‌ای از این پیاده سازی را در ادامه ملاحظه می‌کنید:

```
using System;
using System.Globalization;
using System.Web.Mvc;

namespace MvcApplication7.Binders
{
    public class PersianDateModelBinder : IModelBinder
    {

        public object BindModel(ControllerContext controllerContext, ModelBindingContext
bindingContext)
        {
            var valueResult = bindingContext.ValueProvider.GetValue(bindingContext.ModelName);
            var modelState = new ModelState { Value = valueResult };
            object actualValue = null;
            try
            {
```

```
                var parts = valueResult.AttemptedValue.Split('/'); //ex. 1391/1/19
                if (parts.Length != 3) return null;
                int year = int.Parse(parts[0]);
                int month = int.Parse(parts[1]);
                int day = int.Parse(parts[2]);
                actualValue = new DateTime(year, month, day, new PersianCalendar());
            }
            catch (FormatException e)
            {
                modelState.Errors.Add(e);
            }

            bindingContext.ModelState.Add(bindingContext.ModelName, modelState);
            return actualValue;
        }
    }
}
```

سپس برای معرفی PersianDateModelBinder جدید تنها کافی است سطر زیر را

```
ModelBinders.Binders.Add(typeof(DateTime), new PersianDateModelBinder());
```

به متد Application_Start قرار گرفته در فایل Global.asax.cs برنامه اضافه کرد. از این پس کاربران می‌توانند تاریخ‌ها را در برنامه شمسی وارد کنند و model binder بدون مشکل خواهد توانست اطلاعات ورودی را به معادل DateTime میلادی آن تبدیل کند و استفاده نماید.
تعریف مدل بایندر سفارشی در فایل Global.asax.cs آن‌را به صورت سراسری در تمام مدل‌ها و اکشن‌متدها فعال خواهد کرد. اگر نیاز بود تنها یک اکشن متد خاص از این مدل بایندر سفارشی استفاده کند می‌توان به روش زیر عمل کرد:

```
public ActionResult Create([ModelBinder(typeof(PersianDateModelBinder))] User user)
```

همچنین ویژگی ModelBinder را به یک کلاس هم می‌توان اعمال کرد:

```
[ModelBinder(typeof(PersianDateModelBinder))]
public class User
{
```

عنوان: **ASP.NET MVC #12**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۱ ۰۰:۲۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

تولید خودکار فرم‌های ورود و نمایش اطلاعات در ASP.NET MVC بر اساس اطلاعات مدل‌ها

در الگوی MVC، قسمت M یا مدل آن یک سری ویژگی‌های خاص خودش را دارد:
شما را وادار نمی‌کند که مدل را به نحو خاصی طراحی کنید. شما را مجبور نمی‌کند که کلاس‌های مدل را برای نمونه همانند کلاس‌های کنترلرها، از کلاس خاصی به ارث ببرید. یا حتی در مورد نحوه‌ی دسترسی به داده‌ها نیز، نظری ندارد. به عبارتی برنامه نویس است که می‌تواند بر اساس امکانات مهیای در کل اکوسیستم دات نت، در این مورد آزادانه تصمیم گیری کند.
بر همین اساس ASP.NET MVC یک سری قرارداد را برای سهولت اعتبار سنجی یا تولید بهتر رابط کاربری بر اساس اطلاعات مدل‌ها، فراهم آورده است. این قراردادها هم چیزی نیستند جز یک سری metadata که نحوه‌ی دربرگیری اطلاعات را در مدل‌ها توضیح می‌دهند. برای دسترسی به آن‌ها پروژه جاری باید ارجاعی را به اسمبلی‌های System.ComponentModel.DataAnnotations.dll و System.Web.Mvc.dll داشته باشد (که VS.NET به صورت خودکار در ابتدای ایجاد پروژه اینکار را انجام می‌دهد).

یک مثال کاربردی

یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. در پوشه مدل‌های آن، مدل اولیه‌ای را با محتوای زیر ایجاد نمائید:

```
using System;

namespace MvcApplication8.Models
{
    public class Employee
    {
        public int Id { set; get; }
        public string Name { set; get; }
        public decimal Salary { set; get; }
        public string Address { set; get; }
        public bool IsMale { set; get; }
        public DateTime AddDate { set; get; }
    }
}
```

سپس یک کنترلر جدید را هم به نام EmployeeController با محتوای زیر به پروژه اضافه نمائید:

```
using System;
using System.Web.Mvc;
using MvcApplication8.Models;

namespace MvcApplication8.Controllers
{
    public class EmployeeController : Controller
    {
        public ActionResult Create()
        {
            var employee = new Employee { AddDate = DateTime.Now };
            return View(employee);
        }
    }
}
```

بر روی متد Create کلیک راست کرده و یک View ساده را برای آن ایجاد نمائید. سپس محتوای این View را به صورت زیر تغییر دهید:

```
@model MvcApplication8.Models.Employee
@{
    ViewBag.Title = "Create";
}

<h2>Create An Employee</h2>

@using (Html.BeginForm(actionName: "Create", controllerName: "Employee"))
{
    @Html.EditorForModel()
    <input type="submit" value="Save" />
}
```

اکنون اگر پروژه را اجرا کرده و مسیر http://localhost/employee/create را وارد نمائید، یک صفحه ورود اطلاعات تولید شده به صورت خودکار را مشاهده خواهید کرد. متد Html.EditorForModel بر اساس اطلاعات خواص عمومی مدل، یک فرم خودکار را تشکیل می‌دهد.
البته فرم تولیدی به این شکل شاید آنچنان مطلوب نباشد، از این جهت که برای مثال Id را هم لحاظ کرده، در صورتیکه قرار است این Id توسط بانک اطلاعاتی انتساب داده شود و نیازی نیست تا کاربر آن‌را وارد نماید. یا مثلا برچسب AddDate نباید به این شکل صرفا بر اساس نام خاصیت متناظر با آن تولید شود و مواردی از این دست. به عبارتی نیاز به سفارشی سازی کار این فرم ساز توکار ASP.NET MVC وجود دارد که ادامه بحث جاری را تشکیل خواهد داد.

سفارشی سازی فرم ساز توکار ASP.NET MVC با کمک Metadata خواص

برای اینکه بتوان نحوه نمایش فرم خودکار تولید شده را سفارشی کرد، می‌توان از یک سری attribute و data annotations توکار دات نت و ASP.NET MVC استفاده کرد و نهایتا این metadata توسط فریم ورک، مورد استفاده قرار خواهند گرفت. برای مثال:

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Web.Mvc;

namespace MvcApplication8.Models
{
    public class Employee
    {
        //[ScaffoldColumn(false)]

        [HiddenInput(DisplayValue=false)]
        public int Id { set; get; }

        public string Name { set; get; }

        [DisplayName("Annual Salary ($)")]
        public decimal Salary { set; get; }

        public string Address { set; get; }

        [DisplayName("Is Male?")]
        public bool IsMale { set; get; }

        [DisplayName("Start Date")]
        [DataType(DataType.Date)]
        public DateTime AddDate { set; get; }
    }
}
```

در اینجا به کمک ویژگی HiddenInput از نمایش عمومی خاصیت Id جلوگیری خواهیم کرد یا توسط ویژگی DisplayName، برچسب دلخواه خود را به عناصر فرم تشکیل شده، انتساب خواهیم داد. اگر نیاز باشد تا خاصیتی کلا از رابط کاربری حذف شود می‌توان از ویژگی ScaffoldColumn با مقدار false استفاده کرد. یا توسط DataType، مشخص کرده‌ایم که نوع ورودی فقط قرار است Date باشد و نیازی به قسمت Time آن نداریم.
DataType شامل نوع‌های از پیش تعریف شده دیگری نیز هست. برای مثال اگر نیاز به نمایش TextArea بود از مقدار MultilineText، استفاده کنید:

```
[DataType(DataType.MultilineText)]
```

یا برای نمایش PasswordBox از مقدار Password می‌توان کمک گرفت. اگر نیاز دارید تا آدرس ایمیلی به شکل یک لینک mailto نمایش داده شود از مقدار EmailAddress استفاده کنید. به کمک مقدار Url، متن خروجی به صورت خودکار تبدیل به یک آدرس قابل کلیک خواهد شد.
اکنون اگر پروژه را مجددا کامپایل کنیم و به آدرس ایجاد یک کارمند جدید مراجعه نمائیم، با رابط کاربری بهتری مواجه خواهیم شد.

سفارشی سازی ظاهر فرم ساز توکار ASP.NET MVC

در ادامه اگر بخواهیم ظاهر این فرم را اندکی سفارشی‌تر کنیم، بهتر است به سورس صفحه تولیدی در مرورگر مراجعه کنیم. در اینجا یک سری عناصر HTML محصور شده با div را خواهیم یافت. هر کدام از این‌ها هم با classهای css خاص خود تعریف شده‌اند. بنابراین اگر علاقمند باشیم که رنگ و قلم و غیره این موارد تغییر دهیم، تنها کافی است فایل css برنامه را ویرایش کنیم و نیازی به دستکاری مستقیم کدهای برنامه نیست.

انتساب قالب‌های سفارشی به خواص یک شیء

تا اینجا در مورد نحوه سفارشی سازی رنگ، قلم، برچسب و نوع داده‌های هر کدام از عناصر نهایی نمایش داده شده، توضیحاتی را ملاحظه نمودید.
در فرم تولیدی نهایی، خاصیت bool تعریف شده به صورت خودکار به یک checkbox تبدیل شده است. چقدر خوب می‌شد اگر امکان تبدیل آن مثلا به RadioButton انتخاب مرد یا زن بودن کارمند ثبت شده در سیستم وجود داشت. برای اصلاح یا تغییر این مورد، باز هم می‌توان از متادیتای خواص، جهت تعریف قالبی خاص برای هر کدام از خواص مدل استفاده کرد.
به پوشه Views/Shared مراجعه کرده و یک پوشه جدید به نام EditorTemplates را ایجاد نمائید. بر روی این پوشه کلیک راست کرده و گزینه Add view را انتخاب کنید. در صفحه باز شده، گزینه «Create as a partial view» را انتخاب نمائید و نام آن‌را هم مثلا GenderOptions وارد کنید. همچنین گزینه «Create a strongly typed view» را نیز انتخاب کنید. مقدار Model class را مساوی bool وارد نمائید. فعلا یک hello داخل این صفحه جدید وارد کرده و سپس خاصیت IsMale را به نحو زیر تغییر دهید:

```
[DisplayName("Gender")]
[UIHint("GenderOptions")]
public bool IsMale { set; get; }
```

توسط ویژگی UIHint، می‌توان یک خاصیت را به یک partial view متصل کرد. در اینجا خاصیت IsMale به partial view ایی به نام GenderOptions متصل شده است. اکنون اگر برنامه را کامپایل و اجرا کرده و آدرس ایجاد یک کارمند جدید را ملاحظه کنید، بجای Checkbox باید یک hello نمایش داده شود.
محتویات این Partial view هم نهایتا به شکل زیر خواهند بود:

```
@model bool
<p>@Html.RadioButton("", false, !Model) Female</p>
<p>@Html.RadioButton("", true, Model) Male</p>
```

در اینجا Model که از نوع bool تعریف شده، به خاصیت IsMale اشاره خواهد کرد. دو RadioButton هم برای انتخاب بین حالت زن و مرد تعریف شده‌اند.

یا یک مثال جالب دیگر در این زمینه می‌تواند تبدیل enum به یک Dropdownlist باشد. در این حالت partial view ما شکل زیر را خواهد یافت:

```
@model Enum
@Html.DropDownListFor(m => m, Enum.GetValues(Model.GetType())
         .Cast<Enum>()
         .Select(m => {
                   string enumVal = Enum.GetName(Model.GetType(), m);
                   return new SelectListItem() {
                             Selected = (Model.ToString() == enumVal),
                             Text = enumVal,
                             Value = enumVal
                  };
}))
```

و برای استفاده از آن، از ویژگی زیر می‌توان کمک گرفت (مزین کردن خاصیتی از نوع یک enum دلخواه، جهت تبدیل خودکار آن به یک دراپ داون لیست):

```
[UIHint("Enum")]
```

سایر متدهای کمکی تولید و نمایش خودکار اطلاعات از روی اطلاعات مدل‌های برنامه

متدهای دیگری نیز در رده‌ی Templated helpers قرار می‌گیرند. اگر از متد Html.EditorFor استفاده کنیم، از تمام این اطلاعات متادیتای تعریف شده نیز استفاده خواهد کرد. همانطور که در قسمت قبل (قسمت 11) نیز توضیح داده شد، صفحه استاندارد Add view در VS.NET به همراه یک سری قالب تولید فرم‌های Create و Edit هم هست که دقیقا کد نهایی تولیدی را بر اساس همین متد تولید می‌کند.
استفاده از Html.EditorFor انعطاف پذیری بیشتری را به همراه دارد. برای مثال اگر یک طراح وب، طرح ویژه‌ای را در مورد ظاهر فرم‌های سایت به شما ارائه دهد، بهتر است از این روش استفاده کنید. اما خروجی نهایی Html.EditorForModel به کمک تعدادی متادیتا و اندکی دستکاری CSS، از دیدگاه یک برنامه نویس بی نقص است!
به علاوه، متد Html.DisplayForModel نیز مهیا است. بجای اینکه کار تولید رابط کاربری اطلاعات نمایش جزئیات یک شیء را انجام دهید، اجازه دهید تا متد Html.DisplayForModel اینکار را انجام دهد. سفارشی سازی آن نیز همانند قبل است و بر اساس متادیتای خواص انجام می‌شود. در این حالت، مسیر پیش فرض جستجوی قالب‌های UIHint آن، Views/Shared/DisplayTemplates می‌باشد. همچنین Html.DisplayFor نیز جهت کار با یک خاصیت مدل تدارک دیده شده است. البته باید درنظر داشت که استفاده از پوشه Views/Shared اجباری نیست. برای مثال اگر از پوشه Views/Home/DisplayTemplates استفاده کنیم، قالب‌های سفارشی تهیه شده تنها جهت Viewهای کنترلر home قابل استفاده خواهند بود.
یکی دیگر از ویژگی‌هایی که جهت سفارشی سازی نحوه نمایش خودکار اطلاعات می‌تواند مورد استفاده قرار گیرد، DisplayFormat است. برای مثال اگر مقدار خاصیت در حال نمایش نال بود، می‌توان مقدار دیگری را نمایش داد:

```
[DisplayFormat(NullDisplayText = "-")]
```

یا اگر علاقمند بودیم که فرمت اطلاعات در حال نمایش را تغییر دهیم، به نحو زیر می‌توان عمل کرد:

```
[DisplayFormat(DataFormatString  = "{0:n}")]
```

مقدار DataFormatString در پشت صحنه در متد string.Format مورد استفاده قرار می‌گیرد.
و اگر بخواهیم که این ویژگی در حالت تولید فرم ویرایش نیز درنظر گرفته شود، می‌توان خاصیت ApplyFormatInEditMode را نیز مقدار دهی کرد:

```
[DisplayFormat(DataFormatString  = "{0:n}", ApplyFormatInEditMode = true)]
```

بازنویسی قالب‌های پیش فرض تولید فرم یا نمایش اطلاعات خودکار ASP.NET MVC

یکی دیگر از قرارداهای بکارگرفته شده در حین استفاده از قالب‌های سفارشی، استفاده از نام اشیاء می‌باشد. مثلا در پوشه Views/Shared/DisplayTemplates، اگر یک Partial view به نام String.cshtml وجود داشته باشد، از این پس نحوه رندر کلیه خواص رشته‌ای تمام مدل‌ها، بر اساس محتوای فایل String.cshtml مشخص می‌شود؛ به همین ترتیب در مورد datetime و سایر انواع مهیا.
برای مثال اگر خواستید تمام تاریخ‌های میلادی دریافتی از بانک اطلاعاتی را شمسی نمایش دهید، فقط کافی است یک فایل datetime.cshtml سفارشی را تولید کنید که Model آن تاریخ میلادی دریافتی است و نهایتا کار این Partial view، رندر تاریخ تبدیل شده به همراه تگ‌های سفارشی مورد نظر می‌باشد. در این حالت نیازی به ذکر ویژگی UIHint نیز نخواهد بود و همه چیز خودکار است.
به همین ترتیب اگر نام مدل ما Employee باشد و فایل Partial view ایی به نام Employee.cshtml در پوشه Views/Shared/DisplayTemplates قرار گیرد، متد Html.DisplayForModel به صورت پیش فرض از محتوای این فایل جهت رندر اطلاعات نمایش جزئیات شیء Employee استفاده خواهد کرد.
داخل Partial viewهای سفارشی تعریف شده به کمک خاصیت ViewData.TemplateInfo.FormattedModelValue مقدار نهایی فرمت شده قابل استفاده را فراهم می‌کند. این مورد هم از این جهت حائز اهمیت است که نیازی نباشد تا ویژگی DisplayFormat را به صورت دستی پردازش کنیم. همچنین اطلاعات ViewData.ModelMetadata نیز دراینجا قابل دسترسی هستند.

سؤال: Partial View چیست؟

همانطور که از نام Partial view برمی‌آید، هدف آن رندر کردن قسمتی از صفحه است به همراه استفاده مجدد از کدهای تولید رابط کاربری در چندین و چند View؛ چیزی شبیه به User controls در ASP.NET Web forms البته با این تفاوت که Page life cycle و Code behind و سایر موارد مشابه آن در اینجا حذف شده‌اند. همچنین از Partial viewها برای به روز رسانی قسمتی از صفحه حین فراخوانی‌های Ajaxایی نیز استفاده می‌شود. مهم‌ترین کاربرد Partial views علاوه بر استفاده مجدد از کدها، خلوت کردن Viewهای شلوغ است جهت ساده‌تر سازی نگهداری آن‌ها در طول زمان (یک نوع Refactoring فایل‌های View محسوب می‌شوند).
پسوند این فایل‌ها نیز بسته به موتور View مورد استفاده تعیین می‌شود. برای مثال حین استفاده از Razor، پسوند Partial views همان cshtml یا vbhtml می‌باشد. یا اگر از web forms view engine استفاده شود، پسوند آن‌ها ascx است (همانند User

controls در وب فرم‌ها).
البته چون در حالت استفاده از موتور Razor، پسوند View و Partial viewها یکی است، مرسوم شده است که نام Partial viewها را با یک underline شروع کنیم تا بتوان بین این دو تمایز قائل شد.
اگر این فایل‌ها را در پوشه Views/Shared تعریف کنیم، در تمام Viewها قابل استفاده خواهند بود. اما اگر مثلا در پوشه Views/Home آن‌هارا قرار دهیم، تنها در Viewهای متعلق به کنترلر Home، قابل بکارگیری می‌باشند.
Partial views را نیز می‌توان strongly typed تعریف کرد و به این ترتیب با مشخص سازی دقیق نوع model آن، علاوه بر بهره‌مندی از Intellisense خودکار، رندر آن‌را نیز تحت کنترل کامپایلر قرار داد.
مقدار Model در یک View بر اساس اطلاعات مدلی که به آن ارسال شده است تعیین می‌گردد. اما در یک Partial view که جزئی از یک View را نهایتا تشکیل خواهد داد، بر اساس مقدار ارسالی از طریق View معین می‌گردد.

یک مثال

در ادامه قصد داریم کد حلقه نمایش لیستی از عناصر تولید شده توسط VS.NET را به یک Partial view منتقل و Refactor کنیم.
ابتدا یک منبع داده فرضی زیر را در نظر بگیرید:

```
using System;
using System.Collections.Generic;

namespace MvcApplication8.Models
{
    public class Employees
    {
        public IList<Employee> CreateEmployees()
        {
            return new[]
            {
                new Employee { Id = 1, AddDate = DateTime.Now.AddYears(-3), Name = "Emp-01", Salary =
3000},
                new Employee { Id = 2, AddDate = DateTime.Now.AddYears(-2), Name = "Emp-02", Salary =
2000},
                new Employee { Id = 3, AddDate = DateTime.Now.AddYears(-1), Name = "Emp-03", Salary =
1000}
            };
        }
    }
}
```

سپس از آن در یک کنترلر برای بازگشت لیستی از کارکنان استفاده خواهیم کرد:

```
public ActionResult EmployeeList()
{
    var list = new Employees().CreateEmployees();
    return View(list);
}
```

View متناظر با این متد را هم با کلیک راست بر روی متد، انتخاب گزینه Add view و سپس ایجاد یک strongly typed view از نوع کلاس Employee، ایجاد خواهیم کرد.
در ادامه قصد داریم بدنه حلقه زیر را refactor کنیم و آن‌را به یک Parial view منتقل نمائیم تا View ما اندکی خلوت‌تر و مفهوم‌تر شود:

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Salary)
```

```
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Address)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.IsMale)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.AddDate)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}
```

سپس بر روی پوشه Views/Employee کلیک راست کرده و گزینه Add|View را انتخاب کنید. در اینجا نام _EmployeeItem را وارد کرده و همچنین گزینه Create as a partial view و create a strongly typed view را نیز انتخاب کنید. نوع مدل هم Employee خواهد بود. به این ترتیب فایل زیر تشکیل خواهد شد:

```
\Views\Employee\_EmployeeItem.cshtml
```

ابتدای نام فایل‌را با underline شروع کرده‌ایم تا بتوان بین Viewها و Partial views تفاوت قائل شد. همچنین این Partial view چون داخل پوشه Employee تعریف شده، فقط در Viewهای کنترلر Employee در دسترس خواهد بود.
در ادامه کل بدنه حلقه فوق را cut کرده و در این فایل جدید paste نمائید. مرحله اول refactoring یک view به همین نحو آغاز می‌شود. البته در این حالت قادر به استفاده از Partial view نخواهیم بود چون اطلاعاتی که به این فایل ارسال می‌گردد و مدلی که در دسترس آن است از نوع Employee است و نه لیستی از کارمندان. به همین جهت باید item را با Model جایگزین کرد:

```
@model MvcApplication8.Models.Employee

    <tr>
        <td>
            @Html.DisplayFor(x => x.Name)
        </td>
        <td>
            @Html.DisplayFor(x => x.Salary)
        </td>
        <td>
            @Html.DisplayFor(x => x.Address)
        </td>
        <td>
            @Html.DisplayFor(x => x.IsMale)
        </td>
        <td>
            @Html.DisplayFor(x => x.AddDate)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id = Model.Id }) |
            @Html.ActionLink("Details", "Details", new { id = Model.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id = Model.Id })
        </td>
    </tr>
```

سپس برای استفاده از این Partial view در صفحه نمایش لیست کارمندان خواهیم داشت:

```
@foreach (var item in Model) {
    @Html.Partial("_EmployeeItem", item)
```

```
}
```

متد Html.Partial، اطلاعات یک Partial view را پردازش و تبدیل به یک رشته کرده و در اختیار Razor قرار می‌دهد تا در صفحه نمایش داده شود. پارامتر اول آن نام Partial view مورد نظر است (نیازی به ذکر پسوند فایل نیست) و پارامتر دوم، اطلاعاتی است که به آن ارسال خواهد شد.
متد دیگری هم وجود دارد به نام Html.RenderPartial. کار این متد نوشتن مستقیم در Response است، برخلاف Html.Partial که فقط یک رشته را بر می‌گرداند.

نمایش اطلاعات از کنترلرهای مختلف در یک صفحه

Html.Partial بر اساس اطلاعات مدل ارسالی از یک کنترلر، کار رندر قسمتی از آن‌را در یک View خاص عهده دار خواهد شد. اما اگر بخواهیم مثلا در یک صفحه یک قسمت را به نمایش آخرین اخبار و یک قسمت را به نمایش آخرین وضعیت آب و هوا اختصاص دهیم، از روش دیگری به نام RenderAction می‌توان کمک گرفت. در اینجا هم دو متد Html.Action و Html.RenderAction وجود دارند. اولی یک رشته را بر می‌گرداند و دومی اطلاعات را مستقیما در Response درج می‌کند.

یک مثال:
کنترلر جدیدی را به نام MenuController به پروژه اضافه کنید:

```
using System.Web.Mvc;

namespace MvcApplication8.Controllers
{
    public class MenuController : Controller
    {
        [ChildActionOnly]
        public ActionResult ShowMenu(string options)
        {
            return PartialView(viewName: "_ShowMenu", model: options);
        }
    }
}
```

سپس بر روی نام متد کلیک راست کرده و گزینه Add view را انتخاب کنید. در اینجا قصد داریم یک partial view که نامش با underline شروع می‌شود را اضافه کنیم. مثلا با محتوای زیر (با توجه به اینکه مدل ارسالی از نوع رشته‌ای است):

```
@model string

<ul>
    <li>
        @Model
    </li>
</ul>
```

حین فراخوانی متد Html.Action، یک متد در یک کنترلر فراخوانی خواهد شد (که شامل ارائه درخواست و طی سیکل کامل پردازشی آن کنترلر نیز خواهد بود). سپس آن متد با بازگشت دادن یک PartialView، اطلاعات پردازش شده یک partial view را به فراخوان بازگشت می‌دهد. اگر نامی ذکر نشود، همان نام متد در نظر گرفته خواهد شد. البته از آنجائیکه در این مثال در ابتدای نام Partial view یک underline قرار دادیم، نیاز خواهد بود تا این نام صریحا ذکر گردد (چون دیگر هم نام متد یا ActionName آن نیست). ویژگی ChildActionOnly سبب می‌شود تا این متد ویژه تنها از طریق فراخوانی Html.Action در دسترس باشد.
برای استفاده از آن هم در Viewایی دیگر خواهیم داشت:

```
@Html.Action(actionName: "ShowMenu", controllerName: "Menu",
                    routeValues: new { options = "some data..." })
```

در اینجا هم پارامتر ارسالی به کمک anonymously typed objects مشخص و مقدار دهی شده است.

سؤال مهم: چه تفاوتی بین RenderPartial و RenderAction وجود دارد؟ به نظر هر دو یک کار را انجام می‌دهند، هر دو مقداری HTML را پس از پرداش به صفحه تزریق می‌کنند.

پاسخ: اگر View والد، دارای کلیه اطلاعات لازم جهت نمایش اطلاعات Partial view است، از RenderPartial استفاده کنید. به این ترتیب برخلاف حالت RenderAction درخواست جدیدی به ASP.NET Pipeline صادر نشده و کارآیی نهایی بهتر خواهد بود. صرفا یک الحاق ساده به صفحه انجام خواهد شد.

اما اگر برای رندر کردن این قسمت از صفحه که قرار است اضافه شود، نیاز به دریافت اطلاعات دیگری خارج از اطلاعات مهیا می‌باشد، از روش RenderAction استفاده کنید. برای مثال اگر در صفحه جاری قرار است لیست پروژه‌ها نمایش داده شود و در کنار صفحه مثلا منوی خاصی باید قرار گیرد، اطلاعات این منو در View جاری فراهم نیست (و همچنین مرتبط به آن هم نیست). بنابراین از روش RenderAction برای حل این مساله می‌توان کمک گرفت.

به صورت خلاصه برای نمایش اطلاعات تکراری در صفحات مختلف سایت در حالتیکه این اطلاعات از قسمت‌های دیگر صفحه ایزوله است (مثلا نمایش چند ویجت مختلف در صفحه)، روش RenderAction ارجحیت دارد.

یک نکته

فراخوانی متدهای RenderAction و RenderPartial در حین کار با Razor باید به شکل فراخوانی یک متد داخل {} باشند:

```
@{ Html.RenderAction("About"); }
And not @Html.RenderAction("About")
```

علت این است که @ به تنهایی به معنای نوشتن در Response است. متد RenderAction هم خروجی ندارد و مستقیما در Response اطلاعات خودش را درج می‌کند. بنابراین این دو با هم همخوانی ندارند و باید به شکل یک متد معمولی با آن رفتار کرد.

اگر حجم اطلاعاتی که قرار است در صفحه درج شود بالا است، متدهای RenderAction و RenderPartial نسبت به Html.Action و Html.Partial کارآیی بهتری دارند؛ چون یک مرحله تبدیل کل اطلاعات به رشته و سپس درج نتیجه در Response، در آن‌ها حذف شده است.

نظرات خوانندگان

نویسنده: Mojtaba
تاریخ: ۱۳۹۱/۰۱/۲۸ ۰۰:۰۱:۵۴

سلام
ضمن تشکر فراوان از زحماتتان
اگر ممکنه استفاده از DisplayTemplates برای نمایش عمومی فیلد datetime و datetime? رو در قالب یک مثال توضیح دهید
چندین مرتبه روشی رو که فرمودید انجام داد ولی فقط یک پروژه بود که دقیق عمل میکرد آیا نکته خاصی داره
و همچنین بازهم اگر ممکنه لطف کنید چگونگی یک validator رو برای ورود تاریخ شمسی در MVC توضیح دهید
مجددا بسیار سپاسگزارم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۸ ۰۱:۱۳:۳۲

- برای نمایش خودکار تاریخ فارسی می‌تونید از این فایل استفاده کنید (تست شده) : (^)
- برای حالت nullable هم همین فایل کافی است. یعنی در اصل بهتر است DisplayTemplate برای حالت nullable نوشته شود که برای هر دو حالت مورد استفاده قرار خواهد گرفت.

- در مورد validator هم می‌تونید یک attribute سفارشی تهیه کنید. در قسمت 13 راه کلی انجام کار رو توضیح دادم. برای تاریخ شمسی بحث مفصلی است. یک کلاس قبلا در این مورد تهیه کردم : (^) البته این فقط یک ایده برای شروع است که چه فرمت‌هایی می‌تونه وارد بشه و قابل قبول باشه.

نویسنده: Mojtaba
تاریخ: ۱۳۹۱/۰۱/۲۸ ۰۲:۳۹:۰۸

نکته : تفاوت این دو خط منجر به عدم توانایی برنامه برای استفاده ازDisplayTemplates میشود

```
.Html.DisplayFor(m => item.LastLogOutDate) String.Format("{0:g}", item@
LastLogOutDate
(
```

از لطف شما نیز در پاسخ فنی و سریعتان نیز سپاسگزارم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۸ ۱۰:۰۸:۵۳

- اگر قرار است فایل datetime.cshtml فرمت نهایی را اعمال کند، چرا جایی دیگری می‌خواهید آن‌را فرمت کنید که نوع و نحوه نمایش آن کلا عوض شود؟
- اگر علاقمندید از String.Format استفاده کنید، اصلا نیازی به Html.DisplayFor نبوده. برای نمونه در مثال قسمت 12 جاری، فقط کافی است که Model.AddDate را به تابع String.Format ارسال کنید تا g:0 مورد نظر شما اعمال شود. خروجی نهایی هم یک رشته است و دیگر همانند یک DateTime پردازش نمی‌شود.
- در قسمت 13 در مورد ویژگی به نام DisplayFormat توضیح داده شد. توسط آن هم می‌شود DataFormatString را اعمال کرد. در این حالت برای دستیابی به اطلاعات نهایی فرمت شده در یک display template سفارشی باید از ViewData.TemplateInfo.FormattedModelValue استفاده کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۸ ۱۰:۱۴:۵۱

به صورت خلاصه DisplayTemplates فقط به متد Html.DisplayFor اعمال می‌شوند و نه خارج از آن. زمانیکه مستقلا از String.Format استفاده می‌کنید، یعنی خودتان قصد دارید اطلاعات را پردازش و فرمت کنید.

نویسنده: Mojtaba
تاریخ: ۰۲:۴۴:۵۸ ۱۳۹۱/۰۱/۲۹

سلام آقای نصیری خسته نباشید!
اگر بخواهم ازیک jQuery datepicker تاریخ شمسی، مثل آنچه که معرفی کرده اید در EditorTemplates استفاده کنم، چه راه حلی رو پیشنهاد می‌کنید .
ممنونم از لطفتون
ضمنا محبت کنید برای فایل خلاصه‌ی وبلاگ از آخرین تاریخ تهیه به عنوان اسم استفاده کنید تا تاریخ آپدیت آن مشخص شود.

نویسنده: وحید نصیری
تاریخ: ۰۹:۴۶:۲۲ ۱۳۹۱/۰۲/۰۱

می‌تونید داخل templated helper ایی که تعریف می‌کنید یک Html.TextBox اضافه کنید که پارامتر html attributes آن مقدار دهی شده است. چون برای نمونه این datePicker مثلا به TextBox ایی با class خاص اعمال می‌شود. مثلا: "new { @class = "datepicker

نویسنده: Milad Mohseni
تاریخ: ۱۱:۵۷:۰۰ ۱۳۹۱/۰۲/۱۱

با سلام خدمت جناب نصیری
در خصوص این مبحث یک سوالی برایم پیش آمد:
همانطور که فرمودید در ساخت View های مختلف میتوان از Helper های مختلف استفاده کرد. مثلا Html.DropDownListFor یا BeginForm و غیره ...
نهایتاً تمام این ها سمت سرور اجرا شده و نتیجه HTML برمیگرداند. حال به نظر شما مشکلی دارد پس از آنکه با استفاده از Helper های مختلف View نهایی تولید شد و دیگر قصد تغییر آن را نداشتیم، نتیجهٔ HTML شده را جهت استفاده نهایی استفاده کنیم، یعنی در Vew نهایی که بر روی Host آپلود می شود، کد های HTML تولید شده در مرحله قبل را قرار دهیم.
با این کار دیگر لازم نیست سمت سرور Viewها ساخته شوند و همه چیز آماده است؛ که میتواند باعث افزایش سرعت شود. آیا درست متوجه شدم؟
البته قسمت هایی مثل ActionLink ها که با توجه به موقعیت جاری ساخته میشوند به حالت قبل در View نوشته شوند.

نویسنده: وحید نصیری
تاریخ: ۱۲:۰۷:۴۶ ۱۳۹۱/۰۲/۱۱

- Viewها در ASP.NET MVC پس از اولین بار فراخوانی، کامپایل می‌شوند. البته می‌شود با دستکاری فایل پروژه، آن‌ها را در زمان build هم کامپایل کرد. در قسمت‌های قبل توضیح دادم. بنابراین سربار این مساله بسیار کم است.
- شاید برای مواردی مانند تکست باکس، لینک و امثال آن، آنچنان تفاوتی در این بین نباشد که از اصل آن‌ها استفاده شود، یا یک متد کمکی. اما مثلا در مورد رندر کردن یک گرید پویا بر اساس پارامترهای انتخابی یک گزارش چطور؟ هدف اصلی، بیشتر این نوع موارد است.
- مباحث caching اطلاعات را هم در قسمت‌های بعدی به آن اشاره کردم. این مورد سبب می‌شود تا اصلا کدی در سمت سرور اجرا نشود. البته در مورد بایدها و نبایدهای آن هم بحث شده در همان مطلب.

نویسنده: Milad Mohseni
تاریخ: ۱۶:۱۷:۳۰ ۱۳۹۱/۰۲/۱۱

بسیار متشکرم.
کاملاً فهمیدم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۱ ۱۶:۲۵:۵۱

خواهش می‌کنم. یک مورد دیگر هم هست:
- متدی مانند Html.DisplayFor بر اساس خواص مدل‌های برنامه، به صورت Strongly typed تعریف می‌شود (به کمک همین lambda expressions ایی که مشاهده می‌کنید). اگر مدلی تغییر کند، این View دیگر اجرا نخواهد شد و خطای کامپایل رو دریافت می‌کنید. اما در حالت نوشتن مستقیم و معمولی مثلا یک برچسب یا یک تکست باکس و موارد مشابه، این compile time checking رو از دست خواهید داد.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۱/۰۳/۳۱ ۱۱:۳۲

هنگام استفاده از PartialView در صورتی که بخواهیم به فرض از یک Enume استفاده کنیم با توجه به توضیحاتی که در بالا امده است و انرا هنگام نمایش به DropDownList تبدیل کنیم بایستی فقط از EditorForModel استفاده کنیم ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۳۱ ۱۱:۳۸

«فقط» ؟ خیر.
همان قطعه کد Html.DropDownListFor یاد شده را مستقیما در یک View هم می‌تونید استفاده کنید. اینجا فقط یک کپسوله سازی جهت استفاده مجدد بدون تکرار کدها صورت گرفته است.

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۴/۲۰ ۱۱:۵۳

سلام آقای نصیری
با تشکر از توضیحاتتون
من کد زیر رو که خودتون واسه شمسی کردن تاریخ گذاشته بودین، به نام datetime.cshtml و صورت PartialView داخل پوشه DisplayTemplates قرار دادم. اما همچنان DateTimeها رو به فرمت میلادی نمایش میده. میشه بگین اشکال کارم کجاست ؟

```
@using System.Globalization
@model Nullable<DateTime>

@helper ShamsiDateTime(DateTime info, string separator = "/", bool includeHourMinute = true)
{
    int ym = info.Year;
    int mm = info.Month;
    int dm = info.Day;
    var sss = new PersianCalendar();
    int ys = sss.GetYear(new DateTime(ym, mm, dm, new GregorianCalendar()));
    int ms = sss.GetMonth(new DateTime(ym, mm, dm, new GregorianCalendar()));
    int ds = sss.GetDayOfMonth(new DateTime(ym, mm, dm, new GregorianCalendar()));
    if (includeHourMinute)
    {
        @(ys + separator + ms.ToString("00") + separator + ds.ToString("00") + " " + info.Hour + ":" +
info.Minute)
    }
    else
    {
        @(ys + separator + ms.ToString("00") + separator + ds.ToString("00"))
    }
}

@if (@Model.HasValue)
{
  @ShamsiDateTime(@Model.Value , separator: "/", includeHourMinute: false)
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۲۰ ۱۲:۱

در نظرات بالاتر جواب دادم:
«به صورت خلاصه DisplayTemplates فقط به متد Html.DisplayFor اعمال می‌شوند و نه خارج از آن. زمانیکه مستقلا از String.Format استفاده می‌کنید، یعنی خودتان قصد دارید اطلاعات را پردازش و فرمت کنید.»

نویسنده: محسن
تاریخ: ۱۳۹۱/۰۵/۰۱ ۱۴:۲۵

سلام آقای نصیری خسته نباشید
یه سوال از خدمتتون داشتم اونم اینکه اگر توی پروژه جایی نیاز باشه که ما اطلاعات یک فیلد رو از دیتابیس بخونیم و توی یک DropDownList نمایش بدیم، وقتی که میخوایم Value این DropDownList رو سمت کنترلر بفرستیم باید چیکار کنیم؟ منظورم اینه که فرض کنید در جدولی قرار است Username کاربران به عنوان فیلدی ذخیره شود و نام تمامی کاربران را در DropDownList نمایش داده و برای هر کدام Username را به عنوان Value به DropDownList بایند میکنیم. حال مطابق با متد Srongly Type View برای متد Create در کنترلر یک View ایجاد میکنیم. همون طور که میدونید Razor به صورت پیش فرض برای فیلد Username یک EditorFor قرار میده. در صورتی که ما میخوایم یک DropDownList به کاربر نشون بدیم که به راحتی بتونه کاربر مورد نظرش رو انتخاب کنه. حالا چجوری میشه این Username که Value این DropDownList هست رو در موقع کلیک بر روی دکمه ذخیره به سمت کنترلر فرستاد؟ در واقع من نمیدونم اصلا میشه Value رو از FormCollection گرفت یا نه ؟ امیدوارم منظورمو خوب بیان کرده باشم
و یه سوال دیگه اینکه در موقع ویرایش چجوری میشه Vaule ای که در جدول Insert شده رو به این DropDownList بایند کرد جوری که از بین کل مقدارهای بایند شده این Value خاص انتخاب شده باشد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۰۱ ۱۶:۴۲

- یک سری پیشنیاز طراحی رو باید بدونید مانند « کار با کلیدهای اصلی و خارجی در EF Code first ». در اینجا با نحوه تعریف صحیح کلید خارجی به صورت یک خاصیت عددی آشنا خواهید شد. این مورد همان چیزی است که باید از یک drop down list دریافت شود. فقط primary key یک رکورد مهم است نه تمام خواص و فیلدهای مرتبط با آن.
- مرحله بعد ارسال اطلاعات به View هست به این ترتیب:

```
public ActionResult Index()
{
    var query = db.Users.Select(c => new SelectListItem
                                          {
                                              Value = c.UserId,
                                              Text = c.UserName,
                                              Selected = c.UserId.Equals(3)
                                          });
    var model = new MyFormViewModel
                    {
                        List = query.ToList()
                    };
    return View(model);
}
```

در اینجا بر اساس اطلاعات بانک اطلاعاتی SelectListItemها تشکیل شده و به ViewModel فرم جاری ارسال می‌شود (به علاوه باید با ViewModel کار کنید نه مدل جداول بانک اطلاعاتی).

```
public class MyFormViewModel
{
    public int UserId { get; set; }
    public IList<SelectListItem> List { get; set; }
}
```

در مورد نگاشت‌ها هم مباحث auto-mapper در سایت هست.

عنوان: **ASP.NET MVC #13**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۲ ۲۲:۳۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

اعتبار سنجی اطلاعات ورودی در فرم‌های ASP.NET MVC

زمانیکه شروع به دریافت اطلاعات از کاربران کردیم، نیاز خواهد بود تا اعتبار اطلاعات ورودی را نیز ارزیابی کنیم. در ASP.NET MVC، به کمک یک سری متادیتا، نحوه‌ی اعتبار سنجی، تعریف شده و سپس فریم ورک بر اساس این ویژگی‌ها، به صورت خودکار اعتبار اطلاعات انتساب داده شده به خواص یک مدل را در سمت کلاینت و همچنین در سمت سرور بررسی می‌نماید.
این ویژگی‌ها در اسمبلی System.ComponentModel.DataAnnotations.dll قرار دارند که به صورت پیش فرض در هر پروژه جدید ASP.NET MVC لحاظ می‌شود.

یک مثال کاربردی

مدل زیر را به پوشه مدل‌های یک پروژه جدید خالی ASP.NET MVC اضافه کنید:

```
using System;
using System.ComponentModel.DataAnnotations;

namespace MvcApplication9.Models
{
    public class Customer
    {
        public int Id { set; get; }

        [Required(ErrorMessage = "Name is required.")]
        [StringLength(50)]
        public string Name { set; get; }

        [Display(Name = "Email address")]
        [Required(ErrorMessage = "Email address is required.")]
        [RegularExpression(@"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*",
                           ErrorMessage = "Please enter a valid email address.")]
        public string Email { set; get; }

        [Range(0, 10)]
        [Required(ErrorMessage = "Rating is required.")]
        public double Rating { set; get; }

        [Display(Name = "Start date")]
        [Required(ErrorMessage = "Start date is required.")]
        public DateTime StartDate { set; get; }
    }
}
```

سپس کنترلر جدید زیر را نیز به برنامه اضافه نمائید:

```
using System.Web.Mvc;
using MvcApplication9.Models;

namespace MvcApplication9.Controllers
{
    public class CustomerController : Controller
    {
        [HttpGet]
        public ActionResult Create()
        {
            var customer = new Customer();
            return View(customer);
```

```
        }

        [HttpPost]
        public ActionResult Create(Customer customer)
        {
            if (this.ModelState.IsValid)
            {
                //todo: save data
                return Redirect("/");
            }
            return View(customer);
        }
    }
}
```

بر روی متد Create کلیک راست کرده و گزینه Add view را انتخاب کنید. در صفحه باز شده، گزینه Create a strongly typed view را انتخاب کرده و مدل را Customer انتخاب کنید. همچنین قالب Scaffolding را نیز بر روی Create قرار دهید.

توضیحات تکمیلی

همانطور که در مدل برنامه ملاحظه می‌نمائید، به کمک یک سری متادیتا یا اصطلاحا data annotations، تعاریف اعتبار سنجی، به همراه عبارات خطایی که باید به کاربر نمایش داده شوند، مشخص شده است. ویژگی Required مشخص می‌کند که کاربر مجبور است این فیلد را تکمیل کند. به کمک ویژگی StringLength، حداکثر تعداد حروف قابل قبول مشخص می‌شود. با استفاده از ویژگی RegularExpression، مقدار وارد شده با الگوی عبارت باقاعده مشخص گردیده، مقایسه شده و در صورت عدم تطابق، پیغام خطایی به کاربر نمایش داده خواهد شد. به کمک ویژگی Range، بازه اطلاعات قابل قبول، مشخص می‌گردد.
ویژگی دیگری نیز به نام System.Web.Mvc.Compare مهیا است که برای مقایسه بین مقادیر دو خاصیت کاربرد دارد. برای مثال در یک فرم ثبت نام، عموما از کاربر درخواست می‌شود که کلمه عبورش را دوبار وارد کند. ویژگی Compare در یک چنین مثالی کاربرد خواهد داشت.
در مورد جزئیات کنترلر تعریف شده در قسمت 11 مفصل توضیح داده شد. برای مثال خاصیت this.ModelState.IsValid مشخص می‌کند که آیا کارmodel binding موفق بوده یا خیر و همچنین اعتبار سنجی‌های تعریف شده نیز در اینجا تاثیر داده می‌شوند. بنابراین بررسی آن پیش از ذخیره سازی اطلاعات ضروری است.
در حالت HttpGet صفحه ورود اطلاعات به کاربر نمایش داده خواهد شد و در حالت HttpPost، اطلاعات وارد شده دریافت می‌گردد. اگر دست آخر، ModelState معتبر نبود، همان اطلاعات نادرست وارد شده به کاربر مجددا نمایش داده خواهد شد تا فرم پاک نشود و بتواند آن‌ها را اصلاح کند.
برنامه را اجرا کنید. با مراجعه به مسیر http://localhost/customer/create، صفحه ورود اطلاعات کاربر نمایش داده خواهد شد. در اینجا برای مثال در قسمت ورود اطلاعات آدرس ایمیل، مقدار abc را وارد کنید. بلافاصله خطای اعتبار سنجی عدم اعتبار مقدار ورودی نمایش داده می‌شود. یعنی فریم ورک، اعتبار سنجی سمت کاربر را نیز به صورت خودکار مهیا کرده است.
اگر علاقمند باشید که صرفا جهت آزمایش، اعتبار سنجی سمت کاربر را غیرفعال کنید، به فایل web.config برنامه مراجعه کرده و تنظیم زیر را تغییر دهید:

```
<appSettings>
    <add key="ClientValidationEnabled" value="true"/>
```

البته این تنظیم تاثیر سراسری دارد. اگر قصد داشته باشیم که این تنظیم را تنها به یک view خاص اعمال کنیم، می‌توان از متد زیر کمک گرفت:

```
@{ Html.EnableClientValidation(false); }
```

در این حالت اگر مجددا برنامه را اجرا کرده و اطلاعات نادرستی را وارد کنیم، باز هم همان خطاهای تعریف شده، به کاربر نمایش داده خواهد شد. اما اینبار یکبار رفت و برگشت اجباری به سرور صورت خواهد گرفت، زیرا اعتبار سنجی سمت کاربر (که درون مرورگر و توسط کدهای جاوا اسکریپتی اجرا می‌شود)، غیرفعال شده است. البته امکان غیرفعال کردن جاوا اسکریپت توسط کاربر نیز وجود دارد. به همین جهت بررسی خودکار سمت سرور، امنیت سیستم را بهبود خواهد بخشید.

نحوه تعریف عناصر مرتبط با اعتبار سنجی در Viewهای برنامه نیز به شکل زیر است:

```
<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Customer</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>
```

همانطور که ملاحظه می‌کنید به صورت پیش فرض از jQuery validator در سمت کلاینت استفاده شده است. فایل jquery.validate.unobtrusive متعلق به تیم ASP.NET MVC است و کار آن وفق دادن سیستم موجود، با jQuery validator می‌باشد (validation adapter). در نگارش‌های قبلی، از کتابخانه‌های اعتبار سنجی مایکروسافت استفاده شده بود، اما از نگارش سه به بعد، jQuery به عنوان کتابخانه برگزیده مطرح است.
Unobtrusive همچنین در اینجا به معنای مجزا سازی کدهای جاوا اسکریپتی، از سورس HTML صفحه و استفاده از ویژگی‌های data-* مرتبط با HTML5 برای معرفی اطلاعات مورد نیاز اعتبار سنجی است:

```
<input data-val="true" data-val-required="The Birthday field is required." id="Birthday"
name="Birthday" type="text" value="" />
```

اگر خواستید این مساله را بررسی کنید، فایل web.config قرار گرفته در ریشه اصلی برنامه را باز کنید. در آنجا مقدار UnobtrusiveJavaScriptEnabled را false کرده و بار دیگر برنامه را اجرا کنید. در این حالت کلیه کدهای اعتبار سنجی، به داخل سورس View رندر شده، تزریق می‌شوند و مجزا از آن نخواهند بود.
نحوه‌ی تعریف این اسکریپت‌ها نیز جالب توجه است. متد Url.Content، یک متد سمت سرور می‌باشد که در زمان اجرای برنامه، مسیر نسبی وارد شده را بر اساس ساختار سایت اصلاح می‌کند. حرف ~ بکارگرفته شده، در ASP.NET به معنای ریشه سایت است. بنابراین مسیر نسبی تعریف شده از ریشه سایت شروع و تفسیر می‌شود.
اگر از این متد استفاده نکنیم، مجبور خواهیم شد که مسیرهای نسبی را به شکل زیر تعریف کنیم:

```
<script src="../../Scripts/customvaildation.js" type="text/javascript"></script>
```

در این حالت بسته به محل قرارگیری صفحات و همچنین برنامه در سایت، ممکن است آدرس فوق صحیح باشد یا خیر. اما استفاده از متد Url.Content، کار مسیریابی نهایی را خودکار می‌کند.
البته اگر به فایل Views/Shared/_Layout.cshtml، مراجعه کنید، تعریف و الحاق کتابخانه اصلی jQuery در آنجا انجام شده است.

بنابراین می‌توان این دو تعریف دیگر مرتبط با اعتبار سنجی را به آن فایل هم منتقل کرد تا همه‌جا در دسترس باشند.
توسط متد Html.ValidationSummary، خطاهای اعتبار سنجی مدل که به صورت دستی اضافه شده باشند نمایش داده می‌شود. این مورد در قسمت 11 توضیح داده شد (چون پارامتر آن true وارد شده، فقط خطاهای سطح مدل را نمایش می‌دهد).
متد Html.ValidationMessageFor، با توجه به متادیتای یک خاصیت و همچنین استثناهای صادر شده حین model binding خطایی را به کاربر نمایش خواهد داد.

اعتبار سنجی سفارشی

ویژگی‌های اعتبار سنجی از پیش تعریف شده، پر کاربردترین‌ها هستند؛ اما کافی نیستند. برای مثال در مدل فوق، StartDate نباید کمتر از سال 2000 وارد شود و همچنین در آینده هم نباید باشد. این موارد اعتبار سنجی سفارشی را چگونه باید با فریم ورک، یکپارچه کرد؟
حداقل دو روش برای حل این مساله وجود دارد:
الف) نوشتن یک ویژگی اعتبار سنجی سفارشی
ب) پیاده سازی اینترفیس IValidatableObject

تعریف یک ویژگی اعتبار سنجی سفارشی

```
using System;
using System.ComponentModel.DataAnnotations;

namespace MvcApplication9.CustomValidators
{
    public class MyDateValidator : ValidationAttribute
    {
        public int MinYear { set; get; }

        public override bool IsValid(object value)
        {
            if (value == null) return false;

            var date = (DateTime)value;
            if (date > DateTime.Now || date < new DateTime(MinYear, 1, 1))
                return false;

            return true;
        }
    }
}
```

برای نوشتن یک ویژگی اعتبار سنجی سفارشی، با ارث بری از کلاس ValidationAttribute شروع می‌کنیم. سپس باید متد IsValid آن‌را تحریف کنیم. اگر این متد false برگرداند به معنای شکست اعتبار سنجی می‌باشد.
در ادامه برای بکارگیری آن خواهیم داشت:

```
[Display(Name = "Start date")]
[Required(ErrorMessage = "Start date is required.")]
[MyDateValidator(MinYear = 2000,
                 ErrorMessage = "Please enter a valid date.")]
public DateTime StartDate { set; get; }
```

اکنون مجددا برنامه را اجرا نمائید. اگر تاریخ غیرمعتبری وارد شود، اعتبار سنجی سمت سرور رخ داده و سپس نتیجه به کاربر نمایش داده می‌شود.

اعتبار سنجی سفارشی به کمک پیاده سازی اینترفیس IValidatableObject

یک سؤال: اگر اعتبار سنجی ما پیچیده‌تر باشد چطور؟ مثلا نیاز باشد مقادیر دریافتی چندین خاصیت با هم مقایسه شده و سپس بر این اساس تصمیم گیری شود. برای حل این مشکل می‌توان از اینترفیس IValidatableObject کمک گرفت. در این حالت مدل تعریف شده باید اینترفیس یاد شده را پیاده سازی نماید. برای مثال:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using MvcApplication9.CustomValidators;

namespace MvcApplication9.Models
{
    public class Customer : IValidatableObject
    {
        //... same as before

        public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
        {
            var fields = new[] { "StartDate" };
            if (StartDate > DateTime.Now || StartDate < new DateTime(2000, 1, 1))
                yield return new ValidationResult("Please enter a valid date.", fields);

            if (Rating > 4 && StartDate < new DateTime(2003, 1, 1))
                yield return new ValidationResult("Accepted date should be greater than 2003", fields);
        }
    }
}
```

در اینجا در متد Validate، فرصت خواهیم داشت تا به مقادیر کلیه خواص تعریف شده در مدل دسترسی پیدا کرده و بر این اساس اعتبار سنجی بهتری را انجام دهیم. اگر اطلاعات وارد شده مطابق منطق مورد نظر نباشند، کافی است توسط yield return new ValidationResult، یک پیغام را به همراه فیلدهایی که باید این پیغام را نمایش دهند، بازگردانیم.
به این نوع مدل‌ها، validating models self هم گفته می‌شود.

یک نکته:

از MVC3 به بعد، حین کار با ValidationAttribute، امکان تحریف متد IsValid به همراه پارامتری از نوع ValidationContext نیز وجود دارد. به این ترتیب می‌توان به اطلاعات سایر خواص نیز دست یافت. البته در این حالت نیاز به استفاده از Reflection خواهد بود و پیاده سازی IValidatableObject، طبیعی‌تر به نظر می‌رسد:

```
protected override ValidationResult IsValid(object value, ValidationContext validationContext)
{
    var info = validationContext.ObjectType.GetProperty("Rating");
    //...
    return ValidationResult.Success;
}
```

فعال سازی سمت کلاینت اعتبار سنجی‌های سفارشی

اعتبار سنجی‌های سفارشی تولید شده تا به اینجا، تنها سمت سرور است که فعال می‌شوند. به عبارتی باید یکبار اطلاعات به سرور ارسال شده و در بازگشت، نتیجه عملیات به کاربر نمایش داده خواهد شد. اما ویژگی‌های توکاری مانند Required و Range و امثال آن، علاوه بر سمت سرور، سمت کاربر هم فعال هستند و اگر جاوا اسکریپت در مرورگر کاربر غیرفعال نشده باشد، نیازی به ارسال اطلاعات یک فرم به سرور جهت اعتبار سنجی اولیه، نخواهد بود.
در اینجا باید سه مرحله برای پیاده سازی اعتبار سنجی سمت کلاینت طی شود:
الف) ویژگی سفارشی اعتبار سنجی تعریف شده باید اینترفیس IClientValidatable را پیاده سازی کند.

ب) سپس باید متد jQuery validation متناظر را پیاده سازی کرد.
ج) و همچنین مانند تیم ASP.NET MVC، باید unobtrusive adapter خود را نیز پیاده سازی کنیم. به این ترتیب متادیتای ASP.NET MVC به فرمتی که افزونه jQuery validator آن‌را درک می‌کند، وفق داده خواهد شد.

در ادامه، تکمیل کلاس سفارشی MyDateValidator را ادامه خواهیم داد:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Web.Mvc;
using System.Collections.Generic;

namespace MvcApplication9.CustomValidators
{
    public class MyDateValidator : ValidationAttribute, IClientValidatable
    {
       // ... same as before

        public IEnumerable<ModelClientValidationRule> GetClientValidationRules(
                    ModelMetadata metadata,
                    ControllerContext context)
        {
            var rule = new ModelClientValidationRule
            {
                ValidationType = "mydatevalidator",
                ErrorMessage = FormatErrorMessage(metadata.GetDisplayName())
            };
            yield return rule;
        }
    }
}
```

در اینجا نحوه پیاده سازی اینترفیس IClientValidatable را ملاحظه می‌نمائید. ValidationType، نام متدی خواهد بود که در سمت کلاینت، کار بررسی اعتبار داده‌ها را به عهده خواهد گرفت.
سپس برای مثال یک فایل جدید به نام customvaildation.js به پوشه اسکریپت‌های برنامه با محتوای زیر اضافه خواهیم کرد:

```
/// <reference path="jquery-1.5.1-vsdoc.js" />
/// <reference path="jquery.validate-vsdoc.js" />
/// <reference path="jquery.validate.unobtrusive.js" />

jQuery.validator.addMethod("mydatevalidator",
 function (value, element, param) {
     return Date.parse(value) < new Date();
});

jQuery.validator.unobtrusive.adapters.addBool("mydatevalidator");
```

توسط referenceهایی که مشاهده می‌کنید، intellisense جی‌کوئری در VS.NET فعال می‌شود.
سپس به کمک متد jQuery.validator.addMethod، همان مقدار ValidationType پیشین را معرفی و در ادامه بر اساس مقدار value دریافتی، تصمیم گیری خواهیم کرد. اگر خروجی false باشد، به معنای شکست اعتبار سنجی است.
همچنین توسط متد jQuery.validator.unobtrusive.adapters.addBool، این متد جدید را به مجموعه وفق دهنده‌ها اضافه می‌کنیم.
و در آخر این فایل جدید باید به View مورد نظر یا فایل master page سیستم اضافه شود:

```
<script src="@Url.Content("~/Scripts/customvaildation.js")" type="text/javascript"></script>
```

تغییر رنگ و ظاهر پیغام‌های اعتبار سنجی

اگر از رنگ پیش فرض قرمز پیغام‌های اعتبار سنجی خرسند نیستید، باید اندکی CSS سایت را ویرایش کرد که شامل اعمال تغییرات به موارد ذیل خواهد شد:

```
1. .field-validation-error
2. .field-validation-valid
3. .input-validation-error
4. .input-validation-valid
5. .validation-summary-errors
6. .validation-summary-valid
```

نحوه جدا سازی تعاریف متادیتا از کلاس‌های مدل برنامه

فرض کنید مدل‌های برنامه شما به کمک یک code generator تولید می‌شوند. در این حالت هرگونه ویژگی اضافی تعریف شده در این کلاس‌ها پس از تولید مجدد کدها از دست خواهند رفت. به همین منظور امکان تعریف مجزای متادیتاها نیز پیش بینی شده است:

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer
{
    class CustomerMetadata
     {

     }
}

public partial class Customer : IValidatableObject
{
```

حالت کلی روش انجام آن هم به شکلی است که ملاحظه می‌کنید. کلاس اصلی، به صورت partial معرفی خواهد شد. سپس کلاس partial دیگری نیز به همین نام که در برگیرنده یک کلاس داخلی دیگر برای تعاریف متادیتا است، به پروژه اضافه می‌گردد. به کمک ویژگی MetadataType، کلاسی که قرار است ویژگی‌های خواص از آن خوانده شود، معرفی می‌گردد. موارد عنوان شده، شکل کلی این پیاده سازی است. برای نمونه اگر با WCF RIA Services کار کرده باشید، از این روش زیاد استفاده می‌شود. کلاس خصوصی تو در توی تعریف شده صرفا وظیفه ارائه متادیتاهای تعریف شده را به فریم ورک خواهد داشت و هیچ کاربرد دیگری ندارد.
در ادامه کلیه خواص کلاس Customer به همراه متادیتای آن‌ها باید به کلاس CustomerMetadata منتقل شوند. اکنون می‌توان تمام متادیتای کلاس اصلی Customer را حذف کرد.

اعتبار سنجی از راه دور (remote validation)

فرض کنید شخصی مشغول به پر کردن فرم ثبت نام، در سایت شما است. پس از اینکه نام کاربری دلخواه خود را وارد کرد و مثلا به فیلد ورود کلمه عبور رسید، در همین حال و بدون ارسال کل صفحه به سرور، به او پیغام دهیم که نام کاربری وارد شده، هم اکنون توسط شخص دیگری در حال استفاده است. این مکانیزم از ASP.NET MVC3 به بعد تحت عنوان Remote validation در دسترس است و یک درخواست Ajaxایی خودکار را به سرور ارسال خواهد کرد و نتیجه نهایی را به کاربر نمایش می‌دهد؛ کارهایی که به سادگی توسط کدهای جاوا اسکریپتی قابل مدیریت نیستند و نیاز به تعامل با سرور، در این بین وجود دارد. پیاده سازی آن

هم به نحو زیر است:
برای مثال خاصیت Name را در مدل برنامه به نحو زیر تغییر دهید:

```
[Required(ErrorMessage = "Name is required.")]
[StringLength(50)]
[System.Web.Mvc.Remote(action: "CheckUserNameAndEmail",
                               controller: "Customer",
                               AdditionalFields = "Email",
                               HttpMethod = "POST",
                               ErrorMessage = "Username is not available.")]
public string Name { set; get; }
```

سپس متد زیر را نیز به کنترلر Customer اضافه کنید:

```
[HttpPost]
[OutputCache(Location = OutputCacheLocation.None, NoStore = true)]
public ActionResult CheckUserNameAndEmail(string name, string email)
{
      if (name.ToLowerInvariant() == "vahid") return Json(false);
      if (email.ToLowerInvariant() == "name@site.com") return Json(false);
      //...
      return Json(true);
}
```

توضیحات:
توسط ویژگی System.Web.Mvc.Remote، نام کنترلر و متدی که در آن قرار است به صورت خودکار توسط jQuery Ajax فراخوانی شود، مشخص خواهند شد. همچنین اگر نیاز بود فیلدهای دیگری نیز به این متد کنترلر ارسال شوند، می‌توان آن‌ها را توسط خاصیت AdditionalFields، مشخص کرد.
سپس در کدهای کنترلر مشخص شده، متدی با پارامترهای خاصیت مورد نظر و فیلدهای اضافی دیگر، تعریف می‌شود. در اینجا فرصت خواهیم داشت تا برای مثال پس از بررسی بانک اطلاعاتی، خروجی Json ایی را بازگردانیم. return Json false به معنای شکست اعتبار سنجی است.
توسط ویژگی OutputCache، از کش شدن نتیجه درخواست‌های Ajaxایی جلوگیری کرده‌ایم. همچنین نوع درخواست هم جهت امنیت بیشتر، به HttpPost محدود شده است.
تمام کاری که باید انجام شود همین مقدار است و مابقی مسایل مرتبط با اعمال و پیاده سازی آن خودکار است.

استفاده از مکانیزم اعتبار سنجی مبتنی برمتادیتا در خارج از ASP.Net MVC

مباحثی را که در این قسمت ملاحظه نمودید، منحصر به ASP.NET MVC نیستند. برای نمونه توسط متد الحاقی زیر نیز می‌توان یک مدل را مثلا در یک برنامه کنسول هم اعتبار سنجی کرد. بدیهی است در این حالت نیاز خواهد بود تا ارجاعی را به اسمبلی System.ComponentModel.DataAnnotations، به برنامه اضافه کنیم و تمام عملیات هم دستی است و فریم ورک ویژه‌ای هم وجود ندارد تا یک سری از کارها را به صورت خودکار انجام دهد.

```
using System.ComponentModel.DataAnnotations;

namespace MvcApplication9.Helper
{
    public static class ValidationHelper
    {
        public static bool TryValidateObject(this object instance)
        {
            return Validator.TryValidateObject(instance, new ValidationContext(instance, null, null),
```

```
null);
            }
        }
}
```

اینم کدViewها:

```
@model Sama.Models.CustomProfile
@{
    ViewBag.Title = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";

}
<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")"
type="text/javascript"></script>
@using (Html.BeginForm(actionName: "Create", controllerName: "User"))
    {
        @Html.ValidationSummary(true)

        <fieldset>
            <legend>کاربر جدید</legend>
            <div>
                <table>
                    <tr>
                        <td>
                            <div>
                                @Html.LabelFor(model => model.Username, "نام کاربری")
                            </div>
                        </td>
                        <td>
                            <div>
                                @Html.EditorFor(model => model.Username)
                                @Html.ValidationMessageFor(model => model.Username)
                            </div>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <div>
                                @Html.LabelFor(model => model.Password, "کلمه عبور")
                            </div>
                        </td>
                        <td>
                            <div>
                                @Html.PasswordFor(model => model.Password)
                                @Html.ValidationMessageFor(model => model.Password)
                            </div>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <div>
                                @Html.LabelFor(model => model.FirstName, "نام")
                            </div>
                        </td>
                        <td>
                            <div>
                                @Html.EditorFor(model => model.FirstName)
                                @Html.ValidationMessageFor(model => model.FirstName)
                            </div>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <div>
                                @Html.LabelFor(model => model.LastName, "نام خانوادگی")
                            </div>
                        </td>
                        <td>
                            <div>
                                @Html.EditorFor(model => model.LastName)
                                @Html.ValidationMessageFor(model => model.LastName)
                            </div>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <div>
                                @Html.LabelFor(model => model.NationalCode, "کد ملی")
                            </div>
                        </td>
                        <td>
                            <div>
                                @Html.EditorFor(model => model.NationalCode)
```

```
                            @Html.ValidationMessageFor(model => model.NationalCode)
                        </div>
                    </td>
                </tr>
                <tr>
                    <td>
                        <div>
                            @Html.LabelFor(model => model.Email, "ایمیل")
                        </div>
                    </td>
                    <td>
                        <div>
                            @Html.EditorFor(model => model.Email)
                            @Html.ValidationMessageFor(model => model.Email)
                        </div>
                    </td>
                </tr>
                <tr>
                    <td>
                        <div>
                            @Html.LabelFor(model => model.PhoneNo, "تلفن")
                        </div>
                    </td>
                    <td>
                        <div>
                            @Html.EditorFor(model => model.PhoneNo)
                            @Html.ValidationMessageFor(model => model.PhoneNo)
                        </div>
                    </td>
                </tr>
                <tr>
                    <td>
                        <div>
                            @Html.LabelFor(model => model.MobileNo, "موبایل")
                        </div>
                    </td>
                    <td>
                        <div>
                            @Html.EditorFor(model => model.MobileNo)
                            @Html.ValidationMessageFor(model => model.MobileNo)
                        </div>
                    </td>
                </tr>
                <tr>
                    <td>
                        <div>
                            @Html.LabelFor(model => model.Address, "آدرس")
                        </div>
                    </td>
                    <td>
                        <div>
                            @Html.TextAreaFor(model => model.Address, 5, 30, null)
                            @Html.ValidationMessageFor(model => model.Address)
                        </div>
                    </td>
                </tr>
                <tr>
                    <td>
                        نقش‌ها </td>
                    <td>
                        @Helper.CheckBoxList("Roles", (List<SelectListItem>)ViewBag.Roles)
                    </td>
                </tr>
                <tr>
                    <td>
                    </td>
                    <td>
                        <input type="submit" value="ذخیره" class="btn btn-primary" />
                    </td>
                </tr>
            </table>
        </div>
    </fieldset>

}
```

این View برای Create بود و برای Edit هم مشابه همینه

حالا اگر من درست متوجه شده باشم باید برای Edit ، از یک ViewModel دیگه مثلا CustomProfileEdit استفاده کنم و اونجا متد

عنوان: **ASP.NET MVC #14**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۵ ۰۰:۱۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

آشنایی با نحوه معرفی تعاریف طرحبندی سایت به کمک Razor

ممکن است یک سری از اصطلاحات را در قسمت‌های قبل مانند master page در لابلای توضیحات ارائه شده، مشاهده کرده باشید. این نوع مفاهیم برای برنامه نویس‌های ASP.NET Web forms آشنا است (و اگر با Web forms view engine در ASP.NET MVC کار کنید، دقیقا یکی است؛ البته با این تفاوت که فایل code behind آن‌ها حذف شده است). به همین جهت در این قسمت برای تکمیل بحث، مروری خواهیم داشت بر نحوه‌ی معرفی جدید آن‌ها توسط Razor.
در یک پروژه جدید ASP.NET MVC و در پوشه Views\Shared_Layout.cshtml آن، فایل Layout آن، مفهوم master page را دارد. در این نوع فایل‌ها، زیر ساخت مشترک تمام صفحات سایت قرار می‌گیرند:

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
</head>

<body>
    @RenderBody()
</body>
</html>
```

اگر دقت کرده باشید، در هیچکدام از فایل‌های Viewایی که تا این قسمت به پروژه‌های مختلف اضافه کردیم، تگ‌هایی مانند body، title و امثال آن وجود نداشتند. در ASP.NET مرسوم است کلیه اطلاعات تکراری صفحات مختلف سایت را مانند تگ‌های یاد شده به همراه منویی که باید در تمام صفحات قرار گیرد یا footer مشترک بین تمام صفحات سایت، به یک فایل اصلی به نام master page که در اینجا layout نام گرفته، Refactor کنند. به این ترتیب حجم کدها و markup تکراری که باید در تمام Viewهای سایت قرار گیرند به حداقل خواهد رسید.
برای مثال محل قرار گیری تعاریف Content-Type تمام صفحات و همچنین favicon سایت، بهتر است در فایل layout باشد و نه در تک تک Viewهای تعریف شده:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="shortcut icon" href="@Url.Content("~/favicon.ico")" type="image/x-icon" />
```

در کدهای فوق یک نمونه پیش فرض فایل layout را مشاهده می‌کنید. در اینجا توسط متد RenderBody، محتوای رندر شده یک View درخواستی، به داخل تگ body تزریق خواهد شد.
تا اینجا در تمام مثال‌های قبلی این سری، فایل layout در Viewهای اضافه شده معرفی نشد. اما اگر برنامه را اجرا کنیم باز هم به نظر می‌رسد که فایل layout اعمال شده است. علت این است که در صورت عدم تعریف صریح layout در یک View، این تعریف از فایل Views_ViewStart.cshtml دریافت می‌گردد:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

فایل ViewStart، محل تعریف کدهای تکراری است که باید پیش از اجرای هر View مقدار دهی یا اجرا شوند. برای مثال در اینجا می‌شود بر اساس نوع مرورگر، layout خاصی را به تمام Viewها اعمال کرد. مثلا یک layout ویژه برای مرورگرهای موبایل‌ها و layout ایی دیگر برای مرورگرهای معمولی. امکان دسترسی به متغیرهای تعریف شده در یک View در فایل ViewStart از طریق ViewContext.ViewData میسر است.
ضمن اینکه باید درنظر داشت که می‌توان فایل ViewStart را در زیر پوشه‌های پوشه اصلی View نیز قرار داد. مثلا اگر فایل ViewStart ایی در پوشه Views/Home قرار گرفت، این فایل محتوای ViewStart اصلی قرار گرفته در ریشه پوشه Views را بازنویسی خواهد کرد.
برای معرفی صریح فایل layout، تنها کافی است مسیر کامل فایل layout را در یک View مشخص کنیم:

```
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>
```

اهمیت این مساله هم در اینجا است که یک سایت می‌تواند چندین layout یا master page داشته باشد. برای نمونه یک layout برای صفحات ورود اطلاعات؛ یک layout خاص هم مثلا برای صفحات گزارش گیری نهایی سایت.
همانطور که پیشتر نیز ذکر شد، در ASP.NET حرف ~ به معنای ریشه سایت است که در اینجا ابتدای محل جستجوی فایل layout را مشخص می‌کند.
به این ترتیب زمانیکه یک کنترلر، View خاصی را فراخوانی می‌کند، کار از فایل Views\Shared_Layout.cshtml شروع خواهد شد. سپس View درخواستی پردازش شده و محتوای نهایی آن، جایی که متد RenderBody قرار دارد، تزریق خواهد شد.
همچنین مقدار ViewBag.Title ایی که در فایل View تعریف شده، در فایل layout جهت رندر مقدار تگ title استفاده می‌شود (انتقال یک متغیر از View به یک فایل master page؛ کلاس layout، مدل View ایی را که قرار است رندر کند به ارث می‌برد).

یک نکته:
در نگارش سوم ASP.NET MVC امکان بکارگیری حرف ~ به صورت مستقیم در حین تعریف یک فایل js یا css وجود ندارد و حتما باید از متد سمت سرور Url.Content کمک گرفت. در نگارش چهارم ASP.NET MVC، این محدودیت برطرف شده و دقیقا همانند متغیر Layout ایی که در بالا مشاهده می‌کنید، می‌توان بدون نیاز به متد Url.Content، مستقیما از حرف ~ کمک گرفت و به صورت خودکار پردازش خواهد شد.

تزریق نواحی ویژه یک View در فایل layout

توسط متد RenderBody، کل محتوای View درخواستی در موقعیت تعریف شده آن در فایل Layout، رندر می‌شود. این ویژگی به نحو یکسانی به تمام Viewها اعمال می‌شود. اما اگر نیاز باشد تا view بتواند محتوای markup قسمت ویژه‌ای از master page را مقدار دهی کند، می‌توان از مفهومی به نام Sections استفاده کرد:

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
</head>
<body>
```

```
    <div id="menu">
        @if (IsSectionDefined("Menu"))
        {
            RenderSection("Menu", required: false);
        }
        else
        {
            <span>This is the default ...!</span>
        }
    </div>
    <div id="body">
        @RenderBody()
    </div>
</body>
</html>
```

در اینجا ابتدا بررسی می‌شود که آیا قسمتی به نام Menu در View جاری که باید رندر شود وجود دارد یا خیر. اگر بله، توسط متد RenderSection، آن قسمت نمایش داده خواهد شد. در غیراینصورت، محتوای پیش فرضی را در صفحه قرار می‌دهد. البته اگر از متد RenderSection با آرگومان required: false استفاده شود، درصورتیکه View جاری حاوی قسمتی به نام مثلا menu نباشد، تنها چیزی نمایش داده نخواهد شد. اگر این آرگومان را حذف کنیم، یک استثنای عدم یافت شدن ناحیه یا قسمت مورد نظر صادر می‌گردد.

نحوه‌ی تعریف یک Section در Viewهای برنامه به شکل زیر است:

```
@{
    ViewBag.Title = "Index";
    //Layout = null;
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>
    Index</h2>
@section Menu{
    <ul>
        <li>item 1</li>
        <li>item 2</li>
    </ul>
}
```

برای مثال فرض کنید که یکی از Viewهای شما نیاز به دو فایل اضافی جاوا اسکریپت برای اجرای صحیح خود دارد. می‌توان تعاریف الحاق این دو فایل را در قسمت header فایل layout قرار داد تا در تمام Viewها به صورت خودکار لحاظ شوند. یا اینکه یک section سفارشی را به نحو زیر در آن View خاص تعریف می‌کنیم:

```
@section JavaScript
{
   <script type="text/javascript" src="@Url.Content("~/Scripts/SomeScript.js")" />;
   <script type="text/javascript" src="@Url.Content("~/Scripts/AnotherScript.js")" />;
}
```

سپس کافی است جهت تزریق این کدها به header تعریف شده در master page مورد نظر، یک سطر زیر را اضافه کرد:

```
@RenderSection("JavaScript", required: false)
```

به این ترتیب، اگر view ایی حاوی تعریف قسمت JavaScript نبود، به صورت خودکار شامل تعاریف الحاق اسکریپت‌های یاد شده نیز نخواهد گردید. در نتیجه دارای حجمی کمتر و سرعت بارگذاری بالاتری نیز خواهد بود.

مدیریت بهتر فایل‌ها و پوشه‌های یک برنامه ASP.NET MVC به کمک Areas

به کمک قابلیتی به نام Areas می‌توان یک برنامه بزرگ را به چندین قسمت کوچکتر تقسیم کرد. هر کدام از این نواحی، دارای تعاریف مسیریابی، کنترلرها و Viewهای خاص خودشان هستند. به این ترتیب دیگر به یک برنامه‌ی از کنترل خارج شده ASP.NET MVC که دارای یک پوشه Views به همراه صدها زیر پوشه است، نخواهیم رسید و کنترل این نوع برنامه‌های بزرگ ساده‌تر خواهد شد.

برای مثال یک برنامه بزرگ را درنظر بگیرید که به کمک قابلیت Areas، به نواحی ویژه‌ای مانند گزارشگیری، قسمت ویژه مدیریتی، قسمت کاربران، ناحیه بلاگ سایت، ناحیه انجمن سایت و غیره، تقسیم شده است. به علاوه هر کدام از این نواحی نیز هنوز می‌توانند از اطلاعات ناحیه اصلی برنامه مانند master page آن استفاده کنند. البته باید درنظر داشت که فایل viewStart به پوشه جاری و زیر پوشه‌های آن اعمال می‌شود. اگر نیاز باشد تا اطلاعات این فایل به کل برنامه اعمال شود، فقط کافی است آن‌را به یک سطح بالاتر، یعنی ریشه سایت منتقل کرد.

نحوه افزودن نواحی جدید

افزودن یک Area جدید هم بسیار ساده است. بر روی نام پروژه در VS.NET کلیک راست کرده و سپس گزینه Add|Area را انتخاب کنید. سپس در صفحه باز شده، نام دلخواهی را وارد نمائید. مثلا نام Reporting را وارد نمائید تا ناحیه گزارشگیری برنامه از قسمت‌های دیگر آن مستقل شود. پس از افزودن یک Area جدید، به صورت خودکار پوشه جدیدی به نام Areas به ریشه سایت اضافه می‌شود. سپس داخل آن، پوشه‌ی دیگری به نام Reporting اضافه خواهد شد. پوشه reporting اضافه شده هم دارای پوشه‌های Views ،Model و Controllers خاص خود می‌باشد.

اکنون که پوشه Areas به ریشه سایت اضافه شده است، با کلیک راست بر روی این پوشه نیز گزینه‌ی Add|Area در دسترس می‌باشد. برای نمونه یک ناحیه جدید دیگر را به نام Admin به سایت اضافه کنید تا بتوان امکانات مدیریتی سایت را از سایر قسمت‌های آن مستقل کرد.

نحوه معرفی تعاریف مسیریابی نواحی تعریف شده

پس از اینکه کار با Areas را آغاز کردیم، نیاز است تا با نحوه‌ی مسیریابی آن‌ها نیز آشنا شویم. برای این منظور فایل Global.asax.cs قرار گرفته در ریشه اصلی برنامه را باز کنید. در متد Application_Start، متدی به نام AreaRegistration.RegisterAllAreas، کار ثبت و معرفی تمام نواحی ثبت شده را به فریم ورک، به عهده دارد. کاری که در پشت صحنه انجام خواهد شد این است که به کمک Reflection تمام کلاس‌های مشتق شده از کلاس پایه AreaRegistration به صورت خودکار یافت شده و پردازش خواهند شد. این کلاس‌ها هم به صورت پیش فرض به نام SomeNameAreaRegistration.cs در ریشه اصلی هر Area توسط VS.NET تولید می‌شوند. برای نمونه فایل ReportingAreaRegistration.cs تولید شده، حاوی اطلاعات زیر است:

```
using System.Web.Mvc;

namespace MvcApplication11.Areas.Reporting
{
    public class ReportingAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "Reporting";
            }
        }

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "Reporting_default",
                "Reporting/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional }
            );
        }
    }
```

```
}
```

توسط AreaName، یک نام منحصربفرد در اختیار فریم ورک قرار خواهد گرفت. همچنین از این نام برای ایجاد پیوند بین نواحی مختلف نیز استفاده می‌شود.
سپس در قسمت RegisterArea، یک مسیریابی ویژه خاص ناحیه جاری مشخص گردیده است. برای مثال تمام آدرس‌های ناحیه گزارشگیری سایت باید با http://localhost/reporting آغاز شوند تا مورد پردازش قرارگیرند. سایر مباحث آن هم مانند قبل است. برای مثال در اینجا نام اکشن متد پیش فرض، index تعریف شده و همچنین ذکر قسمت id نیز اختیاری است.
همانطور که ملاحظه می‌کنید، تعاریف مسیریابی و اطلاعات پیش فرض آن منطقی هستند و آنچنان نیازی به دستکاری و تغییر ندارند. البته اگر دقت کرده باشید مقدار نام controller پیش فرض، مشخص نشده است. بنابراین بد نیست که مثلا نام Home یا هر نام مورد نظر دیگری را به عنوان نام کنترلر پیش فرض در اینجا اضافه کرد.

تعاریف کنترلرهای هم نام در نواحی مختلف

در ادامه مثال جاری که دو ناحیه Admin و Reporting به آن اضافه شده، به پوشه‌های Controllers هر کدام، یک کنترلر جدید را به نام HomeController اضافه کنید. همچنین این HomeController را در ناحیه اصلی و ریشه سایت نیز اضافه نمائید. سپس برای متد پیش فرض Index هر کدام هم یک View جدید را با کلیک راست بر روی نام متد و انتخاب گزینه Add view، اضافه کنید. اکنون برنامه را به همین نحو اجرا نمائید. اجرای برنامه با خطای زیر متوقف خواهد شد:

```
Multiple types were found that match the controller named 'Home'. This can happen if the route that
services this
request ('{controller}/{action}/{id}') does not specify namespaces to search for a controller that
matches the request.
If this is the case, register this route by calling an overload of the 'MapRoute' method that takes a
'namespaces' parameter.

The request for 'Home' has found the following matching controllers:
MvcApplication11.Areas.Admin.Controllers.HomeController
MvcApplication11.Controllers.HomeController
```

فوق العاده خطای کاملی است و راه حل را هم ارائه داده است! برای اینکه مشکل ابهام یافتن HomeController برطرف شود، باید این جستجو را به فضاهای نام هر قسمت از نواحی برنامه محدود کرد (چون به صورت پیش فرض فضای نامی برای آن مشخص نشده، کل ناحیه ریشه سایت و زیر مجموعه‌های آن‌را جستجو خواهد کرد). به همین جهت فایل Global.asax.cs را گشوده و متد RegisterRoutes آن‌را مثلا به نحو زیر اصلاح نمائید:

```
public static void RegisterRoutes(RouteCollection routes)
{
     routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

     routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Parameter
defaults
                , namespaces: new[] { "MvcApplication11.Controllers" }
            );
}
```

آرگومان چهارم معرفی شده، آرایه‌ای از نام‌های فضاهای نام مورد نظر را جهت یافتن کنترلرهایی که باید توسط این مسیریابی یافت شوند، تعریف می‌کند.
اکنون اگر مجددا برنامه را اجرا کنیم، بدون مشکل View متناظر با متد Index کنترلر Home نمایش داده خواهد شد.
البته این مشکل با نواحی ویژه و غیر اصلی سایت وجود ندارد؛ چون جستجوی پیش فرض کنترلرها بر اساس ناحیه است.

در ادامه مسیر http://localhost/Admin/Home را نیز در مرورگر وارد کنید. سپس بر روی صفحه در مروگر کلیک راست کرده و سورس صفحه را بررسی کنید. مشاهده خواهید کرد که master page یا فایل layout ایی به آن اعمال نشده است. علت را هم در ابتدای بحث Areas مطالعه کردید. فایل Views_ViewStart.cshtml در سطحی که قرار دارد به ناحیه Admin اعمال نمی‌شود. آن‌را به ریشه سایت منتقل کنید تا layout اصلی سایت نیز به این قسمت اعمال گردد. البته بدیهی است که هر ناحیه می‌تواند layout خاص خودش را داشته باشد یا حتی می‌توان با مقدار دهی خاصیت Layout نیز در هر view، فایل master page ویژه‌ای را انتخاب و معرفی کرد.

نحوه ایجاد پیوند بین نواحی مختلف سایت

زمانیکه پیوندی را به شکل زیر تعریف می‌کنیم:

```
@Html.ActionLink(linkText: "Home", actionName: "Index", controllerName: "Home")
```

یعنی ایجاد لینکی در ناحیه جاری. برای اینکه پیوند تعریف شده به ناحیه‌ای خارج از ناحیه جاری اشاره کند باید نام Area را صریحا ذکر کرد:

```
@Html.ActionLink(linkText: "Home", actionName: "Index", controllerName: "Home",
                 routeValues: new { Area = "Admin" } , htmlAttributes: null)
```

همین نکته را باید حین کار با متد RedirectToAction نیز درنظر داشت:

```
public ActionResult Index()
{
    return RedirectToAction("Index", "Home", new { Area = "Admin" });
}
```

عنوان: **ASP.NET MVC #15**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۶ ۲۳:۱۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

فیلترها در ASP.NET MVC

پایه قسمت‌های بعدی مانند مباحث امنیت، اعتبار سنجی کاربران، caching و غیره، مبحثی است به نام فیلترها در ASP.NET MVC. تابحال با سه فیلتر به نام‌های NonAction، ActionName و AcceptVerbs آشنا شده‌ایم. به این‌ها Action selector filters هم گفته می‌شود. زمانیکه قرار است یک درخواست رسیده به متدی در یک کنترلر خاص نگاشت شود، فریم ورک ابتدا به متادیتای اعمالی به متدها توجه کرده و بر این اساس درخواست را به متدی صحیح هدایت خواهد کرد. ActionName، نام پیش فرض یک متد را بازنویسی می‌کند و توسط AcceptVerbs اجرای یک متد، به افعالی مانند DELETE، GET، POST و امثال آن محدود می‌شود که در قسمت‌های قبل در مورد آن‌ها بحث شد.
علاوه بر این‌ها یک سری فیلتر دیگر نیز در ASP.NET MVC وجود دارند که آن‌ها نیز به شکل متادیتا به متدهای کنترلرها اعمال شده و کار نهایی‌اشان تزریق کدهایی است که باید پیش و پس از اجرای یک اکشن متد، اجرا شوند. 4 نوع فیلتر در ASP.NET MVC وجود دارند:

الف) IAuthorizationFilter
این نوع فیلترها پیش از اجرای هر متد یا فیلتر دیگری در کنترلر جاری اجرا شده و امکان لغو اجرای آن‌را فراهم می‌کنند. پیاده سازی پیش‌فرض آن توسط کلاس AuthorizeAttribute در فریم ورک وجود دارد.
بدیهی است این نوع اعمال را مستقیما داخل متدهای کنترلرها نیز می‌توان انجام داد (بدون نیاز به هیچگونه فیلتری). اما به این ترتیب حجم کدهای تکراری در سراسر برنامه به شدت افزایش می‌یابد و نگهداری آن‌را در طول زمان مشکل خواهد ساخت.

ب) IActionFilter
ActionFilterها پیش (OnActionExecuting) و پس از (OnActionExecuted) اجرای متدهای کنترلر جاری اجرا می‌شوند و همچنین پیش از ارائه خروجی نهایی متدها. به این ترتیب برای مثال می‌توان نحوه رندر یک View را تحت کنترل گرفت. این اینترفیس توسط کلاس ActionFilterAttribute در فریم ورک پیاده سازی شده است.

ج) IResultFilter
ResultFilter بسیار شبیه به ActionFilter است با این تفاوت که تنها پیش از (OnResultExecuting) بازگرداندن نتیجه متد و همچنین پس از (OnResultExecuted) اجرای متد، فراخوانی می‌گردد. کلاس ActionFilterAttribute موجود در فریم ورک، پیاده سازی پیش فرضی از آن‌را ارائه می‌دهد.

د) IExceptionFilter
ExceptionFilterها پس از اجرای تمامی فیلترهای دیگر، همواره اجرا خواهند شد؛ صرفنظر از اینکه آیا در این بین استثنایی رخ داده است یا خیر. بنابراین یکی از کاربردهای آن‌ها می‌تواند ثبت وقایع مرتبط با استثناهای رخداده باشد. پیاده سازی پیش فرض آن توسط کلاس HandleErrorAttribute در فریم ورک موجود است.

علت معرفی 4 نوع فیلتر متفاوت هم به مسایل امنیتی بر می‌گردد. می‌شد تنها موارد ب و ج معرفی شوند اما از آنجائیکه نیاز است مورد الف همواره پیش از اجرای متدی و همچنین تمامی فیلترهای دیگر فراخوانی شود، احتمال بروز اشتباه در نحوه و ترتیب معرفی این فیلترها وجود داشت. به همین دلیل روش معرفی صریح مورد الف در پیش گرفته شد. برای مثال فرض کنید که اگر از روی اشتباه فیلتر کش شدن اطلاعات پیش از فیلتر اعتبار سنجی کاربر جاری اجرا می‌شد چه مشکلات امنیتی ممکن بود بروز کند.

مثالی جهت درک بهتر ترتیب و نحوه اجرای فیلترها:

یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. سپس فیلتر سفارشی زیر را به برنامه اضافه نمائید:

```
using System.Diagnostics;
using System.Web.Mvc;

namespace MvcApplication12.CustomFilters
{
    public class LogAttribute : ActionFilterAttribute
    {
        public override void OnActionExecuting(ActionExecutingContext filterContext)
        {
            Log("OnActionExecuting", filterContext);
        }

        public override void OnActionExecuted(ActionExecutedContext filterContext)
        {
            Log("OnActionExecuted", filterContext);
        }

        public override void OnResultExecuting(ResultExecutingContext filterContext)
        {
            Log("OnResultExecuting", filterContext);
        }

        public override void OnResultExecuted(ResultExecutedContext filterContext)
        {
            Log("OnResultExecuted", filterContext);
        }

        private void Log(string stage, ControllerContext ctx)
        {
            ctx.HttpContext.Response.Write(
                string.Format("{0}:{1} - {2} < br/> ",
                ctx.RouteData.Values["controller"], ctx.RouteData.Values["action"], stage));
        }
    }
}
```

مرسوم است برای ایجاد فیلترهای سفارشی، همانند مثال فوق با ارث بری از پیاده سازی‌های توکار اینترفیس‌های چهارگانه یاد شده، کار شروع شود.
سپس یک کنترلر جدید را به همراه دو متد، به برنامه اضافه نمائید. برای هر کدام از متدها هم یک View خالی را ایجاد کنید. اکنون این ویژگی جدید را به هر کدام از این متدها اعمال نموده و برنامه را اجرا کنید.

```
using System.Web.Mvc;
using MvcApplication12.CustomFilters;

namespace MvcApplication12.Controllers
{
    public class HomeController : Controller
    {
        [Log]
        public ActionResult Index()
        {
            return View();
        }

        [Log]
        public ActionResult Test()
        {
            return View();
        }
    }
}
```

سپس ویژگی Log را از متدها حذف کرده و به خود کنترلر اعمال کنید:

```
[Log]
public class HomeController : Controller
```

در این حالت ویژگی اعمالی، پیش از اجرای متد درخواستی جاری اجرا خواهد شد یا به عبارتی به تمام متدهای قابل دسترسی کنترلر اعمال می‌گردد.

تقدم و تاخر اجرای فیلترهای هم‌خانواده

همانطور که عنوان شد، همیشه ابتدا AuthorizationFilter اجرا می‌شود و در آخر ExceptionFilter. سؤال: اگر در این بین مثلا دو نوع ActionFilter متفاوت به یک متد اعمال شدند، کدامیک ابتدا اجرا می‌شود؟
تمام فیلترها از کلاسی به نام FilterAttribute مشتق می‌شوند که دارای خاصیتی است به نام Order. بنابراین جهت مشخص سازی ترتیب اجرای فیلترها تنها کافی است این خاصیت مقدار دهی شود. برای مثال جهت اعمال دو فیلتر سفارشی زیر:

```
using System.Diagnostics;
using System.Web.Mvc;

namespace MvcApplication12.CustomFilters
{
    public class AuthorizationFilterA : AuthorizeAttribute
    {
        public override void OnAuthorization(AuthorizationContext filterContext)
        {
            Debug.WriteLine("OnAuthorization : AuthorizationFilterA");
        }
    }
}
```

```
using System.Diagnostics;
using System.Web.Mvc;

namespace MvcApplication12.CustomFilters
{
    public class AuthorizationFilterB : AuthorizeAttribute
    {
        public override void OnAuthorization(AuthorizationContext filterContext)
        {
            Debug.WriteLine("OnAuthorization : AuthorizationFilterB");
        }
    }
}
```

خواهیم داشت:

```
using System.Web.Mvc;
using MvcApplication12.CustomFilters;

namespace MvcApplication12.Controllers
{
    public class HomeController : Controller
    {
        [AuthorizationFilterA(Order = 2)]
        [AuthorizationFilterB(Order = 1)]
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

در اینجا با توجه به مقادیر order، ابتدا AuthorizationFilterB اجرا می‌گردد و سپس AuthorizationFilterA.
علاوه بر این‌ها محدوده اجرای فیلترها نیز بر بر این حق تقدم اجرایی تاثیر گذار هستند. برای مثال در پشت صحنه زمانیکه قرار است یک فیلتر جدید اجرا شود، وهله سازی آن به نحوه زیر است که بر اساس مقادیر order و FilterScope صورت می‌گیرد:

```
var filter = new Filter(actionFilter, FilterScope, order);
```

مقادیر FilterScope را در ادامه ملاحظه می‌نمائید:

```
namespace System.Web.Mvc {
      public enum FilterScope {
            First = 0,
            Global = 10,
            Controller = 20,
            Action = 30,
            Last = 100,
        }
}
```

به صورت پیش فرض، ابتدا فیلتری با محدوده اجرای کمتر، اجرا خواهد شد. در اینجا Global به معنای اجرای شدن در تمام کنترلرها است.

تعریف فیلترهای سراسری

برای اینکه فیلتری را عمومی و سراسری تعریف کنیم، تنها کافی است آن‌را در متد Application_Start فایل Global.asax.cs به نحو زیر معرفی نمائیم:

```
GobalFilters.Filters.Add(new AuthorizationFilterA() { Order = 2});
```

به این ترتیب AuthorizationFilterA، به تمام کنترلرها و متدهای قابل دسترسی آن‌ها در برنامه به صورت خودکار اعمال خواهد شد.
یکی از کاربردهای فیلترهای سراسری، نوشتن برنامه‌های پروفایلر است. برنامه‌هایی که برای مثال مدت زمان اجرای متدها را ثبت کرده و بر این اساس بهتر می‌توان کارآیی قسمت‌های مختلف برنامه را دقیقا زیرنظر قرار داد.

یک نکته

کلاس کنترلر در ASP.NET MVC نیز یک فیلتر است:

```
public abstract class Controller : ControllerBase, IActionFilter, IAuthorizationFilter, IDisposable,
IExceptionFilter, IResultFilter
```

به همین دلیل، امکان تحریف متدهای OnActionExecuting، OnActionExecuted و امثال آن که پیشتر ذکر شد، در یک کنترلر نیز وجود دارد.
کلاس کنترلر دارای محدوده اجرایی First و Order ایی مساوی Int32.MinValue است. به این ترتیب کنترلرها پیش از اجرای هر فیلتر دیگری اجرا خواهند شد.

ASP.NET MVC دارای یک سری فیلتر و متادیتای توکار مانند ValidateInpute، RequireHttps، HandleError، OutputCache و غیره است که توضیحات بیشتر آن‌ها به قسمت‌های بعد موکول می‌گردد.

عنوان: ASP.NET MVC #16
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۷ ۲۳:۴۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

مدیریت خطاها در یک برنامه ASP.NET MVC

استفاده از فیلتر HandleError

یکی از فیلترهای توکار ASP.NET MVC به نام HandleError، می‌تواند کار هدایت کاربر را به یک صفحه‌ی خطای عمومی، در حین بروز استثنایی در برنامه، انجام دهد. برای آزمایش آن یک برنامه خالی جدید ASP.NET MVC را آغاز کنید. سپس یک کنترلر جدید را با محتوای زیر به آن اضافه نمائید:

```
using System;
using System.Web.Mvc;

namespace MvcApplication13.Controllers
{
    public class HomeController : Controller
    {
        [HandleError]
        public ActionResult Index()
        {
            throw new InvalidOperationException();
            return View();
        }
    }
}
```

در اینجا جهت آزمایش برنامه، به عمد یک استثنای دستی را صادر می‌کنیم. برای آزمایش برنامه هم نیاز است آن‌را خارج از دیباگر VS.NET اجرا کرد (آدرس برنامه را مستقیما خارج از VS.NET در یک مرورگر وارد کنید). همچنین یک سطر زیر را نیز لازم است به فایل web.config برنامه اضافه نمائید:

```
<system.web>
   <customErrors mode="On" />
```

اکنون اگر برنامه را خارج از مرورگر اجرا کنید، با توجه به استفاده از ویژگی HandleError و همچنین بروز یک استثنا در متد Index، خودبخود صفحه Views\Shared\Error.cshtml به کاربر نمایش داده خواهد شد. در غیراینصورت صفحه زرد رنگ پیش فرض خطای ASP.NET به کاربر نمایش داده می‌شود که محتوای آن‌ها بیشتر برای برنامه نویس‌ها مناسب است و نه کاربران نهایی سیستم.
اگر علاقمند باشید که این ویژگی به صورت خودکار به تمام متدهای کنترلرهای برنامه اعمال شود، کافی است یک سطر زیر را به متد Application_Start فایل Global.asax.cs اضافه نمائید:

```
GlobalFilters.Filters.Add(new HandleErrorAttribute());
```

البته نیازی به انجام اینکار نیست زیرا اگر به متد RegisterGlobalFilters فایل Global.asax.cs دقت کنیم، اینکار پیشتر توسط قالب پیش فرض VS.NET انجام شده است. فقط برای فعال سازی آن نیاز است تگ customErrors در فایل وب کانفیگ برنامه مقدار دهی و تنظیم شود.

استفاده از صفحه خطای سفارشی دیگری بجای فایل Error.cshtml

امکان تنظیم نمایش صفحه خطای سفارشی دیگری نیز وجود دارد. برای مثال استفاده از فایل Views\Shared\CustomErrorView.cshtml :

```
[HandleError(View = "CustomErrorView")]
```

استفاده از صفحات خطای متفاوت به ازای استثناهای مختلف

می‌توان فیلتر HandleError را تنها به یک نوع استثنای خاص محدود کرد. همچنین امکان استفاده از چندین ویژگی HandleError برای یک متد نیز وجود دارد:

```
[HandleError(ExceptionType = typeof(NullReferenceException), View = "ErrorHandling")]
```

دسترسی به اطلاعات استثناء در صفحه نمایش خطاها

زمانیکه برنامه به صفحه خطا هدایت می‌شود، نوع Model آن System.Web.Mvc.HandleErrorInfo می‌باشد:

```
@model System.Web.Mvc.HandleErrorInfo

@{
    ViewBag.Title = "DbError";
}

<h2>An Error Has Occurred</h2>

@if (Model != null)
{
    <p>@Model.Exception.GetType().Name<br />
    thrown in @Model.ControllerName @Model.ActionName</p>
}
```

البته این نکته را صرفا به عنوان اطلاعات عمومی در نظر داشته باشید. زیرا اگر قرار باشد مجددا اصل استثناء را نمایش دهیم، همان صفحه زرد رنگ ASP.NET شاید بهتر باشد.

استفاده از تگ customErrors در فایل Web.config برنامه

ویژگی حالت تگ customErrors در فایل web.config برنامه، سه مقدار را می‌تواند بپذیرد:
الف) Off : صفحه زرد رنگ معرفی خطای ASP.NET را به همراه تمام اطلاعات مرتبط با استثنای رخ داده نمایش می‌دهد.
ب) RemoteOnly : همان حالت الف است با این تفاوت که صفحه خطا را فقط در کامپیوتری که وب سرور بر روی آن نصب است

نمایش خواهد داد.
ج) On : یک صفحه خطای سفارشی شده را نمایش می‌دهد.

بنابراین هیچگاه از حالت Off استفاده نکنید. زیرا خطاهای نمایش داده شده، علاوه بر برنامه نویس، برای مهاجم به یک سایت نیز بسیار دلپذیر است!
حالت RemoteOnly در زمان توسعه برنامه توصیه می‌شود.
حالت On حین توزیع برنامه باید بکارگرفته شود.

مدیریت خطاهای رخ داده خارج از MVC Pipeline

HandleErrorAttribute تنها استثناهای رخ داده داخل ASP.NET MVC Pipeline را مدیریت می‌کند (یا خطاهایی از نوع 500). اگر این نوع استثناها خارج از آن رخ دهند مثلا فایلی یافت نشود (خطای 404) و امثال آن، باید به روش زیر عمل کرد:

```
<customErrors mode="On" defaultRedirect="error">
      <error statusCode="404" redirect="error/notfound" />
      <error statusCode="403" redirect="error/forbidden" />
</customErrors>
```

در اینجا اگر فایلی یافت نشد، کاربر به کنترلری به نام error و متدی به نام notfound هدایت خواهد شد. بنابراین نیاز به کنترلر زیر وجود دارد؛ به علاوه به ازای هر متد هم یک View متناظر باید اضافه شود (کلیک راست روی نام متد و انتخاب گزینه افزودن View جدید).

```
using System.Web.Mvc;

namespace MvcApplication13.Controllers
{
    public class ErrorController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult NotFound()
        {
            return View();
        }

        public ActionResult Forbidden()
        {
            return View();
        }
    }
}
```

برای آزمایش این قسمت، برنامه را اجرا کرده و سپس مثلا آدرس غیرموجود http://localhost/xyz را وارد کنید.

استفاده از فیلتر HandleError اجباری نیست

در همین قسمت قبل پس از افزودن customErrors و defaultRedirect آن که به نام یک کنترلر اشاره می‌کند، کلیه فیلترهای HandleError اضافه شده به برنامه را حذف کنید. سپس برنامه را خارج از محیط VS.NET اجرا کنید. باز هم متد Index کنترلر Error اجرا خواهد شد. به عبارتی الزاما نیازی به استفاده از فیلتر HandleError نیست و به کمک مقدار دهی صحیح تگ customErrors، کار نمایش خودکار صفحه سفارشی خطاها به کاربر انجام خواهد شد.

البته بدیهی است که گزینه‌های نمایش یک View خاص به ازای استثنایی ویژه، یکی از مزیت‌های استفاده از فیلتر HandleError می‌باشد که امکان تنظیم آن در فایل web.config وجود ندارد.

ثبت اطلاعات استثناهای رخ داده به کمک ELMAH

نمایش صفحه‌ی خطای سفارشی به کاربر، یکی از موارد ضروری تمام برنامه‌های ASP.NET است، اما کافی نیست. ثبت اطلاعات جزئیات استثناهای رخ داده در طول زمان می‌توانند به بالا بردن کیفیت برنامه به شدت کمک کنند. برای این منظور می‌توان همانند سابق از متد Application_Error قابل تعریف در فایل Global.asax.cs کمک گرفت؛ اما با وجود افزونه‌ای به نام ELMAH اینکار اتلاف وقت است و اصلا توصیه نمی‌شود. همچنین به کمک ELMAH می‌توان مشکلات را تبدیل به ایمیل‌های خودکار کرد یا از آن‌ها فید RSS درست نمود.
برای دریافت ELMAH یا به سایت اصلی آن مراجعه نمائید و یا به کمک NuGet هم به سادگی قابل دریافت است. پس از دریافت، ارجاعی را به اسمبلی آن (Elmah.dll) اضافه نمائید. در ادامه فایل web.config برنامه را گشوده و چند سطر زیر را به آن در قسمت configuration اضافه کنید:

```
<configuration>
  <configSections>
    <sectionGroup name="elmah">
      <section name="security" requirePermission="false" type="Elmah.SecuritySectionHandler, Elmah"/>
      <section name="errorLog" requirePermission="false" type="Elmah.ErrorLogSectionHandler, Elmah"/>
      <section name="errorMail" requirePermission="false" type="Elmah.ErrorMailSectionHandler, Elmah"/>
      <section name="errorFilter" requirePermission="false" type="Elmah.ErrorFilterSectionHandler,
Elmah"/>
      <section name="errorTweet" requirePermission="false" type="Elmah.ErrorTweetSectionHandler,
Elmah"/>
    </sectionGroup>
  </configSections>
```

سپس ذیل قسمت appSettings، تنظیمات پروایدر ذخیره سازی اطلاعات آن‌را وارد نمائید. مثلا در اینجا از فایل‌های XML برای ذخیره سازی اطلاعات استفاده خواهد شد (که امن‌ترین حالت ممکن است؛ از این لحاظ که اگر بانک اطلاعاتی را انتخاب کنید، ممکن است مشکل اصلی از همانجا ناشی شده باشد. بنابراین خطایی ثبت نخواهد شد. همچنین در این حالت نیازی به سایر DLLهای همراه ELMAH هم نیست). در اینجا مسیر ذخیره سازی اطلاعات در پوشه app_data/errorslog تنظیم شده است:

```
<elmah>
    <security allowRemoteAccess="1"/>
    <errorLog type="Elmah.XmlFileErrorLog, Elmah" logPath="~/App_Data/ErrorsLog"/>
  </elmah>
```

در ادامه در قسمت system.web، دو تعریف زیر را اضافه نمائید. به این ترتیب امکان دسترسی به آدرس http://server/elmah.axd مهیا می‌گردد:

```
<httpModules>
      <add name="ErrorLog" type="Elmah.ErrorLogModule, Elmah"/>
</httpModules>
<httpHandlers>
      <add verb="POST,GET,HEAD" path="elmah.axd" type="Elmah.ErrorLogPageFactory, Elmah"/>
</httpHandlers>
```

البته برای IIS7 تنظیمات ذیل نیز باید اضافه شوند:

```
<system.webServer>
    <validation validateIntegratedModeConfiguration="false"/>
    <modules runAllManagedModulesForAllRequests="true">
      <add name="ErrorLog" type="Elmah.ErrorLogModule, Elmah"/>
    </modules>
    <handlers>
      <add name="Elmah" verb="POST,GET,HEAD" path="elmah.axd" type="Elmah.ErrorLogPageFactory, Elmah"/>
    </handlers>
</system.webServer>
```

و به این ترتیب تنظیمات اولیه ELMAH به پایان می‌رسد (و با ASP.NET Web forms هیچ تفاوتی ندارد).
مرحله بعد، تنظیمات مسیریابی ASP.NET MVC است برای اینکه آدرس http://server/elmah.axd را وارد سیستم پردازشی خود نکند. البته اینکار پیشتر انجام شده است:

```
public static void RegisterRoutes(RouteCollection routes)
{
    //routes.IgnoreRoute("elmah.axd");
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

بنابراین همین تنظیمات، به همراه قالب پیش فرض یک پروژه جدید ASP.NET MVC برای استفاده از ELMAH کفایت می‌کند. اکنون پروژه جاری را یکبار دیگر خارج از VS.NET اجرا کرده و سپس به مسیر http://localhost/elmah.axd جهت مشاهده خطاهای لاگ شده به همراه جزئیات کامل آن‌ها مراجعه کنید.

مشکل: استثناهای برنامه توسط ELMAH لاگ نمی‌شوند!

فیلتر HandleError با ELMAH سازگار نیست. زیرا با استفاده از آن، متدهای کنترلرها به صورت خودکار داخل یک try/catch اجرا شده و به این ترتیب استثناهای رخ داده، مدیریت گردیده و به ELMAH هدایت نمی‌شوند. بنابراین نیاز است به متد RegisterGlobalFilters فایل Global.asax.cs مراجعه کرده و سطر زیر را حذف کنید:

```
filters.Add(new HandleErrorAttribute());
```

و یا اگر قصد نداشتید اینکار را انجام دهید، می‌توان به نحو زیر نیز مشکل را حل کرد:

```
using System.Web.Mvc;
using Elmah;

namespace MvcApplication13.CustomFilters
{
    public class ElmahHandledErrorLoggerFilter : IExceptionFilter
    {
        public void OnException(ExceptionContext context)
        {
            if (context.ExceptionHandled)
                ErrorSignal.FromCurrentContext().Raise(context.Exception);
            // all other exceptions will be caught by ELMAH anyway
        }
    }
}
```

در اینجا یک فیلتر سفارشی به برنامه اضافه شده است تا خطاهای مدیریت شده برنامه (خطاهای مدیریت شده توسط فیلتر

HandleError توکار) را به موتور ELMAH هدایت کند. سایر خطاهای مدیریت نشده به صورت خودکار توسط ELMAH ثبت خواهند شد و نیازی به انجام کار اضافی در این مورد نیست.
سپس این فیلتر جدید را به صورت سراسری تعریف کنید:

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
      filters.Add(new ElmahHandledErrorLoggerFilter());
      filters.Add(new HandleErrorAttribute());
}
```

ترتیب این‌ها هم مهم است. ابتدا باید ElmahHandledErrorLoggerFilter معرفی شود.

تذکر مهم!

حین استفاده از ELMAH یک نکته را فراموش نکنید:
اگر allowRemoteAccess آن‌را به عدد 1 تنظیم کرده‌اید، به هیچ عنوان از نام پیش فرض elmah.axd استفاده نکنید (هر نام اختیاری دیگری را که علاقمند بودید و به سادگی قابل حدس زدن نبود، در فایل web.config وارد کنید).

خلاصه بحث

-1 در ASP.NET MVC نیازی نیست تا متدهای کنترلرها را با try/catch شلوغ کنید.
-2 حتما قسمت customErrors فایل وب کانفیگ برنامه را دهی کنید (این مورد را به چک لیست اجباری تهیه یک برنامه ASP.NET MVC اضافه کنید).
-3 استفاده از فیلتر HandleError اختیاری است. اگر از قابلیت فیلتر کردن استثناهای ویژه آن استفاده نمی‌کنید، مقدار دهی customErrors وب کانفیگ برنامه هم همان کار را انجام می‌دهد.
-4 برای ثبت جزئیات دقیق استثناهای رخ داده در برنامه، از ELMAH استفاده کنید و بی‌جهت وقت خودتان را صرف بازنویسی این افزونه ارزشمند نکنید.

مطالب مشابه

معرفی ELMAH
ثبت استثناهای مدیریت شده توسط ELMAH

عنوان: ASP.NET MVC #17
نویسنده: وحید نصیری
تاریخ: ۰۸:۲۵:۰۰ ۱۳۹۱/۰۱/۲۹
آدرس: www.dotnettips.info
برچسب‌ها: MVC

فیلترهای امنیتی ASP.NET MVC

ASP.NET MVC به همراه تعدادی فیلتر امنیتی توکار است که در این قسمت به بررسی آن‌ها خواهیم پرداخت.

بررسی اعتبار درخواست (Request Validation) در ASP.NET MVC

ASP.NET MVC امکان ارسال اطلاعاتی را که دارای تگ‌های HTML باشند، نمی‌دهد. این قابلیت به صورت پیش فرض فعال است و جلوی ارسال انواع و اقسام اطلاعاتی که ممکن است سبب بروز حملات XSS Cross site scripting attacks شود را می‌گیرد. نمونه‌ای از خطای نمایش داده:

```
A potentially dangerous Request.Form value was detected from the client (Html="<a>").
```

بنابراین تصمیم گرفته شده صحیح است؛ اما ممکن است در قسمتی از سایت نیاز باشد تا کاربران از یک ویرایشگر متنی پیشرفته استفاده کنند. خروجی این نوع ویرایشگرها هم HTML است. در این حالت می‌توان صرفا برای متدی خاص امکانات Request Validation را به کمک ویژگی ValidateInput غیرفعال کرد:

```
[HttpPost]
[ValidateInput(false)]
public ActionResult CreateBlogPost(BlogPost post)
```

از ASP.NET MVC 3.0 به بعد راه حل بهتری به کمک ویژگی AllowHtml معرفی شده است. غیرفعال کردن ValidateInput ایی که معرفی شد، بر روی تمام خواص شیء BlogPost اعمال می‌شود. اما اگر فقط بخواهیم که مثلا خاصیت Text آن از مکانیزم بررسی اعتبار درخواست خارج شود، بهتر است دیگر از ویژگی ValidateInput استفاده نشده و به نحو زیر عمل گردد:

```
using System;
using System.Web.Mvc;

namespace MvcApplication14.Models
{
    public class BlogPost
    {
        public int Id { set; get; }
        public DateTime AddDate { set; get; }
        public string Title { set; get; }

        [AllowHtml]
        public string Text { set; get; }
    }
}
```

در اینجا فقط خاصیت Text مجاز به دریافت محتوای HTML ایی خواهد بود. اما خاصیت Title چنین مجوزی را ندارد. همچنین دیگر

نیازی به استفاده از ویژگی ValidateInput غیرفعال شده نیز نخواهد بود.
به علاوه همانطور که در قسمت‌های قبل نیز ذکر شد، خروجی Razor به صورت پیش فرض Html encoded است مگر اینکه صریحا آن‌را تبدیل به HTML کنیم (مثلا استفاده از متد Html.Raw). به عبارتی خروجی Razor در حالت پیش فرض در مقابل حملات XSS مقاوم است مگر اینکه آگاهانه بخواهیم آن‌را غیرفعال کنیم.

مطلب تکمیلی
مقابله با XSS ؛ یکبار برای همیشه!

فیلتر RequireHttps

به کمک ویژگی یا فیلتر RequireHttps، تمام درخواست‌های رسیده به یک متد خاص باید از طریق HTTPS انجام شوند و حتی اگر شخصی سعی به استفاده از پروتکل HTTP معمولی کند، به صورت خودکار به HTTPS هدایت خواهد شد:

```
[RequireHttps]
public ActionResult LogOn()
{
}
```

فیلتر ValidateAntiForgeryToken

نوع دیگری از حملات که باید در برنامه‌های وب به آن‌ها دقت داشت به نام CSRF یا Cross site request forgery معروف هستند.
برای مثال فرض کنید کاربری قبل از اینکه بتواند در سایت شما کار خاصی را انجام دهد، نیاز به اعتبار سنجی داشته باشد. پس از لاگین شخص و ایجاد کوکی و سشن معتبر، همین شخص به سایت دیگری مراجعه می‌کند که در آن مهاجمی بر اساس وضعیت جاری اعتبار سنجی او مثلا لینک حذف کاربری یا افزودن اطلاعات جدیدی را به برنامه ارائه می‌دهد. چون سشن شخص و کوکی مرتبط به سایت اول هنوز معتبر هستند و شخص سایت را نبسته است، «احتمال» اجرا شدن درخواست مهاجم بالا است (خصوصا اگر از مرورگرهای قدیمی استفاده کند).
بنابراین نیاز است بررسی شود آیا درخواست رسیده واقعا از طریق فرم‌های برنامه ما صادر شده است یا اینکه شخصی از طریق سایت دیگری اقدام به جعل درخواست‌ها کرده است.
برای مقابله با این نوع خطاها ابتدا باید داخل فرم‌های برنامه از متد Html.AntiForgeryToken استفاده کرد. کار این متد ایجاد یک فیلد مخفی با مقداری منحصربفرد بر اساس اطلاعات سشن جاری کاربر است، به علاوه ارسال یک کوکی خودکار تا بتوان از تطابق اطلاعات اطمینان حاصل کرد:

```
@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
```

در مرحله بعد باید فیلتر ValidateAntiForgeryToken جهت بررسی مقدار token دریافتی به متد ثبت اطلاعات اضافه شود:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult CreateBlogPost(BlogPost post)
```

در اینجا مقدار دریافتی از فیلد مخفی فرم :

```
<input name="__RequestVerificationToken" type="hidden" value="C0iPfy/3T....=" />
```

با مقدار موجود در کوکی سایت بررسی و تطابق داده خواهند شد. اگر این مقادیر تطابق نداشته باشند، یک استثنا صادر شده و از پردازش اطلاعات رسیده جلوگیری می‌شود.
علاوه بر این‌ها بهتر است حین استفاده از متد و فیلتر یاد شده، از یک salt مجزا نیز به ازای هر فرم، استفاده شود:

```
[ValidateAntiForgeryToken(Salt="1234")]
@Html.AntiForgeryToken(salt:"1234")
```

به این ترتیب tokenهای تولید شده در فرم‌های مختلف سایت یکسان نخواهند بود.
به علاوه باید دقت داشت که ValidateAntiForgeryToken فقط با فعال بودن کوکی‌ها در مرورگر کاربر کار می‌کند و اگر کاربری پذیرش کوکی‌ها را غیرفعال کرده باشد، قادر به ارسال اطلاعاتی به برنامه نخواهد بود. همچنین این فیلتر تنها در حالت HttpPost قابل استفاده است. این مورد هم در قسمت‌های قبل تاکید گردید که برای مثال بهتر است بجای داشتن لینک delete در برنامه که با HttpGet ساده کار می‌کند، آن‌را تبدیل به HttpPost نمود تا میزان امنیت برنامه بهبود یابد. از HttpGet فقط برای گزارشگیری و خواندن اطلاعات از برنامه استفاده کنید و نه ثبت اطلاعات.
بنابراین استفاده از AntiForgeryToken را به چک لیست اجباری تولید تمام فرم‌های برنامه اضافه نمائید.

مطلب مشابه
Anti CSRF module for ASP.NET

فیلتر سفارشی بررسی Referrer

یکی دیگر از روش‌های مقابله با CSRF، بررسی اطلاعات هدر درخواست ارسالی است. اگر اطلاعات Referrer آن با دومین جاری تطابق نداشت، به معنای مشکل دار بودن درخواست رسیده است. فیلتر سفارشی زیر می‌تواند نمونه‌ای باشد جهت نمایش نحوه بررسی UrlReferrer درخواست رسیده:

```
using System.Web.Mvc;

namespace MvcApplication14.CustomFilter
{
    public class CheckReferrerAttribute : AuthorizeAttribute
    {
        public override void OnAuthorization(AuthorizationContext filterContext)
        {
            if (filterContext.HttpContext != null)
            {
                if (filterContext.HttpContext.Request.UrlReferrer == null)
                    throw new System.Web.HttpException("Invalid submission");

                if (filterContext.HttpContext.Request.UrlReferrer.Host != "mysite.com")
                    throw new System.Web.HttpException("This form wasn't submitted from this site!");
            }

            base.OnAuthorization(filterContext);
        }
    }
}
```

و برای استفاده از آن:

```
[HttpPost]
[CheckReferrer]
[ValidateAntiForgeryToken]
public ActionResult DeleteTask(int id)
```

نکته‌ای امنیتی در مورد آپلود فایل‌ها در ASP.NET

هر جایی که کاربر بتواند فایلی را به سرور شما آپلود کند، مشکلات امنیتی هم از همانجا شروع خواهند شد. مثلا در متد Upload قسمت 11 این سری، منعی در آپلود انواع فایل‌ها نیست و کاربر می‌تواند انواع و اقسام شل‌ها را جهت تحت کنترل گرفتن سایت و سرور آپلود و اجرا کند. راه حل چیست؟
از همان روش امنیتی مورد استفاده توسط تیم ASP.NET MVC استفاده می‌کنیم. فایل web.config قرار گرفته در پوشه Views را باز کنید (نه فایل وب کانفیگ ریشه اصلی سایت‌را). چنین تنظیمی را می‌توان مشاهده کرد:
برای IIS6 :

```
<system.web>
    <httpHandlers>
      <add path="*" verb="*" type="System.Web.HttpNotFoundHandler"/>
    </httpHandlers>
</system.web>
```

برای IIS7 :

```
<system.webServer>
    <handlers>
      <remove name="BlockViewHandler"/>
      <add name="BlockViewHandler" path="*" verb="*" preCondition="integratedMode"
type="System.Web.HttpNotFoundHandler" />
    </handlers>
  </system.webServer>
```

تنظیم فوق، موتور اجرایی ASP.NET را در این پوشه خاص از کار می‌اندازد. به عبارتی اگر شخصی مسیر یک فایل aspx یا cshtml یا هر فایل قرار گرفته در پوشه Views را مستقیما در مرورگر خود وارد کند، با پیغام HttpNotFound مواجه خواهد شد.
این روش هم با ASP.NET Web forms سازگار است و هم با ASP.NET MVC؛ چون مرتبط است به موتور اجرایی ASP.NET که هر دوی این فریم ورک‌ها برفراز آن معنا پیدا می‌کنند.
بنابراین در پوشه فایل‌های آپلودی به سرور خود یک web.config را با محتوای فوق ایجاد کنید (و فقط باید مواظب باشید که این فایل حین آپلود فایل‌های جدید، overwrite نشود. مهم!). به این ترتیب این مسیر دیگر از طریق مرورگر قابل دسترسی نخواهد بود (با هر محتوایی). سپس برای ارائه فایل‌های آپلودی به کاربران از روش زیر استفاده کنید:

```
public ActionResult Download()
{
      return File(Server.MapPath("~/Myfiles/test.txt"), "text/plain");
}
```

مزیت مهم روش ذکر شده این است که کاربران مجاز به آپلود هر نوع فایلی خواهند بود و نیازی نیست لیست سیاه تهیه کنید که مثلا فایل‌هایی با پسوندهای خاص آپلود نشوند (که در این بین ممکن است لیست سیاه شما کامل نباشد ...).

علاوه بر تمام فیلترهای امنیتی که تاکنون بررسی شدند، فیلتر دیگری نیز به نام Authorize وجود دارد که در قسمت‌های بعدی بررسی خواهد شد.

عنوان: **ASP.NET MVC #18**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۳۱ ۱۰:۴۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

اعتبار سنجی کاربران در ASP.NET MVC

دو مکانیزم اعتبارسنجی کاربران به صورت توکار در ASP.NET MVC در دسترس هستند: Forms authentication و Windows authentication.
در حالت Forms authentication، برنامه موظف به نمایش فرم لاگین به کاربرها و سپس بررسی اطلاعات وارده توسط آن‌ها است. برخلاف آن، Windows authentication حالت یکپارچه با اعتبار سنجی ویندوز است. برای مثال زمانیکه کاربری به یک دومین ویندوزی وارد می‌شود، از همان اطلاعات ورود او به شبکه داخلی، به صورت خودکار و یکپارچه جهت استفاده از برنامه کمک گرفته خواهد شد و بیشترین کاربرد آن در برنامه‌های نوشته شده برای اینترانت‌های داخلی شرکت‌ها است. به این ترتیب کاربران یک بار به دومین وارد شده و سپس برای استفاده از برنامه‌های مختلف ASP.NET، نیازی به ارائه نام کاربری و کلمه عبور نخواهند داشت. Forms authentication بیشتر برای برنامه‌هایی که از طریق اینترنت به صورت عمومی و از طریق انواع و اقسام سیستم عامل‌ها قابل دسترسی هستند، توصیه می‌شود (و البته منعی هم برای استفاده در حالت اینترانت ندارد).
ضمنا باید به معنای این دو کلمه هم دقت داشت: هدف از Authentication این است که مشخص گردد هم اکنون چه کاربری به سایت وارد شده است. Authorization، سطح دسترسی کاربر وارد شده به سیستم و اعمالی را که مجاز است انجام دهد، مشخص می‌کند.

فیلتر Authorize در ASP.NET MVC

یکی دیگر از فیلترهای امنیتی ASP.NET MVC به نام Authorize، کار محدود ساختن دسترسی به متدهای کنترلرها را انجام می‌دهد. زمانیکه اکشن متدی به این فیلتر یا ویژگی مزین می‌شود، به این معنا است که کاربران اعتبارسنجی نشده، امکان دسترسی به آن‌را نخواهند داشت. فیلتر Authorize همواره قبل از تمامی فیلترهای تعریف شده دیگر اجرا می‌شود.
فیلتر Authorize با پیاده سازی اینترفیس System.Web.Mvc.IAuthorizationFilter توسط کلاس System.Web.Mvc.AuthorizeAttribute در دسترس می‌باشد. این کلاس علاوه بر پیاده سازی اینترفیس یاد شده، دارای دو خاصیت مهم زیر نیز می‌باشد:

```
public string Roles { get; set; } // comma-separated list of role names
public string Users { get; set; } // comma-separated list of usernames
```

زمانیکه فیلتر Authorize به تنهایی بکارگرفته می‌شود، هر کاربر اعتبار سنجی شده‌ای در سیستم قادر خواهد بود به اکشن متد مورد نظر دسترسی پیدا کند. اما اگر همانند مثال زیر، از خواص Roles و یا Users نیز استفاده گردد، تنها کاربران اعتبار سنجی شده مشخصی قادر به دسترسی به یک کنترلر یا متدی در آن خواهند شد:

```
[Authorize(Roles="Admins")]
public class AdminController : Controller
{
    [Authorize(Users="Vahid")]
    public ActionResult DoSomethingSecure()
    {
    }
}
```

در این مثال، تنها کاربرانی با نقش Admins قادر به دسترسی به کنترلر جاری Admin خواهند بود. همچنین در بین این کاربران ویژه، تنها کاربری به نام Vahid قادر است متد DoSomethingSecure را فراخوانی و اجرا کند.

اکنون سؤال اینجا است که فیلتر Authorize چگونه از دو مکانیزم اعتبار سنجی یاد شده استفاده می‌کند؟ برای پاسخ به این سؤال، فایل web.config برنامه را باز نموده و به قسمت authentication آن دقت کنید:

```
<authentication mode="Forms">
      <forms loginUrl="~/Account/LogOn" timeout="2880" />
</authentication>
```

به صورت پیش فرض، برنامه‌های ایجاد شده توسط VS.NET جهت استفاده از حالت Forms یا همان Forms authentication تنظیم شده‌اند. در اینجا کلیه کاربران اعتبار سنجی نشده، به کنترلری به نام Account و متد LogOn در آن هدایت می‌شوند.
برای تغییر آن به حالت اعتبار سنجی یکپارچه با ویندوز، فقط کافی است مقدار mode را به Windows تغییر داد و تنظیمات forms آن‌را نیز حذف کرد.

یک نکته: اعمال تنظیمات اعتبار سنجی اجباری به تمام صفحات سایت
تنظیم زیر نیز در فایل وب کانفیگ برنامه، همان کار افزودن ویژگی Authorize را انجام می‌دهد با این تفاوت که تمام صفحات سایت را به صورت خودکار تحت پوشش قرار خواهد داد (البته منهای loginUrl ایی که در تنظیمات فوق مشاهده نمودید):

```
<authorization>
     <deny users="?" />
</authorization>
```

در این حالت دسترسی به تمام آدرس‌های سایت تحت تاثیر قرار می‌گیرند، منجمله دسترسی به تصاویر و فایل‌های CSS و غیره. برای اینکه این موارد را برای مثال در حین نمایش صفحه لاگین نیز نمایش دهیم، باید تنظیم زیر را پیش از تگ system.web به فایل وب کانفیگ برنامه اضافه کرد:

```
<!-- we don't want to stop anyone seeing the css and images -->
<location path="Content">
    <system.web>
      <authorization>
        <allow users="*" />
      </authorization>
    </system.web>
</location>
```

در اینجا پوشه Content از سیستم اعتبارسنجی اجباری خارج می‌شود و تمام کاربران به آن دسترسی خواهند داشت.
به علاوه امکان امن ساختن تنها قسمتی از سایت نیز میسر است؛ برای مثال:

```
<location path="secure">
  <system.web>
    <authorization>
      <allow roles="Administrators" />
      <deny users="*" />
    </authorization>
  </system.web>
</location>
```

در اینجا مسیری به نام secure، نیاز به اعتبارسنجی اجباری دارد. به علاوه تنها کاربرانی در نقش Administrators به آن دسترسی خواهند داشت.

نکته: به تنظیمات انجام شده در فایل Web.Config دقت داشته باشید

همانطور که می‌شود دسترسی به یک مسیر را توسط تگ location بازگذاشت، امکان بستن آن هم فراهم است (بجای allow از deny استفاده شود). همچنین در ASP.NET MVC به سادگی می‌توان تنظیمات مسیریابی را در فایل global.asax.cs تغییر داد. برای مثال اینبار مسیر دسترسی به صفحات امن سایت، Admin خواهد بود نه Secure. در این حالت چون از فیلتر Authorize استفاده نشده و همچنین فایل web.config نیز تغییر نکرده، این صفحات بدون محافظت رها خواهند شد.

بنابراین اگر از تگ location برای امن سازی قسمتی از سایت استفاده می‌کنید، حتما باید پس از تغییرات مسیریابی، فایل web.config را هم به روز کرد تا به مسیر جدید اشاره کند.

به همین جهت در ASP.NET MVC بهتر است که صریحا از فیلتر Authorize بر روی کنترلرها (جهت اعمال به تمام متدهای آن) یا بر روی متدهای خاصی از کنترلرها استفاده کرد.

امکان تعریف AuthorizeAttribute در فایل global.asax.cs و متد RegisterGlobalFilters آن به صورت سراسری نیز وجود دارد. اما در این حالت حتی صفحه لاگین سایت هم دیگر در دسترس نخواهد بود. برای رفع این مشکل در ASP.NET MVC 4 فیلتر دیگری به نام AllowAnonymousAttribute معرفی شده است تا بتوان قسمت‌هایی از سایت را مانند صفحه لاگین، از سیستم اعتبارسنجی اجباری خارج کرد تا حداقل کاربر بتواند نام کاربری و کلمه عبور خودش را وارد نماید:

```
[System.Web.Mvc.AllowAnonymous]
public ActionResult Login()
{
    return View();
}
```

بنابراین در ASP.NET MVC 4.0، فیلتر AuthorizeAttribute را سراسری تعریف کنید. سپس در کنترلر لاگین برنامه از فیلتر AllowAnonymous استفاده نمائید.

البته نوشتن فیلتر سفارشی AllowAnonymousAttribute در ASP.NET MVC 3.0 نیز میسر است. برای مثال:

```
public class LogonAuthorize : AuthorizeAttribute {
     public override void OnAuthorization(AuthorizationContext filterContext) {
         if (!(filterContext.Controller is AccountController))
             base.OnAuthorization(filterContext);
     }
}
```

در این فیلتر سفارشی، اگر کنترلر جاری از نوع AccountController باشد، از سیستم اعتبار سنجی اجباری خارج خواهد شد. مابقی کنترلرها همانند سابق پردازش می‌شوند. به این معنا که اکنون می‌توان LogonAuthorize را به صورت یک فیلتر سراسری در فایل global.asax.cs معرفی کرد تا به تمام کنترلرها، منهای کنترلر Account اعمال شود.

مثالی جهت بررسی حالت Windows Authentication

یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. سپس یک کنترلر جدید را به نام Home نیز به آن اضافه کنید. در ادامه متد Index آن‌را با ویژگی Authorize، مزین نمائید. همچنین بر روی نام این متد کلیک راست کرده و یک View خالی را برای آن ایجاد کنید:

```
using System.Web.Mvc;
```

```
namespace MvcApplication15.Controllers
{
    public class HomeController : Controller
    {
        [Authorize]
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

محتوای View متناظر با متد Index را هم به شکل زیر تغییر دهید تا نام کاربر وارد شده به سیستم را نمایش دهد:

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

<h2>Index</h2>
Current user: @User.Identity.Name
```

به علاوه در فایل Web.config برنامه، حالت اعتبار سنجی را به ویندوز تغییر دهید:

```
<authentication mode="Windows" />
```

اکنون اگر برنامه را اجرا کنید و وب سرور آزمایشی انتخابی هم IIS Express باشد، پیغام HTTP Error 401.0 - Unauthorized نمایش داده می‌شود. علت هم اینجا است که Windows Authentication به صورت پیش فرض در این وب سرور غیرفعال است. برای فعال سازی آن به مسیر My Documents\IISExpress\config مراجعه کرده و فایل applicationhost.config را باز نمائید. تگ windowsAuthentication را یافته و ویژگی enabled آن‌را که false است به true تنظیم نمائید. اکنون اگر برنامه را مجددا اجرا کنیم، در محل نمایش User.Identity.Name، نام کاربر وارد شده به سیستم نمایش داده خواهد شد.
همانطور که مشاهده می‌کنید در اینجا همه چیز یکپارچه است و حتی نیازی نیست صفحه لاگین خاصی را به کاربر نمایش داد. همینقدر که کاربر توانسته به سیستم ویندوزی وارد شود، بر این اساس هم می‌تواند از برنامه‌های وب موجود در شبکه استفاده کند.

بررسی حالت Forms Authentication

برای کار با Forms Authentication نیاز به محلی برای ذخیره سازی اطلاعات کاربران است. اکثر مقالات را که مطالعه کنید شما را به مباحث membership مطرح شده در زمان ASP.NET 2.0 ارجاع می‌دهند. این روش در ASP.NET MVC هم کار می‌کند؛ اما الزامی به استفاده از آن نیست.

برای بررسی حالت اعتبار سنجی مبتنی بر فرم‌ها، یک برنامه خالی ASP.NET MVC جدید را آغاز کنید. یک کنترلر Home ساده را نیز به آن اضافه نمائید.
سپس نیاز است نکته «تنظیمات اعتبار سنجی اجباری تمام صفحات سایت» را به فایل وب کانفیگ برنامه اعمال نمائید تا نیازی نباشد فیلتر Authorize را در همه جا معرفی کرد. سپس نحوه معرفی پیش فرض Forms authentication تعریف شده در فایل web.config نیز نیاز به اندکی اصلاح دارد:

```
<authentication mode="Forms">
      <!--one month ticket-->
      <forms name=".403MyApp"
             cookieless="UseCookies"
             loginUrl="~/Account/LogOn"
             defaultUrl="~/Home"
             slidingExpiration="true"
             protection="All"
             path="/"
             timeout="43200"/>
</authentication>
```

در اینجا استفاده از کوکی‌ها اجباری شده است. loginUrl به کنترلر و متد لاگین برنامه اشاره می‌کند. defaultUrl مسیری است که کاربر پس از لاگین به صورت خودکار به آن هدایت خواهد شد. همچنین نکته‌ی مهم دیگری را که باید رعایت کرد، name ایی است که در این فایل config عنوان می‌کنید. اگر بر روی یک وب سرور، چندین برنامه وب ASP.Net را در حال اجرا دارید، باید برای هر کدام از این‌ها نامی جداگانه و منحصربفرد انتخاب کنید، در غیراینصورت تداخل رخ داده و گزینه مرا به خاطر بسپار شما کار نخواهد کرد.
کار slidingExpiration که در اینجا تنظیم شده است نیز به صورت زیر می‌باشد:
اگر لاگین موفقیت آمیزی ساعت 5 عصر صورت گیرد و timeout شما به عدد 10 تنظیم شده باشد، این لاگین به صورت خودکار در 5:10 منقضی خواهد شد. اما اگر در این حین در ساعت 5:05 ، کاربر، یکی از صفحات سایت شما را مرور کند، زمان منقضی شدن کوکی ذکر شده به 5:15 تنظیم خواهد شد(مفهوم تنظیم slidingExpiration). لازم به ذکر است که اگر کاربر پیش از نصف زمان منقضی شدن کوکی (مثلا در 5:04)، یکی از صفحات را مرور کند، تغییری در این زمان نهایی منقضی شدن رخ نخواهد داد.
اگر timeout ذکر نشود، زمان منقضی شدن کوکی ماندگار (persistent) مساوی زمان جاری + زمان منقضی شدن سشن کاربر که پیش فرض آن 30 دقیقه است، خواهد بود.

سپس یک مدل را به نام Account به پوشه مدل‌های برنامه با محتوای زیر اضافه نمائید:

```
using System.ComponentModel.DataAnnotations;

namespace MvcApplication15.Models
{
    public class Account
    {
        [Required(ErrorMessage = "Username is required to login.")]
        [StringLength(20)]
        public string Username { get; set; }

        [Required(ErrorMessage = "Password is required to login.")]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        public bool RememberMe { get; set; }
    }
}
```

همچنین مطابق تنظیمات اعتبار سنجی مبتنی بر فرم‌های فایل وب کانفیگ، نیاز به یک AccountController نیز هست:

```
using System.Web.Mvc;
using MvcApplication15.Models;

namespace MvcApplication15.Controllers
{
    public class AccountController : Controller
    {
        [HttpGet]
        public ActionResult LogOn()
        {
            return View();
```

```
        }

        [HttpPost]
        public ActionResult LogOn(Account loginInfo, string returnUrl)
        {
            return View();
        }
    }
}
```

در اینجا در حالت HttpGet فرم لاگین نمایش داده خواهد شد. بنابراین بر روی این متد کلیک راست کرده و گزینه Add view را انتخاب کنید. سپس در صفحه باز شده گزینه Create a strongly typed view را انتخاب کرده و مدل را هم بر روی کلاس Account قرار دهید. قالب scaffolding را هم Create انتخاب کنید. به این ترتیب فرم لاگین برنامه ساخته خواهد شد.
اگر به متد HttpPost فوق دقت کرده باشید، علاوه بر دریافت وهله‌ای از شیء Account، یک رشته را به نام returnUrl نیز تعریف کرده است. علت هم اینجا است که سیستم Forms authentication، صفحه بازگشت را به صورت خودکار به شکل یک کوئری استرینگ به انتهای Url جاری اضافه می‌کند. مثلا:

```
http://localhost/Account/LogOn?ReturnUrl=something
```

بنابراین اگر یکی از پارامترهای متد تعریف شده به نام returnUrl باشد، به صورت خودکار مقدار دهی خواهد شد.

تا اینجا زمانیکه برنامه را اجرا کنیم، ابتدا بر اساس تعاریف مسیریابی پیش فرض برنامه، آدرس کنترلر Home و متد Index آن فراخوانی می‌گردد. اما چون در وب کانفیگ برنامه authorization را فعال کرده‌ایم، برنامه به صورت خودکار به آدرس مشخص شده در loginUrl قسمت تعاریف اعتبارسنجی مبتنی بر فرم‌ها هدایت خواهد شد. یعنی آدرس کنترلر Account و متد LogOn آن درخواست می‌گردد. در این حالت صفحه لاگین نمایان خواهد شد.

مرحله بعد، اعتبار سنجی اطلاعات وارد شده کاربر است. بنابراین نیاز است کنترلر Account را به نحو زیر بازنویسی کرد:

```
using System.Web.Mvc;
using System.Web.Security;
using MvcApplication15.Models;

namespace MvcApplication15.Controllers
{
    public class AccountController : Controller
    {
        [HttpGet]
        public ActionResult LogOn(string returnUrl)
        {
            if (User.Identity.IsAuthenticated) //remember me
            {
                if (shouldRedirect(returnUrl))
                {
                    return Redirect(returnUrl);
                }
                return Redirect(FormsAuthentication.DefaultUrl);
            }

            return View(); // show the login page
        }

        [HttpGet]
        public void LogOut()
        {
            FormsAuthentication.SignOut();
        }

        private bool shouldRedirect(string returnUrl)
        {
```

```
            // it's a security check
            return !string.IsNullOrWhiteSpace(returnUrl) &&
                                Url.IsLocalUrl(returnUrl) &&
                                returnUrl.Length > 1 &&
                                returnUrl.StartsWith("/") &&
                                !returnUrl.StartsWith("//") &&
                                !returnUrl.StartsWith("/\\");
        }

        [HttpPost]
        public ActionResult LogOn(Account loginInfo, string returnUrl)
        {
            if (this.ModelState.IsValid)
            {
                if (loginInfo.Username == "Vahid" && loginInfo.Password == "123")
                {
                    FormsAuthentication.SetAuthCookie(loginInfo.Username, loginInfo.RememberMe);
                    if (shouldRedirect(returnUrl))
                    {
                        return Redirect(returnUrl);
                    }
                    FormsAuthentication.RedirectFromLoginPage(loginInfo.Username,
loginInfo.RememberMe);
                }
            }
            this.ModelState.AddModelError("", "The user name or password provided is incorrect.");
            ViewBag.Error = "Login faild! Make sure you have entered the right user name and
password!";
            return View(loginInfo);
        }
    }
}
```

در اینجا با توجه به گزینه «مرا به خاطر بسپار»، اگر کاربری پیشتر لاگین کرده و کوکی خودکار حاصل از اعتبار سنجی مبتنی بر فرم‌های او نیز معتبر باشد، مقدار User.Identity.IsAuthenticated مساوی true خواهد بود. بنابراین نیاز است در متد LogOn از نوع HttpGet به این مساله دقت داشت و کاربر اعتبار سنجی شده را به صفحه پیش‌فرض تعیین شده در فایل web.config برنامه یا returnUrl هدایت کرد.

در متد LogOn از نوع HttpPost، کار اعتبارسنجی اطلاعات ارسالی به سرور انجام می‌شود. در اینجا فرصت خواهد بود تا اطلاعات دریافتی، با بانک اطلاعاتی مقایسه شوند. اگر اطلاعات مطابقت داشتند، ابتدا کوکی خودکار FormsAuthentication تنظیم شده و سپس به کمک متد RedirectFromLoginPage کاربر را به صفحه پیش فرض سیستم هدایت می‌کنیم. یا اگر returnUrl ایی وجود داشت، آن‌را پردازش خواهیم کرد.

برای پیاده سازی خروج از سیستم هم تنها کافی است متد FormsAuthentication.SignOut فراخوانی شود تا تمام اطلاعات سشن و کوکی‌های مرتبط، به صورت خودکار حذف گردند.

تا اینجا فیلتر Authorize بدون پارامتر و همچنین در حالت مشخص سازی صریح کاربران به نحو زیر را پوشش دادیم:

```
[Authorize(Users="Vahid")]
```

اما هنوز حالت استفاده از Roles در فیلتر Authorize باقی مانده است. برای فعال سازی خودکار بررسی نقش‌های کاربران نیاز است یک Role provider سفارشی را با پیاده سازی کلاس RoleProvider، طراحی کنیم. برای مثال:

```
using System;
using System.Web.Security;

namespace MvcApplication15.Helper
{
    public class CustomRoleProvider : RoleProvider
    {
        public override bool IsUserInRole(string username, string roleName)
```

```
        {
            if (username.ToLowerInvariant() == "ali" && roleName.ToLowerInvariant() == "User")
                return true;
            // blabla ...
            return false;
        }

        public override string[] GetRolesForUser(string username)
        {
            if (username.ToLowerInvariant() == "ali")
            {
                return new[] { "User", "Helpdesk" };
            }

            if(username.ToLowerInvariant()=="vahid")
            {
                return new [] { "Admin" };
            }

            return new string[] { };
        }

        public override void AddUsersToRoles(string[] usernames, string[] roleNames)
        {
            throw new NotImplementedException();
        }

        public override string ApplicationName
        {
            get
            {
                throw new NotImplementedException();
            }
            set
            {
                throw new NotImplementedException();
            }
        }

        public override void CreateRole(string roleName)
        {
            throw new NotImplementedException();
        }

        public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)
        {
            throw new NotImplementedException();
        }

        public override string[] FindUsersInRole(string roleName, string usernameToMatch)
        {
            throw new NotImplementedException();
        }

        public override string[] GetAllRoles()
        {
            throw new NotImplementedException();
        }

        public override string[] GetUsersInRole(string roleName)
        {
            throw new NotImplementedException();
        }

        public override void RemoveUsersFromRoles(string[] usernames, string[] roleNames)
        {
            throw new NotImplementedException();
        }

        public override bool RoleExists(string roleName)
        {
            throw new NotImplementedException();
        }
    }
}
```

در اینجا حداقل دو متد IsUserInRole و GetRolesForUser باید پیاده سازی شوند و مابقی اختیاری هستند.

بدیهی است در یک برنامه واقعی این اطلاعات باید از یک بانک اطلاعاتی خوانده شوند؛ برای نمونه به ازای هر کاربر تعدادی نقش وجود دارد. به ازای هر نقش نیز تعدادی کاربر تعریف شده است (یک رابطه many-to-many باید تعریف شود).
در مرحله بعد باید این Role provider سفارشی را در فایل وب کانفیگ برنامه در قسمت system.web آن تعریف و ثبت کنیم:

```
<roleManager>
      <providers>
        <clear />
        <add name="CustomRoleProvider" type="MvcApplication15.Helper.CustomRoleProvider"/>
      </providers>
    </roleManager>
```

همین مقدار برای راه اندازی بررسی نقش‌ها در ASP.NET MVC کفایت می‌کند. اکنون امکان تعریف نقش‌ها، حین بکارگیری فیلتر Authorize میسر است:

```
[Authorize(Roles = "Admin")]
public class HomeController : Controller
```

عنوان: **ASP.NET MVC #19**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۱ ۱۴:۱۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

مروری بر امکانات Caching اطلاعات در ASP.NET MVC

در برنامه‌های وب، بالاترین حد کارآیی برنامه‌ها از طریق بهینه سازی الگوریتم‌ها حاصل نمی‌شود، بلکه با بکارگیری امکانات Caching سبب خواهیم شد تا اصلا کدی اجرا نشود. در ASP.NET MVC این هدف از طریق بکارگیری فیلتری به نام OutputCache میسر می‌گردد:

```
using System.Web.Mvc;

namespace MvcApplication16.Controllers
{
    public class HomeController : Controller
    {
        [OutputCache(Duration = 60, VaryByParam = "none")]
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

همانطور که ملاحظه می‌کنید، OutputCache را به یک اکشن متد یا حتی به یک کنترلر نیز می‌توان اعمال کرد. به این ترتیب HTML نهایی حاصل از View متناظر با اکشن متد جاری فراخوانی شده، Cache خواهد شد. سپس زمانیکه درخواست بعدی به سرور ارسال می‌شود، نتیجه دریافت شده، همان اطلاعات Cache شده قبلی است و عملا در سمت سرور کدی اجرا نخواهد شد. در اینجا توسط پارامتر Duration، مدت زمان معتبر بودن کش حاصل، برحسب ثانیه مشخص می‌شود. VaryByParam مشخص می‌کند که اگر متدی پارامتری را دریافت می‌کند، آیا باید به ازای هر مقدار دریافتی، مقادیر کش شده متفاوتی ذخیره شوند یا خیر. در اینجا چون متد Index پارامتری ندارد، از مقدار none استفاده شده است.

مثال یک

یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. سپس کنترلر جدید Home را نیز به آن اضافه نمائید:

```
using System;
using System.Web.Mvc;

namespace MvcApplication16.Controllers
{
    public class HomeController : Controller
    {
        [OutputCache(Duration = 60, VaryByParam = "none")]
        public ActionResult Index()
        {
            ViewBag.ControllerTime = DateTime.Now;
            return View();
        }
    }
}
```

همچنین کدهای View متد Index را نیز به نحو زیر تغییر دهید:

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

<h2>Index</h2>
<p>@ViewBag.ControllerTime</p>
<p>@DateTime.Now</p>
```

در اینجا نمایش دو زمان دریافتی از کنترلر و زمان محاسبه شده در View را مشاهده می‌کنید. هدف این است که بررسی کنیم آیا فیلتر OutputCache بر روی این دو مقدار تاثیری دارد یا خیر.
برنامه را اجرا نمائید. سپس چند بار صفحه را Refresh کنید. مشاهده خواهید کرد که هر دو زمان یاد شده تا 60 ثانیه، تغییری نخواهند کرد و حاصل نهایی از Cache خواهنده می‌شود.
کاربرد یک چنین حالتی برای مثال نمایش اطلاعات بازدیدهای یک سایت است. نباید به ازای هر کاربر وارد شده به سایت، یکبار به بانک اطلاعاتی مراجعه کرد و آمار جدیدی را تهیه نمود. یا برای نمونه اگر جایی قرار است اطلاعات وضعیت آب و هوا نمایش داده شود، بهتر است این اطلاعات، مثلا هر نیم ساعت یکبار به روز شود و نه به ازای هر بازدید جدید از سایت، توسط صدها بازدید کننده همزمان. یا برای مثال کش کردن خروجی فید RSS یک بلاگ به مدت چند ساعت نیز ایده خوبی است. از این لحاظ که اگر اطلاعات بلاگ شما روزی یکبار به روز می‌شود، نیازی نیست تا به ازای هر برنامه فیدخوان، یکبار اطلاعات از بانک اطلاعاتی دریافت شده و پروسه رندر نهایی فید صورت گیرد. منوهای پویای یک سایت نیز در همین رده قرار می‌گیرند. دریافت اطلاعات منوهای پویای سایت به ازای هر درخواست رسیده کاربری جدید، کار اشتباهی است. این اطلاعات نیز باید کش شوند تا بار سرور کاهش یابد. البته تمام این‌ها زمانی میسر خواهند شد که اطلاعات سمت سرور کش شوند.

مثال دو

همان مثال قبلی را در اینجا جهت بررسی پارامتر VaryByParam به نحو زیر تغییر می‌دهیم:

```
using System;
using System.Web.Mvc;

namespace MvcApplication16.Controllers
{
    public class HomeController : Controller
    {
        [OutputCache(Duration = 60, VaryByParam = "none")]
        public ActionResult Index(string parameter)
        {
            ViewBag.Msg = parameter ?? string.Empty;
            ViewBag.ControllerTime = DateTime.Now;
            return View();
        }
    }
}
```

در اینجا یک پارامتر به متد Index اضافه شده است. مقدار آن به ViewBag.Msg انتساب داده شده و سپس در View ، در بین تگ‌های h2 نمایش داده خواهد شد. همچنین یک فرم ساده هم جهت ارسال parameter به متد Index اضافه شده است:

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

<h2>@ViewBag.Msg</h2>

<p>@ViewBag.ControllerTime</p>
<p>@DateTime.Now</p>
```

```
@using (Html.BeginForm())
{
    @Html.TextBox("parameter")
    <input type="submit" />
}
```

اکنون برنامه را اجرا کنید. در TextBox نمایش داده شده یکبار مثلا بنویسید Test1 و فرم را به سرور ارسال نمائید. سپس مقدار Test2 را وارد کرده و ارسال نمائید. در بار دوم، خروجی صفحه همانند زمانی است که مقدار Test1 ارسال شده است. علت این است که مقدار VaryByParam به none تنظیم شده است و صرفنظر از ورودی کاربر، همان اطلاعات کش شده قبلی بازگشت داده خواهد شد. برای رفع این مشکل، متد Index را به نحو زیر تغییر دهید، به طوریکه مقدار VaryByParam به نام پارامتر متد جاری اشاره کند:

```
[OutputCache(Duration = 60, VaryByParam = "parameter")]
public ActionResult Index(string parameter)
```

در ادامه مجددا برنامه را اجرا کنید. اکنون یکبار مقدار Test1 را به سرور ارسال کنید. سپس مقدار Test2 را ارسال نمائید. مجددا همین دو مرحله را با مقادیر Test1 و Test2 تکرار کنید. مشاهده خواهید کرد که اینبار اطلاعات بر اساس مقدار پارامتر ارسالی کش شده است.

تنظیمات متفاوت OutputCache

الف) VaryByParam : اگر مساوی none قرار گیرد، همواره همان مقدار کش شده قبلی نمایش داده می‌شود. اگر مقدار آن به نام پارامتر خاصی تنظیم شود، اطلاعات کش شده بر اساس مقادیر متفاوت پارامتر دریافتی، متفاوت خواهند بود. در اینجا پارامترهای متفاوت را با یک «,» می‌توان از هم جدا ساخت. اگر تعداد پارامترها زیاد است می‌توان مقدار VaryByParam را مساوی با * قرار داد. در این حالت به ازای مقادیر متفاوت دریافتی پارامترهای مختلف، اطلاعات مجزایی در کش قرار خواهد گرفت. این روش آخر آنچنان توصیه نمی‌شود چون سربار بالایی دارد و حجم بالایی از اطلاعات بر اساس پارامترهای مختلف، باید در کش قرار گیرند.
ب) Location : مکان قرارگیری اطلاعات کش شده را مشخص می‌کند. مقدار آن نیز بر اساس یک enum به نام OutputCacheLocation مشخص می‌گردد. در این حالت برای مثال می‌توان مکان‌های Client، Server و ServerAndClient را مقدار دهی نمود. مقدار Downstream به معنای کش شدن اطلاعات بر روی پروکسی سرورهای بین راه و یا مرورگرها است. پیش فرض آن Any است که ترکیبی از Server و Downstream می‌باشد.
اگر قرار است اطلاعات یکسانی به تمام کاربران نمایش داده شود، مثلا محتوای لیست یک منوی پویا، محل قرارگیری اطلاعات کش باید سمت سرور باشد. اگر نیاز است به ازای هر کاربر محتوای اطلاعات کش شده متفاوت باشد، بهتر است محل سمت کلاینت را مقدار دهی نمود.
ج) VaryByHeader : اطلاعات، بر اساس هدرهای مشخص شده، کش می‌شوند. برای مثال مرسوم است که از Accept-Language در اینجا استفاده شود تا اطلاعات مثلا فرانسوی کش شده، به کاربر آلمانی تحویل داده نشود.
د) VaryByCustom : در این حالت نام یک متد استاتیک تعریف شده در فایل global.asax.cs باید مشخص گردد. توسط این متد کلید رشته‌ای اطلاعاتی که قرار است کش شود، بازگشت داده خواهد شد.
ه) SqlDependency : در این حالت اطلاعات تا زمانیکه تغییری در جداول بانک اطلاعاتی SQL Server صورت نگیرد، کش خواهد شد.
و) Nostore : به پروکسی سرورهای بین راه و همچنین مرورگرها اطلاع می‌دهد که اطلاعات را نباید کش کنند. اگر قسمت اعتبار سنجی این سری را به خاطر داشته باشید، چنین تعریفی در قسمت Remote validation بکارگرفته شد:

```
[OutputCache(Location = OutputCacheLocation.None, NoStore = true)]
```

و یا می‌توان برای اینکار یک فیلتر سفارشی را نیز تهیه کرد:

```
using System;
using System.Web.Mvc;

namespace MvcApplication16.Helper
{
    /// <summary>
    /// Adds "Cache-Control: private, max-age=0" header,
    /// ensuring that the responses are not cached by the user's browser.
    /// </summary>
    public class NoCachingAttribute : ActionFilterAttribute
    {
        public override void OnActionExecuted(ActionExecutedContext filterContext)
        {
            base.OnActionExecuted(filterContext);
            filterContext.HttpContext.Response.CacheControl = "private";
            filterContext.HttpContext.Response.Cache.SetMaxAge(TimeSpan.FromSeconds(0));
        }
    }
}
```

کار این فیلتر اضافه کردن هدر «Cache-Control: private, max-age=0» به Response است.

استفاده از فایل Web.Config برای معرفی تنظیمات Caching

یکی دیگر از تنظیمات ویژگی OutputCache، پارامتر CacheProfile است که امکان تنظیم آن در فایل web.config نیز وجود دارد. برای نمونه تنظیمات زیر را به قسمت system.web فایل وب کانفیگ برنامه اضافه کنید:

```
<system.web>
    <caching>
      <outputCacheSettings>
        <outputCacheProfiles>
          <add name="Aggressive" location="ServerAndClient" duration="300"/>
          <add name="Mild" duration="100"  location="Server" />
        </outputCacheProfiles>
      </outputCacheSettings>
    </caching>
```

سپس مثلا برای استفاده از پروفایلی به نام Aggressive، خواهیم داشت:

```
[OutputCache(CacheProfile = "Aggressive", VaryByParam = "parameter")]
public ActionResult Index(string parameter)
```

استفاده از ویژگی به نام donut caching

تا اینجا به این نتیجه رسیدیم که OutputCache، کل خروجی یک View را بر اساس پارامترهای مختلفی که دریافت می‌کند، کش خواهد کرد. در این بین اگر بخواهیم تنها قسمت کوچکی از صفحه کش نشود چه باید کرد؟ برای حل این مشکل قابلیتی به نام cache substitution که به donut caching هم معروف است (چون آن‌را می‌توان به شکل یک donut تصور کرد!) در ASP.NET MVC

قابل استفاده است.

```
@{ Response.WriteSubstitution(ctx => DateTime.Now.ToShortTimeString()); }
```

همانطور که ملاحظه می‌کنید برای تعریف یک چنین اطلاعاتی باید از متد Response.WriteSubstitution در یک view استفاده کرد. در این مثال، نمایش زمان جاری معرفی شده، صرف نظر از وضعیت کش صفحه جاری، کش نخواهد شد.

عکس آن هم ممکن است. فرض کنید که صفحه جاری شما از سه partial view تشکیل شده است. هر کدام از این partial viewها نیز مزین به OutpuCache هستند. اما صفحه اصلی درج کننده اطلاعات این سه partial view فاقد ویژگی Output کش است. در این حالت تنها اطلاعات این partial viewها کش خواهند شد و سایر قسمت‌های صفحه با هر بار درخواست از سرور، مجددا بر اساس اطلاعات جدید به روز خواهند شد. حالت توصیه شده نیز همین مورد است و متد Response.WriteSubstitution را صرفا جهت اطلاعات عمومی درنظر داشته باشید.

استفاده از امکانات Data Caching به صورت مستقیم

مطالبی که تا اینجا عنوان شدند به کش کردن اطلاعات Response اختصاص داشتند. اما امکانات Caching موجود، به این مورد خلاصه نشده و می‌توان اطلاعات و اشیاء را نیز کش کرد. برای مثال اطلاعات «با سطح دسترسی عمومی» دریافتی از بانک اطلاعاتی توسط یک کوئری را نیز می‌توان کش کرد. جهت انجام اینکار می‌توان از متدهای HttpRuntime.Cache.Insert و یا HttpContext.Cache.Insert استفاده کرد. استفاده از HttpContext.Cache.Insert حین نوشتن Unit tests دردسر کمتری دارد و mocking آن ساده است؛ از این جهت که بر اساس HttpContextBase تعریف شده‌است.
در ادامه یک کلاس کمکی نوشتن اطلاعات در cache و سپس بازیابی آن‌را ملاحظه می‌کنید:

```
using System;
using System.Web;
using System.Web.Caching;

namespace MvcApplication16.Helper
{
    public static class CacheManager
    {
        public static void CacheInsert(this HttpContextBase httpContext, string key, object data, int
durationMinutes)
        {
            if (data == null) return;
            httpContext.Cache.Add(
                key,
                data,
                null,
                DateTime.Now.AddMinutes(durationMinutes),
                TimeSpan.Zero,
                CacheItemPriority.AboveNormal,
                null);
        }

        public static T CacheRead<T>(this HttpContextBase httpContext, string key)
        {
            var data = httpContext.Cache[key];
            if (data != null)
                return (T)data;
            return default(T);
        }

        public static void InvalidateCache(this HttpContextBase httpContext, string key)
        {
            httpContext.Cache.Remove(key);
        }
    }
}
```

و برای استفاده از آن در یک اکشن متد، ابتدا نیاز است فضای نام این کلاس تعریف شود و سپس برای نمونه متد HttpContext.CacheInsert در دسترس خواهد بود. HttpContext یکی از خواص تعریف شده در شیء کنترلر است که با ارث بری کنترلرها از آن، همواره در دسترس می‌باشد.

در اینجا برای نمونه اطلاعات یک لیست جنریک دریافتی از بانک اطلاعاتی را مثلا 10 دقیقه (بسته به پارامتر durationMinutes آن) می‌توان کش کرد و سپس توسط متد CacheRead آن‌را دریافت نمود. اگر متد CacheRead نال برگرداند به معنای خالی بودن کش است. بنابراین یکبار اطلاعات را از بانک اطلاعاتی دریافت نموده و سپس آن‌را کش خواهیم کردیم.

البته هستند ORMهایی که یک چنین کارهایی را به صورت توکار پشتیبانی کنند. به مکانیزم آن، Second level cache هم گفته می‌شود؛ به علاوه امکان استفاده از پروایدرهای دیگری را بجز کش IIS برای ذخیره سازی موقتی اطلاعات نیز فراهم می‌کنند.

همچنین باید دقت داشت این اعداد مدت زمان، هیچگونه ضمانتی ندارند. اگر IIS احساس کند که با کمبود منابع مواجه شده است، به سادگی شروع به حذف اطلاعات موجود در کش خواهد کرد.

نکته امنیتی مهم!

به هیچ عنوان از OutputCache در صفحاتی که نیاز به اعتبار سنجی دارند، استفاده نکنید و به همین جهت در قسمت کش کردن اطلاعات، بر روی «اطلاعاتی با سطح دسترسی عمومی» تاکید شد.

فرض کنید کارمندی به صفحه مشاهده فیش حقوقی خودش مراجعه کرده است. این ماه هم اضافه حقوق آنچنانی داشته است. شما هم این صفحه را به مدت سه ساعت کش کرده‌اید. آیا می‌توانید تصور کنید اگر همین گزارش کش شده با این اطلاعات، به سایر کارمندان نمایش داده شود چه قشقرقی به پا خواهد شد؟!

بنابراین هیچگاه اطلاعات مخصوص به یک کاربر اعتبار سنجی شده را کش نکنید و «تنها» اطلاعاتی نیاز به کش شدن دارند که عمومی باشند. برای مثال لیست آخرین اخبار سایت؛ لیست آخرین مدخل‌های فید RSS سایت؛ لیست اطلاعات منوی عمومی سایت؛ لیست تعداد کاربران مراجعه کننده به سایت در طول یک روز؛ گزارش آب و هوا و کلیه اطلاعاتی با سطح دسترسی عمومی که کش شدن آن‌ها مشکل ساز نباشد.

به صورت خلاصه هیچگاه در کدهای شما چنین تعریفی نباید مشاهده شود:

```
[Authorize]
[OutputCache(Duration = 60)]
public ActionResult Index()
```

عنوان: ASP.NET MVC #20
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۳ ۱۸:۰۹:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

تهیه گزارشات تحت وب به کمک WebGrid

WebGrid از ASP.NET MVC 3.0 به صورت توکار به شکل یک Html Helper در دسترس می‌باشد و هدف از آن ساده‌تر سازی تهیه گزارشات تحت وب است. البته این گرید، تنها گرید مهیای مخصوص ASP.NET MVC نیست و پروژه MVC Contrib یا شرکت Telerik نیز نمونه‌های دیگری را ارائه داده‌اند؛ اما از این جهت که این Html Helper، بدون نیاز به کتابخانه‌های جانبی در دسترس است، بررسی آن ضروری می‌باشد.

صورت مساله

لیستی از کارمندان به همراه حقوق ماهیانه آن‌ها در دست است. اکنون نیاز به گزارشی تحت وب، با مشخصات زیر می‌باشد:
-1 گزارش باید دارای صفحه بندی بوده و هر صفحه تنها 10 ردیف را نمایش دهد.
-2 سطرها باید یک در میان دارای رنگی متفاوت باشند.
-3 ستون حقوق کارمندان در پایین هر صفحه، باید دارای جمع باشد.
-4 بتوان با کلیک بر روی عنوان هر ستون، اطلاعات را بر اساس ستون انتخابی، مرتب ساخت.
-5 لینک‌های حذف یا ویرایش یک ردیف نیز در این گزارش مهیا باشد.
-6 لیست تهیه شده، دارای ستونی به نام «ردیف» نیست. این ستون را نیز به صورت خودکار اضافه کنید.
-7 لیست نهایی اطلاعات، دارای ستونی به نام مالیات نیست. فقط حقوق کارمندان ذکر شده است. ستون محاسبه شده مالیات نیز باید به صورت خودکار در این گزارش نمایش داده شود. این ستون نیز باید دارای جمع پایین هر صفحه باشد.
-8 تمام اعداد این گزارش در حین نمایش باید دارای جدا کننده سه رقمی باشند.
-9 تاریخ‌های موجود در لیست، میلادی هستند. نیاز است این تاریخ‌ها در حین نمایش شمسی شوند.
-10 انتهای هر صفحه گزارش باید بتوان برچسب «صفحه y/n» را مشاهده کرد. n در اینجا منظور تعداد کل صفحات است و y شماره صفحه جاری می‌باشد.
-11 انتهای هر صفحه گزارش باید بتوان برچسب «رکوردهای y تا x از n» را مشاهده کرد. n در اینجا منظور تعداد کل رکوردها است.
-12 نام کوچک هر کارمند، ضخیم نمایش داده شود.
-13 به ازای هر شماره کارمندی، یک تصویر در پوشه images سایت وجود دارد. برای مثال images/id.jpg. ستونی برای نمایش تصویر متناظر با هر کارمند نیز باید اضافه شود.
-14 به ازای هر کارمند، تعدادی پروژه هم وجود دارد. پروژه‌های متناظر را توسط یک گرید تو در تو نمایش دهید.

راه حل به کمک استفاده از WebGrid

ابتدا یک پروژه خالی ASP.NET MVC را آغاز کنید. سپس مدل‌های زیر را به آن اضافه نمائید (یک کارمند که می‌تواند تعداد پروژه منتسب داشته باشد):

```
using System;
using System.Collections.Generic;

namespace MvcApplication17.Models
{
    public class Employee
    {
        public int Id { set; get; }
```

```
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public DateTime AddDate { get; set; }
        public double Salary { get; set; }
        public IList<Project> Projects { get; set; }
    }
}
```

```
namespace MvcApplication17.Models
{
    public class Project
    {
        public int Id { set; get; }
        public string Name { set; get; }
    }
}
```

سپس منبع داده نمونه زیر را به پروژه اضافه کنید. به عمد از ORM خاصی استفاده نشده تا بتوانید پروژه جاری را به سادگی در یک پروژه آزمایشی جدید، تکرار کنید.

```
using System;
using System.Collections.Generic;

namespace MvcApplication17.Models
{
    public static class EmployeeDataSource
    {
        public static IList<Employee> CreateEmployees()
        {
            var list = new List<Employee>();
            var rnd = new Random();
            for (int i = 1; i <= 1000; i++)
            {
                list.Add(new Employee
                    {
                        Id = i + 1000,
                        FirstName = "fName " + i,
                        LastName = "lName " + i,
                        AddDate = DateTime.Now.AddYears(-rnd.Next(1, 10)),
                        Salary = rnd.Next(400, 3000),
                        Projects = CreateRandomProjects()
                    });
            }
            return list;
        }

        private static IList<Project> CreateRandomProjects()
        {
            var list = new List<Project>();
            var rnd = new Random();
            for (int i = 0; i < rnd.Next(1, 7); i++)
            {
                list.Add(new Project
                {
                    Id = i,
                    Name = "Project " + i
                });
            }
            return list;
        }
    }
}
```

در ادامه یک کنترلر جدید را با محتوای زیر اضافه نمائید:

```
using System.Web.Mvc;
```

```
using MvcApplication17.Models;

namespace MvcApplication17.Controllers
{
    public class HomeController : Controller
    {
        [HttpPost]
        public ActionResult Delete(int? id)
        {
            return RedirectToAction("Index");
        }

        [HttpGet]
        public ActionResult Edit(int? id)
        {
            return View();
        }

        [HttpGet]
        public ActionResult Index(string sort, string sortdir, int? page = 1)
        {
            var list = EmployeeDataSource.CreateEmployees();
            return View(list);
        }
    }
}
```

علت تعریف متد index با پارامترهای sort و غیره به URLهای خودکاری از نوع زیر بر می‌گردد:

```
http://localhost:3034/?sort=LastName&sortdir=ASC&page=3
```

همانطور که ملاحظه می‌کنید، گرید رندر شده، از یک سری کوئری استرینگ برای مشخص سازی صفحه جاری، یا جهت مرتب سازی (صعودی و نزولی بودن آن) یا فیلد پیش فرض مرتب سازی، کمک می‌گیرد.

سپس یک View خالی را نیز برای متد Index ایجاد کنید. تا اینجا تنظیمات اولیه پروژه انجام شد.
کدهای کامل View را در ادامه ملاحظه می‌کنید:

```
@using System.Globalization
@model IList<MvcApplication17.Models.Employee>

@{
    ViewBag.Title = "Index";
}

@helper WebGridPageFirstItem(WebGrid grid)
{
    @(((grid.PageIndex + 1) * grid.RowsPerPage) - (grid.RowsPerPage - 1));
}

@helper WebGridPageLastItem(WebGrid grid)
{
    if (grid.TotalRowCount < (grid.PageIndex + 1 * grid.RowsPerPage))
    {
      @grid.TotalRowCount;
    }
    else
    {
      @((grid.PageIndex + 1) * grid.RowsPerPage);
    }
}

<h2>Employees List</h2>

@{
    var grid = new WebGrid(
```

```
        source: Model,
        canPage: true,
        rowsPerPage: 10,
        canSort: true,
        defaultSort: "FirstName"
    );
    var salaryPageSum = 0;
    var taxPageSum = 0;
    var rowIndex = ((grid.PageIndex + 1) * grid.RowsPerPage) - (grid.RowsPerPage - 1);
}

<div id="container">
    @grid.GetHtml(
            tableStyle: "webgrid",
            headerStyle: "webgrid-header",
            footerStyle: "webgrid-footer",
            alternatingRowStyle: "webgrid-alternating-row",
            selectedRowStyle: "webgrid-selected-row",
            rowStyle: "webgrid-row-style",
            htmlAttributes: new { id = "MyGrid" },
            mode: WebGridPagerModes.All,
            columns: grid.Columns(
                grid.Column(header: "#",
                            style: "text-align-center-col",
                            format: @<text>@(rowIndex++)</text>),
                grid.Column(columnName: "FirstName", header: "First Name",
                            format: @<span style='font-weight: bold'>@item.FirstName</span>,
                            style: "text-align-center-col"),
                grid.Column(columnName: "LastName", header: "Last Name"),
                grid.Column(header: "Image",
                            style: "text-align-center-col",
                            format: @<text><img alt="@item.Id" src="@Url.Content("~/images/" +
@item.Id + ".jpg")" /></text>),
                grid.Column(columnName: "AddDate", header: "Start",
                            style: "text-align-center-col",
                            format: item =>
                            {
                                int ym = item.AddDate.Year;
                                int mm = item.AddDate.Month;
                                int dm = item.AddDate.Day;
                                var persianCalendar = new PersianCalendar();
                                int ys = persianCalendar.GetYear(new DateTime(ym, mm, dm, new
GregorianCalendar()));
                                int ms = persianCalendar.GetMonth(new DateTime(ym, mm, dm, new
GregorianCalendar()));
                                int ds = persianCalendar.GetDayOfMonth(new DateTime(ym, mm, dm, new
GregorianCalendar()));
                                return ys + "/" + ms.ToString("00") + "/" + ds.ToString("00");

                            }),
                grid.Column(columnName: "Salary", header: "Salary",
                            format: item =>
                            {
                                salaryPageSum += item.Salary;
                                return string.Format("${0:n0}", item.Salary);
                            },
                            style: "text-align-center-col"),
                grid.Column(header: "Tax", canSort: true,
                            format: item =>
                            {
                                var tax = item.Salary * 0.2;
                                taxPageSum += tax;
                                return string.Format("${0:n0}", tax);
                            }),
                grid.Column(header: "Projects", columnName: "Projects",
                            style: "text-align-center-col",
                            format: item =>
                            {
                                var subGrid = new WebGrid(
                                                        source: item.Projects,
                                                        canPage: false,
                                                        canSort: false
                                                        );
                                return subGrid.GetHtml(
                                                        htmlAttributes: new { id = "MySubGrid" },
                                                        tableStyle: "webgrid",
                                                        headerStyle: "webgrid-header",
                                                        footerStyle: "webgrid-footer",
                                                        alternatingRowStyle: "webgrid-alternating-row",
                                                        selectedRowStyle: "webgrid-selected-row",
                                                        rowStyle: "webgrid-row-style"
```

```
                                    );
                                }),
                grid.Column(header: "",
                            style: "text-align-center-col",
                            format: item => @Html.ActionLink(linkText: "Edit", actionName: "Edit",
                                                             controllerName: "Home", routeValues: new
{ id = item.Id },
                                                             htmlAttributes: null)),
                grid.Column(header: "",
                            format: @<form action="/Home/Delete/@item.Id" method="post"><input
type="submit"
                                          onclick="return confirm('Do you want to delete this
record?');"
                                          value="Delete"/></form>),
                grid.Column(header: "", format: item => item.GetSelectLink("Select"))
                     )
                 )

    <strong>Page:</strong> @(grid.PageIndex + 1) / @grid.PageCount,
    <strong>Records:</strong> @WebGridPageFirstItem(@grid) - @WebGridPageLastItem(@grid) of
@grid.TotalRowCount

    @*
    @if (@grid.HasSelection)
    {
        @RenderPage("~/views/path/_partial_view.cshtml", new { Employee = grid.SelectedRow })
    }
    *@
</div>

@section script{
<script type="text/javascript">
    $(function () {
        $('#MyGrid tbody:first').append(
        '<tr class="total-row"><td></td>\
         <td></td><td></td><td></td>\
         <td><strong>Total:</strong></td>\
         <td>@string.Format("${0:n0}", @salaryPageSum)</td>\
         <td>@string.Format("${0:n0}", @taxPageSum)</td>\
         <td></td><td></td><td></td></tr>');
    });
</script>
}
```

توضیحات ریز جزئیات View فوق

تعریف ابتدایی شیء WebGrid و مقدار دهی آن

در ابتدا نیاز است یک وهله از شیء WebGrid را ایجاد کنیم. در اینجا می‌توان تنظیم کرد که آیا نیاز است اطلاعات نمایش داده شده دارای صفحه بندی (canPage) خودکار باشند؟ منبع داده (source) کدام است. در صورت فعال سازی صفحه بندی خودکار، چه تعداد ردیف (rowsPerPage) در هر صفحه نمایش داده شود. آیا نیاز است بتوان با کلیک بر روی سر ستون‌ها، اطلاعات را بر اساس فیلد متناظر با آن مرتب (canSort) ساخت؟ همچنین در صورت نیاز به مرتب سازی، اولین باری که گرید نمایش داده می‌شود، بر اساس چه فیلدی (defaultSort) باید مرتب شده نمایش داده شود:

```
@{
    var grid = new WebGrid(
        source: Model,
        canPage: true,
        rowsPerPage: 10,
        canSort: true,
        defaultSort: "FirstName"
    );
    var salaryPageSum = 0;
    var taxPageSum = 0;
    var rowIndex = ((grid.PageIndex + 1) * grid.RowsPerPage) - (grid.RowsPerPage - 1);
}
```

در اینجا همچنین سه متغیر کمکی هم تعریف شده که از این‌ها برای تهیه جمع ستون‌های حقوق و مالیات و همچنین نمایش شماره ردیف جاری استفاده می‌شود. فرمول نحوه محاسبه اولین ردیف هر صفحه را هم ملاحظه می‌کنید. شماره ردیف‌های بعدی، rowIndex++ خواهند بود.

تعریف رنگ و لعاب گرید نمایش داده شده

در ادامه به کمک متد grid.GetHtml، رشته‌ای معادل اطلاعات HTML صفحه جاری، بازگشت داده می‌شود. در اینجا می‌توان یک سری خواص تکمیلی را تنظیم نمود. برای مثال:

```
tableStyle: "webgrid",
headerStyle: "webgrid-header",
footerStyle: "webgrid-footer",
alternatingRowStyle: "webgrid-alternating-row",
selectedRowStyle: "webgrid-selected-row",
rowStyle: "webgrid-row-style",
htmlAttributes: new { id = "MyGrid" },
```

هر کدام از این رشته‌ها در حین رندر نهایی گرید، تبدیل به یک class خواهند شد. برای نمونه:

```
<div id="container">
      <table class="webgrid" id="MyGrid">
        <thead>
          <tr class="webgrid-header">
```

به این ترتیب با اندکی ویرایش css سایت، می‌توان انواع و اقسام رنگ‌ها را به سطرها و ستون‌های گرید نهایی اعمال کرد. برای مثال اطلاعات زیر را به فایل css سایت اضافه نمائید:

```
/* Styles for WebGrid
---------------------------------------------------------*/
.webgrid
{
 width: 100%;
 margin: 0px;
 padding: 0px;
 border: 0px;
 border-collapse: collapse;
 font-family: Tahoma;
 font-size: 9pt;
}

.webgrid a
{
 color: #000;
}

.webgrid-header
{
 padding: 0px 5px;
 text-align: center;
 border-bottom: 2px solid #739ace;
 height: 20px;
 border-top: 2px solid #D6E8FF;
 border-left: 2px solid #D6E8FF;
 border-right: 2px solid #D6E8FF;
}

.webgrid-header th
{
 background-color: #eaf0ff;
 border-right: 1px solid #ddd;
}
```

```
.webgrid-footer
{
 padding: 6px 5px;
 text-align: center;
 background-color: #e8eef4;
 border-top: 2px solid #3966A2;
 height: 25px;
 border-bottom: 2px solid #D6E8FF;
 border-left: 2px solid #D6E8FF;
 border-right: 2px solid #D6E8FF;
}

.webgrid-alternating-row
{
 height: 22px;
 background-color: #f2f2f2;
 border-bottom: 1px solid #d2d2d2;
 border-left: 2px solid #D6E8FF;
 border-right: 2px solid #D6E8FF;
}

.webgrid-row-style
{
 height: 22px;
 border-bottom: 1px solid #d2d2d2;
 border-left: 2px solid #D6E8FF;
 border-right: 2px solid #D6E8FF;
}

.webgrid-selected-row
{
 font-weight: bold;
}

.text-align-center-col
{
 text-align: center;
}

.total-row
{
 background-color:#f9eef4;
}
```

همانطور که ملاحظه می‌کنید، رنگ‌های ردیف‌ها، هدر و فوتر گرید و غیره در اینجا تنظیم می‌شوند.
به علاوه اگر دقت کرده باشید در تعاریف گرید، htmlAttributes هم مقدار دهی شده است. در اینجا به کمک یک anonymously typed object، مقدار id گرید مشخص شده است. از این id در حین کار با jQuery استفاده خواهیم کرد.

تعیین نوع Pager

پارامتر دیگری که در متد grid.GetHtml تنظیم شده است، mode: WebGridPagerModes.All می‌باشد. WebGridPagerModes یک enum با محتوای زیر است و توسط آن می‌توان نوع Pager گرید را تعیین کرد:

```
[Flags]
public enum WebGridPagerModes
{
        Numeric = 1,
        //
        NextPrevious = 2,
        //
        FirstLast = 4,
        //
        All = 7,
}
```

نحوه تعریف ستون‌های گرید

اکنون به مهم‌ترین قسمت تهیه گزارش رسیده‌ایم. در اینجا با مقدار دهی پارامتر columns، نحوه نمایش اطلاعات ستون‌های مختلف مشخص می‌گردد. مقداری که باید در اینجا تنظیم شود، آرایه‌ای از نوع WebGridColumn می‌باشد و مرسوم است به کمک متد کمکی grid.Columns، اینکار را انجام داد.
متد کمکی grid.Column، یک وهله از شیء WebGridColumn را بر می‌گرداند و از آن برای تعریف هر ستون استفاده خواهیم کرد. توسط پارامتر columnName آن، نام فیلدی که باید اطلاعات ستون جاری از آن اخذ شود مشخص می‌شود. به کمک پارامتر header، عبارت سرستون متناظر تنظیم می‌گردد. پارامتر format، مهم‌ترین و توانمندترین پارامتر متد grid.Column است:

```
grid.Column(columnName: "FirstName", header: "First Name",
                            format: @<span style='font-weight: bold'>@item.FirstName</span>,
                            style: "text-align-center-col"),
grid.Column(columnName: "LastName", header: "Last Name"),
```

پارامتر format، به نحو زیر تعریف شده است:

```
Func<dynamic, object> format
```

به این معنا که هر بار پیش از رندر سطر جاری، زمانیکه قرار است سلولی رندر شود، یک شیء dynamic در اختیار شما قرار می‌گیرد. این شیء dynamic یک رکورد از اطلاعات Model جاری است. به این ترتیب به اطلاعات تمام سلول‌های ردیف جاری دسترسی خواهیم داشت. بر این اساس هر نوع پردازشی را که لازم بود، انجام دهید (شبیه به فرمول نویسی در ابزارهای گزارش سازی، اما اینبار با کدهای سی شارپ) و مقدار فرمت شده نهایی را به صورت یک رشته بر گردانید. این رشته نهایتا در سلول جاری درج خواهد شد.
اگر از پارامتر فرمت استفاده نشود، همان مقدار فیلد جاری بدون تغییری رندر می‌گردد.
حداقل به دو نحو می‌توان پارامتر فرمت را مقدار دهی کرد:

```
format: @<span style='font-weight: bold'>@item.FirstName</span>
or
format: item =>
                    {
                         salaryPageSum += item.Salary;
                          return string.Format("${0:n0}", item.Salary);
                     }
```

مستقیما از توانمندی‌های Razor استفاده کنید. مثلا یک تگ کامل را بدون نیاز به محصور سازی آن بین "" شروع کنید. سپس @item به وهله‌ای از رکورد در دسترس اشاره می‌کند که در اینجا وهله‌ای از شیء کارمند است.
و یا همانند روشی که برای محاسبه جمع حقوق هر صفحه مشاهده می‌کنید، مستقیما از lambda expressions برای تعریف یک anonymous delegate استفاده کنید.

نحوه اضافه کردن ستون ردیف

ستون ردیف، یک ستون محاسبه شده (calculated field) است:

```
grid.Column(header: "#",
                style: "text-align-center-col",
                format: @<text>@(rowIndex++)</text>),
```

نیازی نیست حتما یک grid.Column، به فیلدی در کلاس کارمند اشاره کند. مقدار سفارشی آن را به کمک پارامتر format تعیین خواهیم کرد. هر بار که قرار است یک ردیف رندر شود، یکبار این پارامتر فراخوانی خواهد شد. فرمول محاسبه rowIndex ابتدای صفحه را نیز پیشتر ملاحظه نمودید.

نحوه اضافه کردن ستون سفارشی تصاویر کارمندها

ستون تصویر کارمندها نیز مستقیما در کلاس کارمند تعریف نشده است. بنابراین می‌توان آن‌را با مقدار دهی صحیح پارامتر format ایجاد کرد:

```
grid.Column(header: "Image",
                  style: "text-align-center-col",
                  format: @<text><img alt="@item.Id" src="@Url.Content("~/images/" + @item.Id +
".jpg")" /></text>),
```

در این مثال، تصاویر کارمندها در پوشه images واقع در ریشه سایت، قرار دارند. به همین جهت از متد Url.Content برای مقدار دهی صحیح آن استفاده کردیم. به علاوه در اینجا @item.Id به Id رکورد در حال رندر اشاره می‌کند.

نحوه تبدیل تاریخ‌ها به تاریخ شمسی

در ادامه بازهم به کمک پارامتر format، یک وهله از شیء dynamic اشاره کننده به رکورد در حال رندر را دریافت می‌کنیم. سپس فرصت خواهیم داشت تا بر این اساس، فرمول نویسی کنیم. دست آخر هم رشته مورد نظر نهایی را بازگشت می‌دهیم:

```
grid.Column(columnName: "AddDate", header: "Start",
                  style: "text-align-center-col",
                  format: item =>
                  {
                       int ym = item.AddDate.Year;
                       int mm = item.AddDate.Month;
                       int dm = item.AddDate.Day;
                       var persianCalendar = new PersianCalendar();
                       int ys = persianCalendar.GetYear(new DateTime(ym, mm, dm, new
GregorianCalendar()));
                       int ms = persianCalendar.GetMonth(new DateTime(ym, mm, dm, new
GregorianCalendar()));
                       int ds = persianCalendar.GetDayOfMonth(new DateTime(ym, mm, dm, new
GregorianCalendar()));
                       return ys + "/" + ms.ToString("00") + "/" + ds.ToString("00");
                    }),
```

اضافه کردن ستون سفارشی مالیات

در کلاس کارمند، خاصیت حقوق وجود دارد اما مالیات خیر. با توجه به آن می‌توانیم به کمک پارامتر format، به اطلاعات شیء dynamic در حال رندر دسترسی داشته باشیم. بنابراین به اطلاعات حقوق دسترسی داریم و سپس با کمی فرمول نویسی، مقدار نهایی مورد نظر را بازگشت خواهیم داد. همچنین در اینجا می‌توان نحوه بازگشت مقدار حقوق را به صورت رشته‌ای حاوی جدا کننده‌های سه رقمی نیز مشاهده کرد:

```
grid.Column(columnName: "Salary", header: "Salary",
                format: item =>
                            {
                                salaryPageSum += item.Salary;
                                return string.Format("${0:n0}", item.Salary);
                            },
                            style: "text-align-center-col"),
grid.Column(header: "Tax", canSort: true,
```

```
                            format: item =>
                            {
                                var tax = item.Salary * 0.2;
                                taxPageSum += tax;
                                return string.Format("${0:n0}", tax);
                            }),
```

اضافه کردن گردیدهای تو در تو

متد Grid.GetHtml، یک رشته را بر می‌گرداند. بنابراین در هر چند سطح که نیاز باشد می‌توان یک گرید را بر اساس اطلاعات دردسترس رندر کرد و سپس بازگشت داد:

```
grid.Column(header: "Projects", columnName: "Projects",
                  style: "text-align-center-col",
                  format: item =>
                  {
                                var subGrid = new WebGrid(
                                                        source: item.Projects,
                                                        canPage: false,
                                                        canSort: false
                                                        );
                                return subGrid.GetHtml(
                                                        htmlAttributes: new { id = "MySubGrid" },
                                                        tableStyle: "webgrid",
                                                        headerStyle: "webgrid-header",
                                                        footerStyle: "webgrid-footer",
                                                        alternatingRowStyle: "webgrid-alternating-row",
                                                        selectedRowStyle: "webgrid-selected-row",
                                                        rowStyle: "webgrid-row-style"
                                    );
                      }),
```

در اینجا کار اصلی از طریق پارامتر format شروع می‌شود. سپس به کمک item.Projects به لیست پروژه‌های هر کارمند دسترسی خواهیم داشت. بر این اساس یک گرید جدید را تولید کرد و سپس رشته معادل با آن را به کمک متد subGrid.GetHtml دریافت و بازگشت می‌دهیم. این رشته در سلول جاری درج خواهد شد. به نوعی یک گزارش master detail یا sub report را تولید کرده‌ایم.

اضافه کردن دکمه‌های ویرایش، حذف و انتخاب

هر سه دکمه ویرایش، حذف و انتخاب در ستون‌هایی سفارشی قرار خواهند گرفت. بنابراین مقدار دهی header و format متد grid.Column کفایت می‌کند:

```
grid.Column(header: "",
                  style: "text-align-center-col",
                   format: item => @Html.ActionLink(linkText: "Edit", actionName: "Edit",
                                                                    controllerName: "Home", routeValues: new
{ id = item.Id },
                                                                    htmlAttributes: null)),
grid.Column(header: "",
                  format: @<form action="/Home/Delete/@item.Id" method="post"><input type="submit"
                                                onclick="return confirm('Do you want to delete this
record?');"
                                                value="Delete"/></form>),
grid.Column(header: "", format: item => item.GetSelectLink("Select"))
```

نکته جدیدی که در اینجا وجود دارد متد item.GetSelectLink می‌باشد. این متد جزو متدهای توکار گرید است و کار آن بازگشت دادن شیء grid.SelectedRow می‌باشد. این شیء پویا، حاوی اطلاعات رکورد انتخاب شده است. برای مثال اگر نیاز باشد این

اطلاعات به صفحه‌ای ارسال شود، می‌توان از روش زیر استفاده کرد:

```
@if (@grid.HasSelection)
    {
        @RenderPage("~/views/path/_partial_view.cshtml", new { Employee = grid.SelectedRow })
    }
```

نمایش برچسب‌های صفحه x از n و رکوردهای x تا y از z

در یک گزارش خوب باید مشخص باشد که صفحه جاری، کدامین صفحه از چه تعداد صفحه کلی است. یا رکوردهای صفحه جاری چه بازه‌ای از تعداد رکوردهای کلی را تشکیل می‌دهند. برای این منظور چند متد کمکی به نام‌های WebGridPageFirstItem و WebGridPageLastItem تهیه شده‌اند که آن‌ها را در ابتدای View ارائه شده، مشاهده نمودید:

```
<strong>Page:</strong> @(grid.PageIndex + 1) / @grid.PageCount,
    <strong>Records:</strong> @WebGridPageFirstItem(@grid) - @WebGridPageLastItem(@grid) of
@grid.TotalRowCount
```

نمایش جمع ستون‌های حقوق و مالیات در هر صفحه

گرید توکار همراه با ASP.NET MVC در این مورد راه حلی را ارائه نمی‌دهد. بنابراین باید اندکی دست به ابتکار زد. مثلا:

```
@section script{
<script type="text/javascript">
    $(function () {
        $('#MyGrid tbody:first').append(
        '<tr class="total-row"><td></td>\
         <td></td><td></td><td></td>\
         <td><strong>Total:</strong></td>\
         <td>@string.Format("${0:n0}", @salaryPageSum)</td>\
         <td>@string.Format("${0:n0}", @taxPageSum)</td>\
         <td></td><td></td><td></td></tr>');
    });
</script>
}
```

در این مثال به کمک jQuery با توجه به اینکه id گرید ما MyGrid است، یک ردیف سفارشی که همان جمع محاسبه شده است، به tbody جدول نهایی تولیدی اضافه می‌شود. از tbody:first هم در اینجا استفاده شده است تا ردیف اضافه شده به گریدهای تو در تو اعمال نشود.

سپس فایل Views\Shared_Layout.cshtml را گشوده و از section تعریف شده، برای مقدار دهی master page سایت، استفاده نمائید:

```
<head>
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
    @RenderSection("script", required: false)
</head>
```

عنوان: **ASP.NET MVC #21**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۵ ۱۷:۴۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

آشنایی با تکنیک‌های Ajax در ASP.NET MVC

اهمیت آشنایی با Ajax، ارائه تجربه کاربری بهتری از برنامه‌های وب، به مصرف کنندگان نهایی آن می‌باشد. به این ترتیب می‌توان درخواست‌های غیرهمزمانی (asynchronous) را با فرمت XML یا Json به سرور ارسال کرد و سپس نتیجه نهایی را که حجم آن نسبت به یک صفحه کامل بسیار کمتر است، به کاربر ارائه داد. غیرهمزمان بودن درخواست‌ها سبب می‌شود تا ترد اصلی رابط کاربری برنامه قفل نشده و کاربر در این بین می‌تواند به سایر امور خود بپردازد. به این ترتیب می‌توان برنامه‌های وبی را که شبیه به برنامه‌های دسکتاپ هستند تولید نمود؛ کل صفحه مرتبا به سرور ارسال نمی‌شود، flickering و چشمک زدن صفحه کاهش خواهد یافت (چون نیازی به ترسیم مجدد کل صفحه نخواهد بود و عموما قسمتی جزئی از یک صفحه به روز می‌شود) یا بدون نیاز به ارسال کل صفحه به سرور، به کاربری خواهیم گفت که آیا اطلاعاتی که وارد کرده است معتبر می‌باشد یا نه (نمونه‌ای از آن را در قسمت Remote validation اعتبار سنجی اطلاعات ملاحظه نمودید).

مروری بر محتویات پوشه Scripts یک پروژه جدید ASP.NET MVC در ویژوال استودیو

با ایجاد هر پروژه ASP.NET MVC جدیدی در ویژوال استودیو، یک سری اسکریپت هم به صورت خودکار در پوشه Scripts آن اضافه می‌شوند. تعدادی از این فایل‌ها توسط مایکروسافت پیاده سازی شده‌اند. برای مثال:

MicrosoftAjax.debug.js
MicrosoftAjax.js
MicrosoftMvcAjax.debug.js
MicrosoftMvcAjax.js
MicrosoftMvcValidation.debug.js
MicrosoftMvcValidation.js

این فایل‌ها از ASP.NET MVC 3 به بعد، صرفا جهت سازگاری با نگارش‌های قبلی قرار دارند و استفاده از آن‌ها اختیاری است. بنابراین با خیال راحت آن‌ها را delete کنید! روش توصیه شده جهت پیاده سازی ویژگی‌های Ajax ایی، استفاده از کتابخانه‌های مرتبط با jQuery می‌باشد؛ از این جهت که 100ها افزونه برای کار با آن توسط گروه وسیعی از برنامه نویس‌ها در سراسر دنیا تاکنون تهیه شده است. به علاوه فریم ورک jQuery تنها منحصر به اعمال Ajax ایی نیست و از آن جهت دستکاری DOM (document object model) و CSS صفحه نیز می‌توان استفاده کرد. همچنین حجم کمی نیز داشته، با انواع و اقسام مرورگرها سازگار است و مرتبا هم به روز می‌شود.

در این پوشه سه فایل دیگر پایه کتابخانه jQuery نیز قرار دارند:

jquery-xyz-vsdoc.js
jquery-xyz.js
jquery-xyz.min.js

فایل vsdoc برای ارائه نهایی برنامه طراحی نشده است. هدف از آن ارائه Intellisense بهتری از jQuery در VS.NET می‌باشد. فایلی که باید به کلاینت ارائه شود، فایل min یا فشرده شده آن است. اگر به آن نگاهی بیندازیم به نظر obfuscated مشاهده می‌شود. علت آن هم حذف فواصل، توضیحات و همچنین کاهش طول متغیرها است تا اندازه فایل نهایی به حداقل خود کاهش پیدا کند. البته این فایل از دیدگاه مفسر جاوا اسکریپت یک مرورگر، فایل بی‌نقصی است!
اگر علاقمند هستید که سورس اصلی jQuery را مطالعه کنید، به فایل jquery-xyz.js مراجعه نمائید.

محل الحاق اسکریپت‌های عمومی مورد نیاز برنامه نیز بهتر است در فایل master page یا layout برنامه باشد که به صورت پیش فرض اینکار انجام شده است.
سایر فایل‌های اسکریپتی که در این پوشه مشاهده می‌شوند، یک سری افزونه عمومی یا نوشته شده توسط تیم ASP.NET MVC برفراز jQuery هستند.

به چهار نکته نیز حین استفاده از اسکریپت‌های موجود باید دقت داشت:
الف) همیشه از متد Url.Content همانند تعاریفی که در فایل Views\Shared_Layout.cshtml مشاهده می‌کنید، برای مشخص سازی مسیر ریشه سایت، استفاده نمائید. به این ترتیب صرفنظر از آدرس جاری صفحه، همواره آدرس صحیح قرارگیری پوشه اسکریپت‌ها در صفحه ذکر خواهد شد.
ب) ترتیب فایل‌های js مهم هستند. ابتدا باید کتابخانه اصلی jQuery ذکر شود و سپس افزونه‌های آن‌ها.
ج) اگر اسکریپت‌های jQuery در فایل layout سایت تعریف شده‌اند؛ نیازی به تعریف مجدد آن‌ها در Viewهای سایت نیست.
د) اگر View ایی به اسکریپت ویژه‌ای جهت اجرا نیاز دارد، بهتر است آن‌را به شکل یک section داخل view تعریف کرد و سپس به کمک متد RenderSection این قسمت را در layout سایت مقدار دهی نمود. مثالی از آن‌را در قسمت 20 این سری مشاهده نمودید (افزودن نمایش جمع هر ستون گزارش).

یک نکته
اگر آخرین به روز رسانی‌های ASP.NET MVC را نیز نصب کرده باشید، فایلی به نام packages.config به صورت پیش فرض به هر پروژه جدید ASP.NET MVC اضافه می‌شود. به این ترتیب VS.NET به کمک NuGet این امکان را خواهد یافت تا شما را از آخرین به روز رسانی‌های این کتابخانه‌ها مطلع کند.

آشنایی با Ajax Helpers توکار ASP.NET MVC

اگر به تعاریف خواص و متدهای کلاس WebViewPage دقت کنیم:

```
using System;

namespace System.Web.Mvc
{
    public abstract class WebViewPage<TModel> : WebViewPage
    {
        protected WebViewPage();
        public AjaxHelper<TModel> Ajax { get; set; }
        public HtmlHelper<TModel> Html { get; set; }
        public TModel Model { get; }
        public ViewDataDictionary<TModel> ViewData { get; set; }
        public override void InitHelpers();
        protected override void SetViewData(ViewDataDictionary viewData);
    }
}
```

علاوه بر خاصیت Html که وهله‌ای از آن امکان دسترسی به Html helpers توکار ASP.NET MVC را در یک View فراهم می‌کند، خاصیتی به نام Ajax نیز وجود دارد که توسط آن می‌توان به تعدادی متد AjaxHelper توکار دسترسی داشت. برای مثال توسط متد Ajax.ActionLink می‌توان قسمتی از صفحه را به کمک ویژگی‌های Ajax، به روز رسانی کرد.

مثالی در مورد به روز رسانی قسمتی از صفحه به کمک متد Ajax.ActionLink

ابتدا نیاز است فایل Views\Shared_Layout.cshtml را اندکی ویرایش کرد. برای این منظور سطر الحاق jquery.unobtrusive-ajax.min.js را به فایل layout برنامه اضافه نمائید (اگر این سطر اضافه نشود، متد Ajax.ActionLink همانند یک لینک معمولی رفتار خواهد کرد):

```
<head>
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.min.js")"
type="text/javascript"></script>
</head>
```

سپس مدل ساده و منبع داده زیر را نیز به پروژه اضافه کنید:

```
namespace MvcApplication18.Models
{
    public class Employee
    {
        public int Id { set; get; }
        public string Name { set; get; }
    }
}
```

```
using System.Collections.Generic;

namespace MvcApplication18.Models
{
    public static class EmployeeDataSource
    {
        public static IList<Employee> CreateEmployees()
        {
            var list = new List<Employee>();
            for (int i = 0; i < 1000; i++)
            {
                list.Add(new Employee { Id = i + 1, Name = "name " + i });
            }
            return list;
        }
    }
}
```

در ادامه کنترلر جدیدی را به برنامه با محتوای زیر اضافه کنید:

```
using System.Linq;
using System.Web.Mvc;
using MvcApplication18.Models;

namespace MvcApplication18.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost] //for IE-8
        public ActionResult EmployeeInfo(int? id)
        {
            if (!Request.IsAjaxRequest())
                return View("Error");

            if (!id.HasValue)
                return View("Error");

            var list = EmployeeDataSource.CreateEmployees();
            var data = list.Where(x => x.Id == id.Value).FirstOrDefault();
            if (data == null)
```

```
                return View("Error");
            return PartialView(viewName: "_EmployeeInfo", model: data);
        }
    }
}
```

بر روی متد Index کلیک راست کرده و گزینه Add view را انتخاب کنید. یک View خالی را به آن اضافه نمائید. همچنین بر روی متد EmployeeInfo کلیک راست کرده و با انتخاب گزینه Add view در صفحه ظاهر شده یک partial view را اضافه نمائید. جهت تمایز بین partial view و view هم بهتر است نام partial view با یک underline شروع شود.
کدهای partial view مورد نظر را به نحو زیر تغییر دهید:

```
@model MvcApplication18.Models.Employee

<strong>Name:</strong> @Model.Name
```

سپس کدهای View متناظر با متد Index را نیز به صورت زیر اعمال کنید:

```
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>

<div id="EmployeeInfo">
    @Ajax.ActionLink(
     linkText: "Get Employee-1 info",
     actionName: "EmployeeInfo",
     controllerName: "Home",
     routeValues: new { id = 1 },
     ajaxOptions: new AjaxOptions
                  {
                      HttpMethod = "POST",
                      InsertionMode = InsertionMode.Replace,
                      UpdateTargetId = "EmployeeInfo",
                      LoadingElementId = "Progress"
                  })
</div>

<div id="Progress" style="display: none">
    <img src="@Url.Content("~/Content/images/loading.gif")" alt="loading..."  />
</div>
```

توضیحات جزئیات کدهای فوق

متد Ajax.ActionLink لینکی را تولید می‌کند که با کلیک کاربر بر روی آن، اطلاعات اکشن متد واقع در کنترلری مشخص، به کمک ویژگی‌های jQuery Ajax دریافت شده و سپس در مقصدی که توسط UpdateTargetId مشخص می‌گردد، بر اساس مقدار InsertionMode، درج خواهد شد (می‌تواند قبل از آن درج شود یا پس از آن و یا اینکه کل محتوای مقصد را بازنویسی کند). HttpMethod آن هم به POST تنظیم شده تا با IE مشکلی نباشد. از این جهت که IE پیغام‌های GET را کش می‌کند و مساله ساز خواهد شد. توسط پارامتر routeValues، آرگومان مورد نظر به متد EmployeeInfo ارسال خواهد شد.
به علاوه یکی دیگر از خواص کلاس AjaxOptions، برای معرفی حالت بروز خطایی در سمت سرور به نام OnFailure در نظر گرفته شده است. در اینجا می‌توان نام یک متد JavaScript ایی را مشخص کرده و پیغام خطای عمومی را در صورت فراخوانی آن به کاربر نمایش داد. یا توسط خاصیت Confirm آن می‌توان یک پیغام را پیش از ارسال اطلاعات به سرور به کاربر نمایش داد.
به این ترتیب در مثال فوق، id=1 به متد EmployeeInfo به صورت غیرهمزمان ارسال می‌گردد. سپس کارمندی بر این اساس

یافت شده و در ادامه partial view مورد نظر بر اساس اطلاعات کاربر مذکور، رندر خواهد شد. نتیجه کار، در یک div با id مساوی EmployeeInfo درج می‌گردد (InsertionMode.Replace). متد Ajax.ActionLink از این جهت داخل div تعریف شده‌است که پس از کلیک کاربر و جایگزینی محتوا، محو شود. اگر نیازی به محو آن نبود، آن‌را خارج از div تعریف کنید.
عملیات دریافت اطلاعات از سرور ممکن است مدتی طول بکشد (برای مثال دریافت اطلاعات از بانک اطلاعاتی). به همین جهت بهتر است در این بین از تصاویری که نمایش دهنده انجام عملیات است، استفاده شود. برای این منظور یک div با id مساوی Progress تعریف شده و id آن به LoadingElementId انتساب داده شده است. این div با توجه به display: none آن، در ابتدای امر به کاربر نمایش داده نخواهد شد؛ در آغاز کار دریافت اطلاعات از سرور توسط متد Ajax.ActionLink نمایان شده و پس از خاتمه کار مجددا مخفی خواهد شد.
به علاوه اگر به کدهای فوق دقت کرده باشید، از متد Request.IsAjaxRequest نیز استفاده شده است. به این ترتیب می‌توان تشخیص داد که آیا درخواست رسیده از طرف jQuery Ajax صادر شده است یا خیر. البته آنچنان روش قابل ملاحظه‌ای نیست؛ چون امکان دستکاری Http Headers همیشه وجود دارد؛ اما بررسی آن ضرری ندارد. البته این نوع بررسی‌ها را در ASP.NET MVC بهتر است تبدیل به یک فیلتر سفارشی نمود؛ به این ترتیب حجم if و else نویسی در متدهای کنترلرها به حداقل خواهد رسید. برای مثال:

```
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Method)]
public class AjaxOnlyAttribute : ActionFilterAttribute
{
  public override void OnActionExecuting(ActionExecutingContext filterContext)
  {
    if (filterContext.HttpContext.Request.IsAjaxRequest())
    {
      base.OnActionExecuting(filterContext);
    }
    else
    {
      throw new InvalidOperationException("This operation can only be accessed via Ajax requests");
    }
  }
}
```

و برای استفاده از آن خواهیم داشت:

```
[AjaxOnly]
public ActionResult SomeAjaxAction()
{
    return Content("Hello!");
}
```

در مورد کلمه unobtrusive در قسمت بررسی نحوه اعتبار سنجی اطلاعات، توضیحاتی را ملاحظه نموده‌اید. در اینجا نیز از ویژگی‌های data-* برای معرفی پارامترهای مورد نیاز حین ارسال اطلاعات به سرور، استفاده می‌گردد. برای مثال خروجی متد Ajax.ActionLink به شکل زیر است. به این ترتیب امکان حذف کدهای جاوا اسکریپت از صفحه فراهم می‌شود و توسط یک فایل jquery.unobtrusive-ajax.min.js که توسط تیم ASP.NET MVC تهیه شده، اطلاعات مورد نیاز به سرور ارسال خواهد گردید:

```
<a data-ajax="true" data-ajax-loading="#Progress" data-ajax-method="POST"
    data-ajax-mode="replace" data-ajax-update="#EmployeeInfo"
    href="/Home/EmployeeInfo/1">Get Employee-1 info</a>
```

در کل این روش قابلیت نگهداری بهتری نسبت به روش اسکریپت نویسی مستقیم داخل صفحات را به همراه دارد. به علاوه جدا سازی افزونه اسکریپت وفق دهنده این اطلاعات با متد jQuery.Ajax از صفحه جاری، که امکان کش شدن آن‌را به سادگی میسر

می‌سازد.

به روز رسانی اطلاعات قسمتی از صفحه بدون استفاده از متد Ajax.ActionLink

الزامی به استفاده از متد Ajax.ActionLink و فایل jquery.unobtrusive-ajax.min.js وجود ندارد. اینکار را مستقیما به کمک jQuery نیز می‌توان به نحو زیر انجام داد:

```
<a href="#" onclick="LoadEmployeeInfo()">Get Employee-1 info</a>
@section javascript
{
    <script type="text/javascript">
        function LoadEmployeeInfo() {
            showProgress();
            $.ajax({
                type: "POST",
                url: "/Home/EmployeeInfo",
                data: JSON.stringify({ id: 1 }),
                contentType: "application/json; charset=utf-8",
                dataType: "json",
                // controller is returning a simple text, not json
                complete: function (xhr, status) {
                    var data = xhr.responseText;
                    if (status === 'error' || !data) {
                        //handleError
                    }
                    else {
                        $('#EmployeeInfo').html(data);
                    }
                    hideProgress();
                }
            });
        }
        function showProgress() {
            $('#Progress').css("display", "block");
        }
        function hideProgress() {
            $('#Progress').css("display", "none");
        }
    </script>
}
```

توضیحات:

توسط متد jQuery.Ajax نیز می‌توان درخواست‌های Ajax ایی خود را به سرور ارسال کرد. در اینجا type نوع http verb مورد نظر را مشخص می‌کند که به POST تنظیم شده است. Url آدرس کنترلر را دریافت می‌کند. البته حین استفاده از متد توکار Ajax.ActionLink، این لینک به صورت خودکار بر اساس تعاریف مسیریابی برنامه تنظیم می‌شود. اما در صورت استفاده مستقیم از jQuery.Ajax باید دقت داشت که با تغییر تعاریف مسیریابی برنامه نیاز است تا این Url نیز به روز شود.

سه سطر بعدی نوع اطلاعاتی را که باید به سرور POST شوند مشخص می‌کند. نوع json است و همچنین contentType آن برای ارسال اطلاعات یونیکد ضروری است. از متد JSON.stringify برای تبدیل اشیاء به رشته کمک گرفته‌ایم. این متد در تمام مرورگرهای امروزی به صورت توکار پشتیبانی می‌شود و استفاده از آن سبب خواهد شد تا اطلاعات به نحو صحیحی encode شده و به سرور ارسال شوند. بنابراین این رشته ارسالی اطلاعات را به صورت دستی تهیه نکنید؛ چون کاراکترهای زیادی هستند که ممکن است مشکل ساز شده و باید پیش از ارسال به سرور اصطلاحا escape یا encode شوند.

متداول است از پارامتر success برای دریافت نتیجه عملیات متد jQuery.Ajax استفاده شود. اما در اینجا از پارامتر complete آن استفاده شده است. علت هم اینجا است که return PartialView یک رشته را بر می‌گرداند. پارامتر success انتظار دریافت خروجی از نوع json را دارد. به همین جهت در این مثال خاص باید از پارامتر complete استفاده کرد تا بتوان به رشته بدون فرمت خروجی بدون مشکل دسترسی پیدا کرد.

به علاوه چون از یک section برای تعریف اسکریپت‌های مورد نیاز استفاده کرده‌ایم، برای درج خودکار آن در هدر صفحه باید قسمت هدر فایل layout برنامه را به صورت زیر مقدار دهی کرد:

```
@RenderSection("javascript", required: false)
```

دسترسی به اطلاعات یک مدل در View، به کمک jQuery Ajax

اگر جزئی از صفحه که قرار است به روز شود، پیچیده است، روش استفاده از partial viewها توصیه می‌شود؛ برای مثال می‌توان اطلاعات یک مدل را به همراه یک گرید کامل از اطلاعات، رندر کرد و سپس در صفحه درج نمود. اما اگر تنها به اطلاعات چند خاصیت از مدلی نیاز داشتیم، می‌توان از روش‌هایی با سربار کمتر نیز استفاده کرد. برای مثال متد جدید زیر را به کنترلر Home اضافه کنید:

```
[HttpPost] //for IE-8
public ActionResult EmployeeInfoData(int? id)
{
            if (!Request.IsAjaxRequest())
                return Json(false);

            if (!id.HasValue)
                return Json(false);

            var list = EmployeeDataSource.CreateEmployees();
            var data = list.Where(x => x.Id == id.Value).FirstOrDefault();
            if (data == null)
                return Json(false);

            return Json(data);
}
```

سپس View برنامه را نیز به نحو زیر تغییر دهید:

```
<a href="#" onclick="LoadEmployeeInfoData()">Get Employee-2 info</a>
@section javascript
{
    <script type="text/javascript">
        function LoadEmployeeInfoData() {
            showProgress();
            $.ajax({
                type: "POST",
                url: "/Home/EmployeeInfoData",
                data: JSON.stringify({ id: 1 }),
                contentType: "application/json; charset=utf-8",
                dataType: "json",
                // controller is returning the json data
                success: function (result) {
                    if (result) {
                        alert(result.Id + ' - ' + result.Name);
                    }
                    hideProgress();
                },
                error: function (result) {
                    alert(result.status + ' ' + result.statusText);
                    hideProgress();
                }
            });
        }

        function showProgress() {
            $('#Progress').css("display", "block");
        }
        function hideProgress() {
            $('#Progress').css("display", "none");
        }
    </script>
}
```

در این مثال، کنترلر برنامه، اطلاعات مدل را تبدیل به Json کرده و بازگشت خواهد داد. سپس می‌توان به اطلاعات این مدل و خواص آن در View برنامه، در پارامتر success متد jQuery.Ajax، مطابق کدهای فوق دسترسی یافت. اینبار چون خروجی کنترلر تعریف شده از نوع Json است، امکان استفاده از پارامتر success فراهم شده است. همه چیز هم در اینجا خودکار است؛ تبدیل یک شیء به Json و برعکس.
یک نکته: اگر نوع متد کنترلر، HttpGet باشد، نیاز خواهد بود تا پارامتر دوم متد بازگشت Json، مساوی JsonRequestBehavior.AllowGet قرار داده شود.

ارسال اطلاعات فرم‌ها به سرور، به کمک ویژگی‌های Ajax

متد کمکی توکار دیگری به نام Ajax.BeginForm در ASP.NET MVC وجود دارد که کار ارسال غیرهمزمان اطلاعات یک فرم را به سرور انجام داده و سپس اطلاعاتی را از سرور دریافت و قسمتی از صفحه را به روز خواهد کرد. مکانیزم کاری کلی آن بسیار شبیه به متد Ajax.ActionLink می‌باشد. در ادامه با تکمیل مثال قسمت جاری، به بررسی این ویژگی خواهیم پرداخت.
ابتدا متد جستجوی زیر را به کنترلر برنامه اضافه کنید:

```
[HttpPost] //for IE-8
public ActionResult SearchEmployeeInfo(string data)
{
            if (!Request.IsAjaxRequest())
                return Content(string.Empty);

            if (string.IsNullOrWhiteSpace(data))
                return Content(string.Empty);

            var employeesList = EmployeeDataSource.CreateEmployees();
            var list = employeesList.Where(x => x.Name.Contains(data)).ToList();
            if (list == null || !list.Any())
                return Content(string.Empty);

            return PartialView(viewName: "_SearchEmployeeInfo", model: list);
}
```

سپس بر روی نام متد کلیک راست کرده و گزینه add view را انتخاب کنید. در صفحه باز شده، گزینه create a stronlgly typed view را انتخاب کرده و قالب scaffolding را هم بر روی list قرار دهید. سپس گزینه ایجاد partial view را نیز انتخاب کنید. نام آن‌را هم _SearchEmployeeInfo وارد نمائید. برای نمونه خروجی حاصل به نحو زیر خواهد بود:

```
@model IEnumerable<MvcApplication18.Models.Employee>

<table>
    <tr>
        <th>
            Name
        </th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
    </tr>
}

</table>
```

تا اینجا یک متد جستجو را ایجاد کرده‌ایم که می‌تواند لیستی از رکوردهای کارمندان را بر اساس قسمتی از نام آن‌ها که توسط کاربری جستجو شده است، بازگشت دهد. سپس این اطلاعات را به partial view مورد نظر ارسال کرده و یک جدول را بر اساس آن تولید خواهیم نمود.
اکنون به فایل Index.cshtml مراجعه کرده و فرم Ajax ایی زیر را اضافه نمائید:

```
@using (Ajax.BeginForm(actionName: "SearchEmployeeInfo",
                       controllerName: "Home",
                       ajaxOptions: new AjaxOptions
                                    {
                                         HttpMethod = "POST",
                                         InsertionMode = InsertionMode.Replace,
                                         UpdateTargetId = "EmployeeInfo",
                                         LoadingElementId = "Progress"
                                    }))
    {
      @Html.TextBox("data")
      <input type="submit" value="Search" />
    }
```

اینبار بجای استفاده از متد Html.BeginForm از متد Ajax.BeginForm استفاده شده است. به کمک آن اطلاعات Html.TextBox تعریف شده، به کنترلر Home و متد SearchEmployeeInfo آن، بر اساس HttpMethod تعریف شده، ارسال گردیده و نتیجه آن در یک div با id مساوی EmployeeInfo درج می‌گردد. همچنین اگر اطلاعاتی یافت نشد، به کمک متد return Content یک رشته خالی بازگشت داده می‌شود.
متد Ajax.BeginForm نیز از ویژگی‌های data-* برای تعریف اطلاعات مورد نیاز ارسالی به سرور استفاده می‌کند. بنابراین نیاز به سطر الحاق jquery.unobtrusive-ajax.min.js در فایل layout برنامه جهت وفق دادن این اطلاعات unobtrusive به اطلاعات مورد نیاز متد jQuery.Ajax وجود دارد.

```
<form action="/Home/SearchEmployeeInfo" data-ajax="true"
          data-ajax-loading="#Progress" data-ajax-method="POST"
          data-ajax-mode="replace" data-ajax-update="#EmployeeInfo"
          id="form0" method="post">
       <input id="data" name="data" type="text" value="" />
       <input type="submit" value="Search" />
</form>
```

کتابخانه کمکی «ASP.net MVC Awesome - jQuery Ajax Helpers»

علاوه بر متدهای توکار Ajax همراه با ASP.NET MVC، سایر علاقمندان نیز یک سری Ajax helper را بر اساس افزونه‌های jQuery تدارک دیده‌اند که از آدرس زیر قابل دریافت هستند:

/http://awesome.codeplex.com

افزودن فرم‌ها به کمک jQuery.Ajax و فعال سازی اعتبار سنجی سمت کلاینت

در ASP.NET MVC چون ViewState حذف شده است، امکان تزریق فرم‌های جدید به صفحه یا به روز رسانی قسمتی از صفحه توسط jQuery Ajax به سهولت و بدون دریافت پیغام «viewstate is corrupted» در حین ارسال اطلاعات به سرور، میسر است.
در این حالت باید به یک نکته مهم نیز دقت داشت: «اعتبار سنجی سمت کلاینت دیگر کار نمی‌کند». علت اینجا است که در حین بارگذاری متداول یک صفحه، متد زیر به صورت خودکار فراخوانی می‌گردد:

```
$.validator.unobtrusive.parse("#{form-id}");
```

اما با به روز رسانی قسمتی از صفحه، دیگر اینچنین نخواهد بود و نیاز است این فراخوانی را دستی انجام دهیم. برای مثال:

```
$.ajax
({
    url: "/{controller}/{action}/{id}",
    type: "get",
    success: function(data)
    {
        $.validator.unobtrusive.parse("#{form-id}");
    }
});

//or
$.get('/{controller}/{action}/{id}', function (data) { $.validator.unobtrusive.parse("#{form-id}"); });
```

شبیه به همین مساله را حین استفاده از Ajax.BeginForm نیز باید مد نظر داشت:

```
@using (Ajax.BeginForm(
    "Action1",
    "Controller",
    null,
    new AjaxOptions {
        OnSuccess = "onSuccess",
        UpdateTargetId = "result"
    },
    null)
)
{
    <input type="submit" value="Save" />
}

var onSuccess = function(result) {
    // enable unobtrusive validation for the contents
    // that was injected into the <div id="result"></div> node
    $.validator.unobtrusive.parse("#result");
};
```

در این مثال در پارامتر UpdateTargetId، مجددا یک فرم رندر می‌شود. بنابراین اعتبار سنجی سمت کلاینت آن دیگر کار نخواهد کرد مگر اینکه با مقدار دهی خاصیت OnSuccess، مجددا متد unobtrusive.parse را فراخوانی کنیم.

عنوان: ASP.NET MVC #22
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۶ ۲۲:۳۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

تهیه سایت‌های چند زبانه و بومی سازی نمایش اطلاعات در ASP.NET MVC

زمانیکه دات نت فریم ورک نیاز به انجام اعمال حساس به مسایل بومی را داشته باشد، ابتدا به مقادیر تنظیم شده دو خاصیت زیر دقت می‌کند:

الف) System.Threading.Thread.CurrentThread.CurrentCulture

بر این اساس دات نت می‌تواند تشخیص دهد که برای مثال خروجی متد DateTime.Now.ToString در کانادا و آمریکا باید با هم تفاوت داشته باشند. مثلا در آمریکا ابتدا ماه، سپس روز و در آخر سال نمایش داده می‌شود و در کانادا ابتدا سال، بعد ماه و در آخر روز نمایش داده خواهد شد. یا نمونه‌ی دیگری از این دست می‌تواند نحوه نمایش علامت واحد پولی کشورها باشد.

ب) System.Threading.Thread.CurrentThread.CurrentUICulture

مقدار CurrentUICulture بر روی بارگذاری فایل‌های مخصوصی به نام Resource، تاثیر گذار است.

این خواص را یا به صورت دستی می‌توان تنظیم کرد و یا ASP.NET، این اطلاعات را از هدر Accept-Language دریافتی از مرورگر کاربر به صورت خودکار مقدار دهی می‌کند. البته برای این منظور نیاز است یک سطر زیر را به فایل وب کانفیگ برنامه اضافه کرد:

```
<system.web>
    <globalization culture="auto" uiCulture="auto" />
```

یا اگر نیاز باشد تا برنامه را ملزم به نمایش اطلاعات Resource مرتبط با فرهنگ بومی خاصی کرد نیز می‌توان در همین قسمت مقادیر culture و uiCulture را دستی تنظیم نمود و یا اگر همانند برنامه‌هایی که چند لینک را بالای صفحه نمایش می‌دهند که برای مثال به نگارش‌های فارسی/عربی/انگلیسی اشاره می‌کند، اینکار را با کد نویسی نیز می‌توان انجام داد:

```
System.Threading.Thread.CurrentThread.CurrentCulture =
                                    System.Globalization.CultureInfo.CreateSpecificCulture("fa");
```

جهت آزمایش این مطلب، ابتدا تنظیم globalization فوق را به فایل وب کانفیگ برنامه اضافه کنید. سپس به مسیر زیر در IE مراجعه کنید:

```
IE -> Tools -> Intenet options -> Genarl tab -> Languages
```

در اینجا می‌توان هدر Accept-Language را مقدار دهی کرد. برای نمونه اگر مقدار زبان پیش فرض را به فرانسه تنظیم کنیم (به عنوان اولین زبان تعریف شده در لیست) و سپس سعی در نمایش مقدار decimal زیر را داشته باشیم:

```
string.Format("{0:C}", 10.5M)
```

اگر زبان پیش فرض، انگلیسی آمریکایی باشد، $ نمایش داده خواهد شد و اگر زبان به فرانسه تنظیم شود، یورو در کنار عدد مبلغ نمایش داده می‌شود.
تا اینجا تنها با تنظیم culture=auto به این نتیجه رسیده‌ایم. اما سایر قسمت‌های صفحه چطور؟ برای مثال برچسب‌های نمایش داده شده را چگونه می‌توان به صورت خودکار بر اساس Accept-Language مرجح کاربر تنظیم کرد؟ خوشبختانه در دات نت، زیر ساخت مدیریت برنامه‌های چند زبانه به صورت توکار وجود دارد که در ادامه به بررسی آن خواهیم پرداخت.

آشنایی با ساختار فایل‌های Resource

فایل‌های Resource یا منبع، در حقیقت فایل‌هایی هستند مبتنی بر XML با پسوند resx و هدف آن‌ها ذخیره سازی رشته‌های متناظر با فرهنگ‌های مختلف می‌باشد و برای استفاده از آن‌ها حداقل یک فایل منبع پیش فرض باید تعریف شود. برای نمونه فایل mydata.resx را در نظر بگیرید. برای ایجاد فایل منبع اسپانیایی متناظر، باید فایلی را به نام mydata.es.resx تولید کرد. البته نوع فرهنگ مورد استفاده را کاملتر نیز می‌توان ذکر کرد برای مثال mydata.es-mex.resx جهت فرهنگ اسپانیایی مکزیکی بکارگرفته خواهد شد، یا mydata.fr-ca.resx به فرانسوی کانادایی اشاره می‌کند. سپس مدیریت منابع دات نت فریم ورک بر اساس مقدار CurrentUICulture جاری، اطلاعات فایل متناظری را بارگذاری خواهد کرد. اگر فایل متناظری وجود نداشت، از اطلاعات همان فایل پیش فرض استفاده می‌گردد.
حین تهیه برنامه‌ها نیازی نیست تا مستقیما با فایل‌های XML منابع کار کرد. زمانیکه اولین فایل منبع تولید می‌شود، به همراه آن یک فایل cs یا vb نیز ایجاد خواهد شد که امکان دسترسی به کلیدهای تعریف شده در فایل‌های XML را به صورت strongly typed میسر می‌کند. این فایل‌های خودکار، تنها برای فایل پیش فرض mydata.resx تولید می‌شوند،از این جهت که تعاریف اطلاعات سایر فرهنگ‌های متناظر نیز باید با همان کلیدهای فایل پیش فرض آغاز شوند. تنها «مقادیر» کلیدهای تعریف شده در کلاس‌های منبع متفاوت هستند.
اگر به خواص فایل‌های resx در VS.NET دقت کنیم، نوع Build action آن‌ها به embedded resource تنظیم شده است.

مثالی جهت بررسی استفاده از فایل‌های Resource

یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. فایل وب کانفیگ آن‌را ویرایش کرده و تنظیمات globalization ابتدای بحث را به آن اضافه کنید. سپس مدل، کنترلر و View متناظر با متد Index آن‌را با محتوای زیر به پروژه اضافه نمائید:

```
namespace MvcApplication19.Models
{
    public class Employee
    {
        public int Id { set; get; }
        public string Name { set; get; }
    }
}
```

```
using System.Web.Mvc;
using MvcApplication19.Models;

namespace MvcApplication19.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            var employee = new Employee { Name = "Name 1" };
            return View(employee);
        }
    }
}
```

```
@model MvcApplication19.Models.Employee
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
<fieldset>
    <legend>Employee</legend>
    <div class="display-label">
        Name
    </div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Name)
    </div>
</fieldset>
<fieldset>
    <legend>Employee Info</legend>
    @Html.DisplayForModel()
</fieldset>
```

قصد داریم در View فوق بر اساس uiCulture کاربر مراجعه کننده به سایت، برچسب Name را مقدار دهی کنیم. اگر کاربری از ایران مراجعه کند، «نام کارمند» نمایش داده شود و سایر کاربران، «Employee Name» را مشاهده کنند. همچنین این تغییرات باید بر روی متد Html.DisplayForModel نیز تاثیرگذار باشد.

برای این منظور بر روی پوشه Views/Home که محل قرارگیری فایل Index.cshtml فوق است کلیک راست کرده و گزینه Add|New Item را انتخاب کنید. سپس در صفحه ظاهر شده، گزینه «Resources file» را انتخاب کرده و برای مثال نام Index_cshtml.resx را وارد کنید.

به این ترتیب اولین فایل منبع مرتبط با View جاری که فایل پیش فرض نیز می‌باشد ایجاد خواهد شد. این فایل، به همراه فایل Index_cshtml.Designer.cs تولید می‌شود. سپس همین مراحل را طی کنید، اما اینبار نام Index_cshtml. **fa** .resx را حین افزودن فایل منبع وارد نمائید که برای تعریف اطلاعات بومی ایران مورد استفاده قرار خواهد گرفت. فایل دومی که اضافه شده است، فاقد فایل cs همراه می‌باشد.

اکنون فایل Index_cshtml.resx را در VS.NET باز کنید. از بالای صفحه، به کمک گزینه Access modifier، سطح دسترسی متدهای فایل cs همراه آن‌را به public تغییر دهید. پیش فرض آن internal است که برای کار ما مفید نیست. از این جهت که امکان دسترسی به متدهای استاتیک تعریف شده در فایل خودکار Index_cshtml.Designer.cs را در View های برنامه، نخواهیم داشت. سپس دو جفت «نام-مقدار» را در فایل resx وارد کنید. مثلا نام را Name و مقدار آن‌را «Employee Name» و سپس نام دیگر را NameIsNotRight و مقدار آن‌را «Name is required» وارد نمائید.

در ادامه فایل Index_cshtml.fa.resx را باز کنید. در اینجا نیز دو جفت «نام-مقدار» متناظر با فایل پیش فرض منبع را باید وارد کرد. کلیدها یا نام‌ها یکی است اما قسمت مقدار اینبار باید فارسی وارد شود. مثلا نام را Name و مقدار آن‌را «نام کارمند» وارد نمائید. سپس کلید یا نام NameIsNotRight و مقدار «لطفا نام را وارد نمائید» را تنظیم نمائید.

تا اینجا کار تهیه فایل‌های منبع متناظر با View جاری به پایان می‌رسد.

در ادامه با کمک فایل Index_cshtml.Designer.cs که هربار پس از تغییر فایل resx متناظر آن به صورت خودکار توسط VS.NET تولید و به روز می‌شود، می‌توان به کلیدها یا نام‌هایی که تعریف کرده‌ایم، در قسمت‌های مختلف برنامه دست یافت. برای نمونه تعریف کلید Name در این فایل به نحو زیر است:

```
namespace MvcApplication19.Views.Home {
    public class Index_cshtml {
        public static string Name {
            get {
                return ResourceManager.GetString("Name", resourceCulture);
            }
        }
    }
}
```

بنابراین برای استفاده از آن در هر View ایی تنها کافی است بنویسیم:

```
@MvcApplication19.Views.Home.Index_cshtml.Name
```

به این ترتیب بر اساس تنظیمات محلی کاربر، اطلاعات به صورت خودکار از فایل‌های Index_cshtml.fa.resx فارسی یا فایل پیش فرض Index_cshtml.resx، دریافت می‌گردد.
علاوه بر امکان دسترسی مستقیم به کلیدهای تعریف شده در فایل‌های منبع، امکان استفاده از آن‌ها توسط data annotations نیز میسر است. در این حالت می‌توان مثلا پیغام‌های اعتبار سنجی را بومی کرد یا حین استفاده از متد Html.DisplayForModel، بر روی برچسب نمایش داده شده خودکار، تاثیر گذار بود. برای اینکار باید اندکی مدل برنامه را ویرایش کرد:

```
using System.ComponentModel.DataAnnotations;

namespace MvcApplication19.Models
{
    public class Employee
    {
        [ScaffoldColumn(false)]
        public int Id { set; get; }

        [Display(ResourceType = typeof(MvcApplication19.Views.Home.Index_cshtml),
                 Name = "Name")]
        [Required(ErrorMessageResourceType = typeof(MvcApplication19.Views.Home.Index_cshtml),
                  ErrorMessageResourceName = "NameIsNotRight")]
        public string Name { set; get; }
    }
}
```

همانطور که ملاحظه می‌کنید، حین تعریف ویژگی‌های Display یا Required، امکان تعریف نام کلاس متناظر با فایل resx خاصی وجود دارد. به علاوه ErrorMessageResourceName به نام یک کلید در این فایل و یا پارامتر Name ویژگی Display نیز به نام کلیدی در فایل منبع مشخص شده، اشاره می‌کنند. این اطلاعات توسط متدهای Html.DisplayForModel، Html.ValidationMessageFor، Html.LabelFor و امثال آن به صورت خودکار مورد استفاده قرار خواهند گرفت.

نکته‌ای در مورد کش کردن اطلاعات

در این مثال اگر فیلتر OutputCache را بر روی متد Index تعریف کنیم، حتما نیاز است به هدر Accept-Language نیز دقت داشت. در غیراینصورت تمام کاربران، صرفنظر از تنظیمات بومی آن‌ها، یک صفحه را مشاهده خواهند کرد:

```
[OutputCache(Duration = 60, VaryByHeader = "Accept-Language")]
public ActionResult Index()
```

عنوان: **ASP.NET MVC #23**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۷ ۱۱:۱۲:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

اجرای برنامه‌های ASP.NET MVC توسط نگارش‌های متفاوت IIS

تا اینجا برای اجرای برنامه‌های ASP.NET MVC از وب سرور توکار VS.NET استفاده شد که صرفا جهت آزمایش برنامه‌ها طراحی شده است. تا این تاریخ سه رده از وب سرورهای مایکروسافت ارائه شده‌اند که برای نصب ASP.NET MVC می‌توانند مورد استفاده قرار گیرند و هر کدام هم نکته‌های خاص خودشان را دارند که در ادامه به بررسی آن‌ها خواهیم پرداخت.

اجرای برنامه‌های ASP.NET MVC بر روی IIS 5.x ویندوز XP

پس از ایجاد یک دایرکتوری مجازی بر روی پوشه یک برنامه ASP.NET MVC و سعی در اجرای برنامه، بلافاصله پیغام خطای HTTP 403 forbidden مشاهده می‌شود.
اولین کاری که برای رفع این مساله باید صورت گیرد، کلیک راست بر روی نام دایرکتوری مجازی در کنسول IIS، انتخاب گزینه خواص و سپس مراجعه به برگه «ASP.NET» آن است. در اینجا شماره نگارش دات نت فریم ورک مورد استفاده را به 4 تغییر دهید (برای نمونه ASP.NET MVC 3.0 مبتنی بر دات نت فریم ورک 4 است).
بعد از این تغییر، بازهم موفق به اجرای برنامه‌های ASP.NET MVC بر روی IIS 5.x نخواهیم شد؛ چون در آن زمان مفاهیم مسیریابی و Routing که اصل و پایه ASP.NET MVC هستند وجود خارجی نداشتند. این نگارش از IIS به صورت پیش فرض تنها قادر به پردازش درخواست‌های رسیده‌ای که به یک فایل فیزیکی بر روی سرور اشاره می‌کند، می‌باشد (یعنی مشکلی با اجرای برنامه‌های ASP.NET Web forms ندارد).
برای رفع این مشکل، مجددا بر روی نام دایرکتوری مجازی برنامه در کنسول IIS کلیک راست کرده و گزینه خواص را انتخاب کنید. در صفحه ظاهر شده، در برگه «Virtual directory» آن، بر روی دکمه «Configuration» کلیک نمائید. در صفحه باز شده مجددا بر روی دکمه «Add» کلیک کنید.
در صفحه باز شده، مسیر Executable را C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\aspnet_isapi.dll وارد کرده و Extension را به .* (دات هرچی) تنظیم کنید. همین مقدار تنظیم، برای اجرای برنامه‌های ASP.NET MVC بر روی IIS 5.x ویندوز XP کفایت می‌کند.

کاری که در اینجا انجام شده است، نگاشت تمام درخواست‌های رسیده صرفنظر از پسوند فایل‌ها، به موتور ASP.NET می‌باشد. به صورت پیش فرض در IIS 5.x درخواست‌ها تنها بر اساس پسوند فایل‌ها پردازش می‌شوند. مثلا اگر فایل درخواستی aspx است، درخواست رسیده به aspnet_isapi.dll یاد شده هدایت خواهد شد. اگر پسوند فایل php است به isapi مخصوص آن (در صورت نصب) هدایت می‌گردد و به همین ترتیب برای سایر سیستم‌های دیگر. زمانیکه Extension به «دات هرچی» و Executable به aspnet_isapi.dll دات نت 4 تنظیم می‌شود، دایرکتوری مجازی تنظیم شده تنها جهت سرویس دهی به یک برنامه ASP.NET عمل خواهد کرد و تمام درخواست‌های رسیده به آن، به موتور اجرایی ASP.NET هدایت می‌شوند.

بدیهی است تنظیمات فوق تنها به یک دایرکتوری مجازی اعمال شدند. اگر نیاز باشد تا بر روی تمام سایت‌ها تاثیر گذار شود، اینبار در کنسول IIS 5.x بر روی «Default web site» کلیک راست کرده و گزینه خواص را انتخاب کنید. در صفحه باز شده به برگه «Home directory» مراجعه کرده و مراحل ذکر شده را تکرار کنید.

مشکل! این روش بهینه نیست.
روش فوق خوبه، کار می‌کنه، اما بهینه نیست؛ از این جهت که «نگاشت تمام درخواست‌ها به موتور ASP.NET» یعنی پروسه پردازش درخواست یک فایل تصویری، js یا css هم باید از فیلتر موتور ASP.NET عبور کند که ضروری نیست.
برای رفع این مشکل، توصیه شده است که سیستم مسیریابی ASP.NET MVC را در IIS 5.x «پسوند دار» کنید. به این نحو که با

مراجعه به فایل Global.asax.cs، تعاریف مسیریابی را به نحو زیر ویرایش کنید:

```
public static void RegisterRoutes(RouteCollection routes)
{
       routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
       routes.Add(
                new Route("{controller}.aspx/{action}/{id}", new MvcRouteHandler())
                {
                    Defaults = new RouteValueDictionary(new
                    {
                        controller = "Home",
                        action = "Index",
                        id = UrlParameter.Optional
                    })
                });
```

اینبار برای مثال مسیر http://localhost/MyMvcApp/home.aspx/index به علت داشتن پسوند aspx وارد موتور پردازشی ASP.NET خواهد شد. البته در این حالت URL های تمیز ASP.NET MVC را از دست خواهیم داد و مدام باید دقت داشت که مسیرهای کنترلرها حتما باید به aspx ختم شوند. ضمنا با این تنظیم، دیگر نیازی به تغییر تعاریف نگاشت‌ها در کنسول مدیریتی IIS، نخواهد بود.

اجرای برنامه‌های ASP.NET MVC بر روی IIS 6.x ویندوز سرور 2003

تمام نکات عنوان شده جهت IIS 5.x در IIS 6.x نیز صادق هستند. به علاوه برای اجرای برنامه‌های ASP.NET بر روی IIS 6.x باید به دو نکته مهم دیگر نیز دقت داشت:
الف) ASP.NET 4 به صورت پیش فرض در IIS 6.x غیرفعال است که باید با مراجعه به قسمت Web Services Extensions در کنسول مدیریتی IIS، آن‌را از حالت prohibited خارج کرد.
ب) در هر Application pool تنها از یک نگارش دات نت فریم ورک می‌توان استفاده کرد. برای مثال اگر هم اکنون AppPool1 مشغول سرویس دهی به یک سایت ASP.NET 3.5 است، از آن نمی‌توانید جهت اجرای برنامه‌های ASP.NET MVC 3 به بعد استفاده کنید. زیرا برای مثال ASP.NET MVC 3 مبتنی بر دات نت فریم ورک 4 است. به همین جهت حتما نیاز است تا یک Application pool مجزا را برای برنامه‌های دات نت 4 در IIS 6 اضافه نمائید و سپس در تنظیمات سایت، از این Application pool جدید استفاده نمائید.
البته روش صحیح و اصولی کار با IIS از نگارش 6 به بعد هم مطابق شرحی است که عنوان شد. برای دستیابی به بهترین کارآیی و امنیت بیشتر، بهتر است به ازای هر سایت، از یک Application pool مجزا استفاده نمائید.

اطلاعات تکمیلی:
نکات نصب برنامه‌های ASP.NET 4.0 بر روی IIS 6
مروری بر تاریخچه محدودیت حافظه مصرفی برنامه‌های ASP.NET در IIS

اجرای برنامه‌های ASP.NET MVC بر روی IIS 7.x ویندوز 7 و ویندوز سرور 2008

اگر برنامه ASP.NET MVC در IIS 7.x در حالت یکپارچه (integrated mode) اجرا شود، بدون نیاز به هیچگونه تغییری در تنظیمات سرور یا برنامه، بدون مشکل قابل اجرا خواهد بود. بدیهی است در اینجا نیز بهتر است به ازای هر برنامه، یک Application pool مجزا را ایجاد کرد.
اما در حالت classic (که برای برنامه‌های جدید توصیه نمی‌شود) نیاز است همان مراحل IIS 5,x تکرار شود. البته اینبار مسیر زیر را باید طی کرد تا به صفحه افزودن نگاشت‌ها رسید:

```
Right-click on a web site -> Properties -> Home Directory tab -> click on the Configuration button -> Mappings tab
```

نکته‌ای مهم در تمام نگارش‌های IIS

ترتیب نصب دات نت فریم ورک 4 و IIS مهم است. اگر ابتدا IIS نصب شود و سپس دات نت فریم ورک 4، به صورت خودکار، کار نگاشت اطلاعات ASP.NET به IIS صورت خواهد گرفت.
اگر ابتدا دات نت فریم ورک 4 نصب شود و سپس IIS، برای مثال دیگر از برگه ASP.NET در IIS 6.x خبری نخواهد بود. برای رفع این مشکل دستور زیر را در خط فرمان اجرا کنید:

```
C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\aspnet_regiis.exe /i
```

به این ترتیب، اطلاعات مرتبط با موتور ASP.NET مجددا به تنظیمات IIS اضافه خواهند شد.

عنوان: **ASP.NET MVC #24**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۸ ۰۰:۲۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

مروری بر نمونه سؤالات ASP.NET MVC امتحانات مایکروسافت در چند سال اخیر

در قسمت آخر سری ASP.NET MVC بد نیست مروری داشته باشیم بر نمونه سؤالات امتحانات مایکروسافت؛ امتحانات 515-70 و 519-70 که در آن‌ها تعدادی از سؤالات به ASP.NET MVC اختصاص دارند. در این سؤالات امکان انتخاب بیش از یک گزینه نیز وجود دارد.

1) شما در حال توسعه یک برنامه‌ی ASP.NET MVC هستید. باید درخواست Ajax ایی از صفحه‌ای صادر شده و خروجی زیر را از اکشن متدی دریافت کند:

```
["Adventure Works","Contoso"]
```

کدام نوع خروجی اکشن متد زیر را برای اینکار مناسب می‌دانید؟

```
a) AjaxHelper
b) XDocument
c) JsonResult
d) DataContractJsonSerializer
```

2) شما در حال طراحی یک برنامه ASP.NET MVC هستید. محتوای یک View باید بر اساس نیازمندی‌های زیر تشکیل شود:
الف) ارائه محتوای رندر شده user controls/partial views به مرورگر
ب) کار انتخاب user controls/partial views مناسب در اکشن متد کنترلر باید انجام شود
استفاده از کدام روش زیر را توصیه می‌کنید؟

```
a) Use the Html.RenderPartial extension method
b) Use the Html.RenderAction extension method
c) Use the PartialViewResult class
d) Use the ContentResult class
```

3) در حین طراحی یک برنامه ASP.NET MVC، نیاز است منطق مدیریت استثناهای رخ داده و همچنین ثبت وقایع مرتبط را در یک مکان یا کلاس مرکزی مدیریت کنید. کدام روش زیر را پیشنهاد می‌دهید؟
a) استفاده از try/catch در تمام متدها
b) تحریف متد OnException در کنترلرها
c) مزین سازی تمام کنترلرها به ویژگی HandleError سفارشی شده
d) مزین سازی تمام کنترلرها به ویژگی HandleError پیش فرض

4) شما در حال توزیع برنامه‌ی ASP.NET MVC خود جهت اجرا بر روی IIS 6.x هستید. چه ملاحظاتی را باید مدنظر داشته باشید تا برنامه به درستی کار کند؟
a) تنظیم IIS به نحوی‌که تمام درخواست‌ها را بر اساس wildcard خاصی به aspnet_isapi.dll هدایت کند.

b) تنظیم IIS به نحوی‌که تمام درخواست‌ها را بر اساس wildcard خاصی به aspnet_wp.exe هدایت کند.
c) تغییر برنامه به نحوی‌که تمام درخواست‌ها را به یک HttpHandler خاص هدایت کند.
d) تغییر برنامه به نحوی‌که تمام درخواست‌ها را به یک HttpModule خاص هدایت کند.

5) شما در حال توسعه برنامه‌ی ASP.NET MVC هستید که در پوشه Views/Shared/DisplayTemplates آن، فایلی به نام score.cshtml به عنوان یک templated helper نمایش سفارشی اعداد صحیح تعریف شده است. مدل برنامه هم مطابق تعاریف زیر است:

```
public class Player
{
   public String Name { get; set; }
   public int LastScore { get; set; }
   public int HighScore { get; set; }
}
```

در اینجا اگر نیاز باشد تا فایل score.cshtml یاد شده به صورت خودکار به خاصیت LastScore در حین فراخوانی متد HtmlHelper.DisplayForModel اعمال شود، چه روشی را پیشنهاد می‌دهید؟
a) فایل score.cshtml باید به LastScore.cshtml تغییر نام یابد.
b) فایل یاد شده باید از پوشه Views/Shared/DisplayTemplates به پوشه Views/Player/DisplayTemplates منتقل شود.
c) باید از ویژگی UIHint به همراه مقدار score جهت مزین سازی خاصیت LastScore استفاده کرد.

```
[UIHint("Score")]
```

d) باید از ویژگی زیر برای مزین سازی خاصیت مورد نظر استفاده کرد:

```
[Display(Name="LastScore", ShortName="Score")]
```

6) شما در حال طراحی برنامه‌ی ASP.NET MVC هستید که در آن متد Edit کنترلری باید تنها توسط کاربران اعتبارسنجی شده قابل دسترسی باشد. استفاده از کدام دو گزینه زیر را برای این منظور توصیه می‌کنید؟

```
a) [Authorize(Users = "")]
b) [Authorize(Roles = "")]
c) [Authorize(Users = "*")]
d) [Authorize(Roles = "*")]
```

7) قطعه کد HTML زیر را درنظر بگیرید:

```
<span id="ref">
<a name=Reference>Check out</a>
the FAQ on
<a href="http://www.contoso.com">
Contoso</a>'s web site for more information:
<a href="http://www.contoso.com/faq">FAQ</a>.
</span>
<a href="http://www.contoso.com/home">Home</a>
```

قصد داریم به کمک jQuery در span ایی با id مساوی ref، متن تمام لینک‌ها را ضخیم کنیم. کدام گزینه زیر را پیشنهاد می‌دهید؟

```
a) $("#ref").filter("a[href]").bold();
b) $("ref").filter("a").css("bold");
c) $("a").css({fontWeight:"bold"});
d) $("#ref a[href]").css({fontWeight:"bold"});
```

عنوان: CheckBoxList در ASP.NET MVC
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۰ ۲۰:۴۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVC

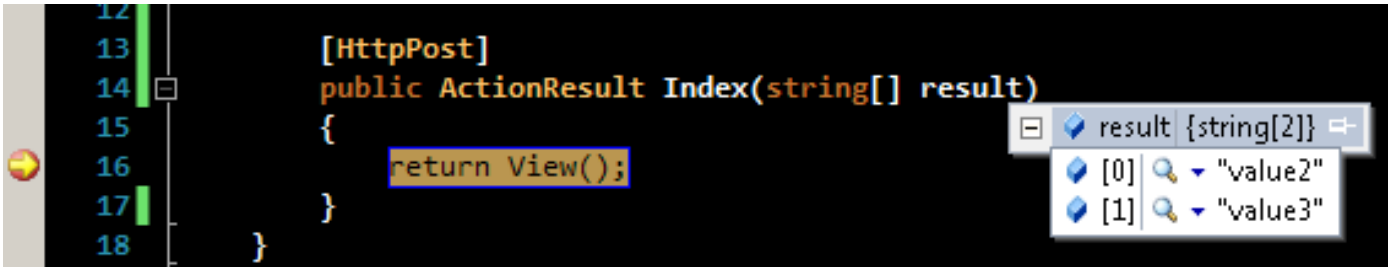
ASP.NET MVC به همراه HtmlHelper توکاری جهت نمایش یک ChekBoxList نیست؛ اما سیستم Model binder آن، این نوع کنترل‌ها را به خوبی پشتیبانی می‌کند. برای مثال، یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. سپس یک کنترلر Home جدید را نیز به آن اضافه کنید. در ادامه، برای متد Index آن، یک View خالی را ایجاد نمائید. سپس محتوای این View را به نحو زیر تغییر دهید:

```
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
@using (Html.BeginForm())
{
    <input type='checkbox' name='Result' value='value1' />
    <input type='checkbox' name='Result' value='value2' />
    <input type='checkbox' name='Result' value='value3' />
    <input type="submit" value="submit" />
}
```

و کنترلر Home را نیز مطابق کدهای زیر ویرایش کنید:

```
using System.Web.Mvc;

namespace MvcApplication21.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult Index(string[] result)
        {
            return View();
        }
    }
}
```

یک breakpoint را در تابع Index دوم که آرایه‌ای را دریافت می‌کند، قرار دهید. سپس برنامه را اجرا کرده، تعدادی از checkboxها را انتخاب و فرم نمایش داده شده را به سرور ارسال کنید:



شکل ۱

بله. همانطور که ملاحظه می‌کنید، تمام عناصر ارسالی انتخاب شده که دارای نامی مشابه بوده‌اند، به یک آرایه قابل بایند هستند و سیستم model binder می‌داند که چگونه باید این اطلاعات را دریافت و پردازش کند.
از این مقدمه می‌توان به عنوان پایه و اساس نوشتن یک HtmlHelper سفارشی CheckBoxList استفاده کرد.
برای این منظور یک پوشه جدید را به نام app_code، به ریشه پروژه اضافه نمائید. سپس یک فایل خالی را به نام Helpers.cshtml نیز به آن اضافه کنید. محتوای این فایل را به نحو زیر تغییر دهید:

```
@helper CheckBoxList(string name, List<System.Web.Mvc.SelectListItem> items)
    {
    <div class="checkboxList">
        @foreach (var item in items)
        {
            @item.Text
            <input type="checkbox" name="@name"
                   value="@item.Value"
                   @if (item.Selected) { <text>checked="checked"</text> }
               />
            < br />
        }
    </div>
}
```

و برای استفاده از آن، کنترلر Home را مطابق کدهای زیر ویرایش کنید:

```
using System.Collections.Generic;
using System.Web.Mvc;

namespace MvcApplication21.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public ActionResult Index()
        {
            ViewBag.Tags = new List<SelectListItem>
            {
                new SelectListItem { Text = "Item1", Value = "Val1", Selected = false },
                new SelectListItem { Text = "Item2", Value = "Val2", Selected = false },
                new SelectListItem { Text = "Item3", Value = "Val3", Selected = true }
            };
            return View();
        }

        [HttpPost]
        public ActionResult GetTags(string[] tags)
        {
            return View();
        }

        [HttpPost]
        public ActionResult Index(string[] result)
        {
            return View();
        }
    }
}
```

و در این حالت View برنامه به شکل زیر درخواهد آمد:

```
@{
```

```
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
@using (Html.BeginForm())
{

    <input type='checkbox' name='Result' value='value1' />
    <input type='checkbox' name='Result' value='value2' />
    <input type='checkbox' name='Result' value='value3' />
    <input type="submit" value="submit" />
}

@using (Html.BeginForm(actionName: "GetTags", controllerName: "Home"))
{
    @Helpers.CheckBoxList("Tags", (List<SelectListItem>)ViewBag.Tags)
    <input type="submit" value="submit" />
}
```

با توجه به اینکه کدهای Razor قرار گرفته در پوشه خاص app_code در ریشه سایت، به صورت خودکار در حین اجرای برنامه کامپایل می‌شوند، متد Helpers.CheckBoxList در تمام Viewهای برنامه در دسترس خواهد بود. در این متد، یک نام و لیستی از SelectListItemها دریافت می‌گردد. سپس به صورت خودکار یک CheckboxList را تولید خواهد کرد. برای دریافت مقادیر ارسالی آن به سرور هم باید مطابق متد GetTags تعریف شده در کنترلر Home عمل کرد. در اینجا Value عناصر انتخابی به صورت آرایه‌ای از رشته‌ها در دسترس خواهد بود.

روشی جامع‌تر

در آدرس زیر می‌توانید یک HtmlHelper بسیار جامع را جهت تولید CheckBoxList در ASP.NET MVC بیابید. در همان صفحه روش استفاده از آن، به همراه چندین مثال ارائه شده است:

https://github.com/devnoob/MVC3-Html.CheckBoxList-custom-extension

عنوان: RadioButtonList در ASP.NET MVC
نویسنده: وحید نصیری
تاریخ: ۰۹:۲۷:۰۰ ۱۳۹۱/۰۲/۱۲
آدرس: www.dotnettips.info
برچسب‌ها: MVC

برای تهیه یک RadioButtonList نیز می‌توان از همان نکته‌ی CheckBoxList استفاده کرد: نام عناصر radio button اضافه شده به صفحه را یکسان وارد می‌کنیم. به این ترتیب یک گروه تشکیل خواهد شد و زمانیکه اطلاعات این عناصر به سرور ارسال می‌شود، اینبار بجای یک آرایه، تنها مقدار کنترل انتخاب شده، ارسال می‌گردد. یک مثال:
یک پروژه جدید و خالی ASP.NET MVC را آغاز کنید. سپس کنترلر Home و View خالی Index را نیز ایجاد نمائید. محتویات این دو را به نحو زیر تغییر دهید:

```
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
<fieldset>
    <legend>HandleForm1 (Normal)</legend>
    @using (Html.BeginForm(actionName: "HandleForm1", controllerName: "Home"))
    {
        @:your favorite tech: <br />
        @Html.RadioButton(name: "tech", value: ".NET", isChecked: true) @:DOTNET <br />
        @Html.RadioButton(name: "tech", value: "JAVA", isChecked: false) @:JAVA <br />
        @Html.RadioButton(name: "tech", value: "PHP", isChecked: false) @:PHP <br />
        <input type="submit" value="Submit" />
    }
</fieldset>
```

```
using System.Collections.Generic;
using System.Web.Mvc;

namespace MvcApplication23.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult HandleForm1(string tech)
        {
            return RedirectToAction("Index");
        }
    }
}
```

در اینجا سه RadioButton با نامی یکسان در صفحه اضافه شده‌اند. سپس داخل متد HandleForm1 یک breakpoint قرار دهید. اکنون برنامه را اجرا کنید و فرم را به سرور ارسال نمائید. پارامتر tech با value عنصر انتخابی مقدار دهی خواهد شد.

تهیه یک RadioButtonList عمومی

اطلاعات فوق را می‌توان تبدیل به یک HtmlHelper با قابلیت استفاده مجدد نیز نمود:

```
@helper RadioButtonList(string groupName, IEnumerable<System.Web.Mvc.SelectListItem> items)
{
    <div class="RadioButtonList">
        @foreach (var item in items)
        {
            @item.Text
            <input type="radio" name="@groupName"
                   value="@item.Value"
                   @if (item.Selected) { <text>checked="checked"</text> }
               />
            <br />
        }
    </div>
}
```

برای مثال یک فایل را در مسیر app_code\Helpers.cshtml ایجاد کرده و اطلاعات فوق را به آن اضافه نمائید.
اینبار برای استفاده از آن خواهیم داشت:

```
using System.Collections.Generic;
using System.Web.Mvc;

namespace MvcApplication23.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public ActionResult Index()
        {
            ViewBag.Tags = new[]
                           {
                               new SelectListItem { Text = ".NET", Value = "Val1", Selected = true },
                               new SelectListItem { Text = "JAVA", Value = "Val2", Selected = false },
                               new SelectListItem { Text = "PHP", Value = "Val3", Selected = false }
                           };
            return View();
        }

        [HttpPost]
        public ActionResult HandleForm2(string preferredTechnology)
        {
            return RedirectToAction("Index");
        }
    }
}
```

```
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>

<fieldset>
    <legend>HandleForm2 (Helper)</legend>
    @using (Html.BeginForm(actionName: "HandleForm2", controllerName: "Home"))
    {
        @:your favorite tech: <br />
        @Helpers.RadioButtonList("preferredTechnology", (SelectListItem[])ViewBag.Tags)
        <input type="submit" value="Submit" />
    }
</fieldset>
```

متد سفارشی تهیه شده، یک آرایه از SelectListItem ها را دریافت کرده و به صورت خودکار تبدیل به RadioButtonList می‌کند. بر اساس نام آن می‌توان به مقدار انتخاب شده ارسالی به سرور در کنترلر مرتبط، دسترسی یافت.

تهیه یک Templated helper سفارشی

در عمل زمانیکه با مدل‌ها کار می‌کنیم و اطلاعات برنامه قرار است Strongly typed باشند، مرسوم است لیستی از انتخاب‌ها را به صورت یک enum تعریف کنند. برای مثال مدل زیر را به برنامه اضافه کنید:

```
using System.ComponentModel.DataAnnotations;

namespace MvcApplication23.Models
{
    public enum Gender
    {
        [Display(Name = "مرد")]
        Male,
        [Display(Name = "زن")]
        Female,
    }

    public class User
    {
        [ScaffoldColumn(false)]
        public int Id { set; get; }

        [Display(Name = "نام")]
        public string Name { set; get; }

        [Display(Name = "جنسیت")]
        [UIHint("EnumRadioButtonList")]
        public Gender Gender { set; get; }
    }
}
```

قصد داریم یک Templated helper سفارشی را به نام EnumRadioButtonList، ایجاد کنیم تا در زمان فراخوانی متد Html.EditorForModel، به صورت خودکار enum تعریف شده را به صورت یک RadioButtonList نمایش دهد.
برای این منظور فایل جدید Views\Shared\EditorTemplates\EnumRadioButtonList.cshtml را به برنامه اضافه کنید. محتوای آن‌را به نحو زیر تغییر دهید:

```
@using System.ComponentModel.DataAnnotations
@using System.Globalization
@model Enum
@{
    Func<Enum, string> getDescription = enumItem =>
    {
        var type = enumItem.GetType();
        var memInfo = type.GetMember(enumItem.ToString());
        if (memInfo != null && memInfo.Any())
        {
            var attrs = memInfo[0].GetCustomAttributes(typeof(DisplayAttribute), false);
            if (attrs != null && attrs.Any())
                return ((DisplayAttribute)attrs[0]).GetName();
        }
        return enumItem.ToString();
    };

    var listItems = Enum.GetValues(Model.GetType())
                        .OfType<Enum>()
                        .Select(enumItem =>
                                    new SelectListItem()
                                    {
                                        Text = getDescription(enumItem),
                                        Value = enumItem.ToString(),
                                        Selected = enumItem.Equals(Model)
                                    });

    string prefix = ViewData.TemplateInfo.HtmlFieldPrefix;
    ViewData.TemplateInfo.HtmlFieldPrefix = string.Empty;

    int index = 0;
    foreach (var li in listItems)
```

```
    {
        string fieldName = string.Format(CultureInfo.InvariantCulture, "{0}_{1}", prefix, index++);
    <div class="editor-radio">
        @Html.RadioButton(prefix, li.Value, li.Selected, new { @id = fieldName })
        @Html.Label(fieldName, li.Text)
    </div>
    }

    ViewData.TemplateInfo.HtmlFieldPrefix = prefix;
}
```

در اینجا به کمک Reflection به اطلاعات enum دریافتی دسترسی خواهیم داشت. بر این اساس می‌توان نام عناصر آن‌را یافت و تبدیل به یک RadioButtonList کرد. البته کار به همینجا ختم نمی‌شود. در این بین باید دقت داشت که ممکن است از ویژگی Display (مانند مدل نمونه فوق) بر روی تک تک عناصر یک enum نیز استفاده شود. به همین جهت این مورد نیز باید پردازش گردد.

نهایتا برای استفاده از این Templated helper سفارشی، کنترلر و View برنامه را به نحو زیر می‌توان تغییر داد:

```
using System.Collections.Generic;
using System.Web.Mvc;
using MvcApplication23.Models;

namespace MvcApplication23.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public ActionResult Index()
        {
            var user = new User { Id = 1, Name = "name 1", Gender = Gender.Male };
            return View(user);
        }

        [HttpPost]
        public ActionResult HandleForm3(User user)
        {
            return RedirectToAction("Index");
        }
    }
}
```

```
@model MvcApplication23.Models.User
@{
    ViewBag.Title = "Index";
}
<h2>
    Index</h2>
<fieldset>
    <legend>HandleForm3 (EditorForModel)</legend>
    @using (Html.BeginForm(actionName: "HandleForm3", controllerName: "Home"))
    {
        @Html.EditorForModel()
        <input type="submit" value="Submit" />
    }
</fieldset>
```

برای استفاده از یک templated helper سفارشی چندین روش وجود دارد:

الف) همانند مثال فوق از ویژگی UIHint استفاده شود.

ب) نام فایل را به enum.cshtml تغییر دهیم. به این ترتیب از این پس کلیه enumها در صورت استفاده از متد Html.EditorForModel، به صورت خودکار تبدیل به یک RadioButtonList می‌شوند.

ج) متد زیر نیز همین کار را انجام می‌دهد:

```
@Html.EditorFor(model => model.EnumProperty, "EnumRadioButtonList")
```