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

نظرات خوانندگان

نویسنده: Naser Tahery
تاریخ: ۱۳۹۱/۰۱/۱۹ ۱۱:۱۱:۰۲

آموزش ها واقعا عالیه. ممنون از شما
فقط اگه میشه در مورد UpdateModel و TryUpdateModel یکم بیشتر توضیح بدید که کارکردشو چیه؟ برای چه منظوری استفاده میشن؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۹ ۱۱:۵۶:۰۵

یک راه دوم یا راه بکارگیری صریح امکانات هستند. model binder پیش‌فرض در پشت صحنه همین متدها را به صورت خودکار فراخوانی می‌کند. بنابراین ضرورتی ندارد که دستی این متدها را فراخوانی کرد. UpdateModel در صورت بروز مشکلی یک استثناء را صادر می‌کند (InvalidOperationException) اما TryUpdateModel خیر و فرصت خواهید داشت تا با بررسی نتیجه اعتبار سنجی، تصمیم بهتری را اتخاذ کنید.
کلا هرجایی در دات نت Try دیدید، هدفش همین مساله است. مثلا int.Parse داریم و int.TryParse. در مورد سایر نوع‌های عددی هم این دو نوع متد وجود دارند. متد Parse اگر نتواند تبدیل را انجام دهد، یک استثناء را صادر می‌کند، اما متد TryParse سعی خودش را خواهد کرد. اگر نشد، فقط false بر می‌گرداند.

نویسنده: Salehi
تاریخ: ۱۳۹۱/۰۱/۱۹ ۱۲:۱۵:۳۶

من که در مورد mvc مطالعه میکردم به نظرم روش درست و جالب همین" viewmodel بدون وابستگی به model"هست که شما فرمودین . ولی تو این مثال isAdmin من متوجه نشدم که چه مشکلی پیش میاد. به فرض کاربر به طور جعلی به isadmin مقدار بده، مگه متد post برای changePassword با متد post برایِ Edit متفاوت نخواهد بود؟ پس مقدار جعلی isAdmin چه تاثیری میخواد بذاره.
درضمن ممکنه در مورد "صفحه جعلی" توضیح بدید که منظور چی هست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۱۹ ۱۲:۲۳:۱۸

- صفحه تعویض پسورد کاربر، چک باکس IsAdmin را ندارد. اما چون Model کامل کاربر در پشت صحنه آپدیت می‌شود، اگر با جعل صفحه یا درخواست ارسالی به سرور، اطلاعات IsAdmin هم ارسال شود، آنگاه Model binder که با model کامل User کار می‌کند، اطلاعات IsAdmin اضافی را هم دریافت کرده و پردازش می‌کند. بنابراین به نحوی باید سطح کلی در معرض عموم قرارگیری model را کاهش داد که روش‌های آن هم ذکر شد.
- در مورد نحوه انجام این حمله توضیحی نخواهم داد... فقط بدونید که سایت github چند وقت قبل بر همین اساس هک شد و سر و صدای زیادی هم به پا کرد: (^)

نویسنده: amir hosein jelodari
تاریخ: ۱۳۹۱/۰۱/۲۰ ۰۲:۳۸:۰۱

واقعا فوق العاده بود ... هر چی پیشرفته تر میشه جالب تر میشه :دی ... من از ViewModel برای جاهای دیگم استفاده میکنم ... مثه جستجوی پیشرفته(در کل فیلترینگ) که نیاز به چندین پارامتر مختلف داره ...

نویسنده: Davoud Zeini
تاریخ: ۱۳۹۱/۰۱/۲۰ ۱۴:۴۷:۴۸

با سلام و خسته نباشید

کاملاً مشهوده که رفته رفته ViewModel ها بیشتر مورد استقبال قرار می گیرند. اما به نظر من استفاده از ViewModel ها در MVC صحیح نیست. دلیل اصلی اش اینه که ساختار MVC و اهداف اون رو تحت تأثیر قرار می ده. بقیه دلایل هم نهایتاً به همین بر می گردند.

معمولا ظاهر نرم افزارهای کاربردی تحت وب بیشتر از منطق درونی آنها تغییر می کنند. مثلا برای تنوع، یک کار ثابت رو به شکل های جدید انجام می دهند. در واقع برای یک کاری که توسط کنترلر بر روی مدل انجام می گیرد، view های جدید درست می شود. در این حالت برای اینکه بتوانیم تغییرات جدید رو اعمال کنیم، در هر بار علاوه بر اینکه باید ViewModel رو تصحیح کنیم (قطعات جدید اضافه یا حذف کنیم)، باید کنترلرهای مربوطه رو هم هماهنگ کنیم که این باعث کاهش استقلال لایه ها می شود.

دلیل محبوبیت ViewModel ها همانا سادگی آن ها و راحتی طراحی است. یعنی در یک صفحه نسبتاً پیچیده که قسمت های مختلفی دارد، خیلی زود می شه یک ViewModel درست کرد و همه قسمت های صفحه رو توش پوشش داد. اما به نظر من این بعداً مشکل ساز خواهد شد. بهتر است به دور از عجله کمی به ساختار صفحه و راه حل های جایگزین فکر کرد. به عنوان نمونه:

-1 می شه از امکانات خود MVC برای انتقال اطلاعات بین کنترلر و ویو استفاده کرد. می تونیم نمونه ای از مدل رو به ویو بفرستیم و برای موارد اضافی از ViewBag استفاده کنیم. مثلاً می شه اسامی استان ها رو که به یک DropDownList بایند خواهند شد، به شکل زیر بفرستیم:

```
ViewBag.StatesList = new SelectList(_bll.GetStatesList(), "Id", "Title"); //ltr
```

-2 این توضیح رو بدم که MVC ما رو تشویق می کنه که منطق نرم افزارمون رو به ماژول های کوچک تقسیم کنیم و هر ماژول رو به تنهایی مدیریت کنیم. همه جا این تقسیم بندی ها هست. اول کل ساختار نرم افزار رو به سه لایه M و V و C تقسیم می کنه. بعد View ها و کنترلر های هر مدل رو که (که در عالم واقعیت هم مستقل هستند) از هم تفکیک می کنه و یاد میده اینها هر کدوم باید تو فولدر جداگانه ای باشند. Area ها هم در همین راستا هستند. حتی در داخل کنترلر ها، هر کدوم از اکشن ها یک کار واحد کوچک رو انجام می دهند. اما ViewModel این ساختار رو می شکنه و ما رو مجبور می کنه با مفاهیم مرکب سرو کار داشته باشیم. شاید سؤال این باشه که در عمل ما صفحاتی داریم که باید چند ماهیت رو در اونها نمایش بدیم. مثلاً ممکنه در یک صفحه واحد، هم لیست دانش آموز ها رو به همراه امکانات حذف، اضافه و ویرایش داشته باشیم و هم لیست کلاس ها رو. علاوه بر اون یک فرم ارسال نظر در پایین صفحه، یک textbox برای فیلتر کردن و ... داشته باشیم. درسته که می شه با زحمت کمی یک ViewModel حاوی تمام اینها درست کرد و بعد یک View از اون ساخت و در نگاه اول همه چی به خوبی کار کنه. ولی بهتره هر کدوم از اینها مستقلاً پیاده سازی شوند. مثلاً صفحه ای برای دانش آموزان، صفحه ای برای کلاس ها و ... غصه اون صفحه بزرگ رو هم نخورید. می شه هر کدوم از View های مستقل رو در داخل اون نمایش داد و همون ظاهر رو بدست آورد (مثلاً با استفاده از Html.Action یا Ajax). حسن بزرگی که داره اینه که هر کدوم از این قطعات در قسمت های مختلف قابل استفاده می شوند. به این ترتیب می توانیم با جابجایی آن ها در صفحات مختلف، ظاهر نرم افزارمون رو به دلخواه تغییر بدیم بدون اینکه مجبور باشیم تغییری در کنترلر ها و قسمت های دیگه ایجاد کنیم.

در کل ViewModel ها استقلال لایه ها را کاهش می دهند، شدیداً موجب افزایش کدها می شوند، قابلیت گسترش منطقی ندارند و رفته رفته بزرگتر و بزرگتر می شوند، ماژولار نیستند و ...

بنابراین استفاده از ViewModel مکروه بوده و دوری از آن احتیاط واجب است! منتظر فتوای شما در این زمینه هستم. چون به نظر من ارجحه و فصل الخطاب.

با اینکه سعی کردم کوتاه بشه نشد. ببخشید. دوستون دارم. داود زینی

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۰ ۱۶:۲۱:۵۳

خلاصه موردی را که عنوان کردید این است:

اگر یک صفحه داریم که از مثلا 4 ویجت نمایش آب و هوا، نمایش اخبار، نمایش تعداد کاربران حاضر در سایت و آمار سایت و نمایش منوی پویای سایت تشکیل شده، تمام این‌ها رو در یک ViewModel قرار ندیدم که اشتباه است. بله این مورد درست است؛ اما ... به معنای نفی استفاده از ViewModel ها نیست.

هر کدام از ویجت‌ها را می‌شود به Partial viewهای مختلفی برای مثال خرد کرد. اما نهایتا هر کدام از این اجزای کلی سیستم اصلی، نیاز به ViewModel دارند که این مورد، بحث اصلی جاری است. یعنی تفاوت قائل شدن بین domain model و view model.

پوشه‌ی مدلی که در ساختار پروژه پیش فرض ASP.NET MVC قرار داره در واقع امر یک ViewModel است و نه مدل به معنای تعریف مدل‌های domain سیستم چون قرار است خواص آن مستقیما در View مورد استفاده قرار گیرند.

استفاده از ViewModelها کار را اندکی بیشتر می‌کنند، چون نهایتا خواص آن‌ها باید به مدل اصلی نگاشت شوند اما خوشبختانه برای این مورد هم راه حل هست و روش پیشنهادی، استفاده از کتابخانه‌ی سورس بازی است به نام AutpoMapper : (^)

نویسنده: Salehi
تاریخ: ۱۳۹۱/۰۱/۲۰ ۲۳:۳۴:۵۱

یکی از اون مسائلی که در ابتدای بررسی mvc واسه من مساله بود این بود که چطور صفحه ای که از چند بخش تشکیل شده باشه رو با یه مدل باید ساخت .تا اینکه به viewModel رسیدم و به نظرم پاسخگوی مساله بود.
من اینطور فهمیده بودم که viewModel فقط برای strongly type بودن view هستش و هیچ منطقی قرار نیست در اون پیاده بشه. بنابراین اگه چند بخش مختلف مثل اخبار، نفرات حاضر و ... بوسیله اون نمایش داده بشه مشکلی ایجاد نمی کنه. چون ViewModel به فرض فقط لیستی از خبرها رو نگهداری میکنه و پر کردن اون لیست وظیفه اش نیست.
ولی مثل اینکه دوستان نظر دیگه ای دارن.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۱ ۰۰:۲۱:۳۵

در مورد Refactoring فایل‌های View در قسمت بعدی توضیح دادم.

نویسنده: Ariyous Abdiyous
تاریخ: ۱۳۹۱/۰۱/۳۰ ۱۶:۴۲:۲۳

واقعا عالی بود. ممنون بابت مطلبتون

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۲ ۱۷:۱۵

اگر امکانش هست درباره نحوه نامگذاری ViewModel ها و آیا برای هر View باید ما یک ViewModel داشته باشیم یه توضیحی بدبد؟مثل CreateViewModel -EditViewModel باتشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۲ ۱۷:۳۱

به شخصه برای تمام Viewهایی که درست می‌کنم از ViewModel استفاده می‌کنم و نام آن‌ها را هم نام view مورد نظر + کلمه ViewModel درنظر می‌گیرم.
به هیچ عنوان هم از یک مدل (domain model) به عنوان مدل یک View استفاده نمی‌کنم خصوصا به دلایل امنیتی که نباید کل model را در معرض دید قرار داد. برای مثال اگر یک صفحه فقط به نام کاربری و پسورد نیاز دارد، این view به یک viewmodel به نام UserInfoViewModel با دو فیلد یاد شده نیاز دارد و نه بیشتر.
به طور خلاصه برای مثال اگر از EF Code first استفاده می‌کنید، کلاس‌های مدل آن نباید مستقیما در MVC استفاده شوند. این M در MVC به معنای مدلی مخصوص View است و نه کلاس‌های اصلی دومین شما.

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۲ ۱۷:۴۷

حق با شماست ولی آیا به نظر شما این کار تکرار کد نویسیو باعث نمیشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۲ ۱۷:۵۲

تکرار نیست. هر View ساختار مستقل خودش رو داره. به نظر استفاده از یک مدل کلی برای چندین View حجم کدنویسی کمتری داره ولی واقعیت این است که مدام باید یک سری فیلدها را exclude کنید تا به مشکل امنیتی برنخورید.
در کل انجام کار اصولی الزاما به معنای کمتر کدنوشتن نیست. ضمن اینکه کتابخانه auto-mapper کار نگاشت viewmodel به model رو می‌تونه خیلی ساده انجام بده. بنابراین حداقل در این قسمت حجم کدنویسی کمی خواهید داشت.

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۳ ۱:۳۹

آیا برای insert یا update باید از viewmodelهای متفاوتی استفاده کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۳ ۹:۱۱

کاری که من انجام می‌دم، Refactor یک فرم به یک PartialView است و سپس استفاده از این PartialView که با یک ViewModel کار می‌کند، در چندین و چندجا.

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۹:۲۹

یه سوال:
من یه کلاکشنی از مدلم دارم میخام از طریق auto-mapper اینو مپ کنم به viewmodel و تو ویو نمایس بدم میشه راهنماییم کنید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۹:۴۲

کار با auto-mapper نیاز به یک مقاله مجزا دارد که در طی روزهای آتی سعی می‌کنم مطلبی در مورد آن تهیه کنم.

نویسنده: محمد شهریاری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۳:۳

با سلام
در صورتی که برای ایجاد فرم‌ها از خاصیت Scaffold template استفاده کنیم دستور BeginForm مقادیر controller و Action مقداری ندارند . هنگامی که فرم submit می‌شود از بر چه اساسی Action کنترلر مربوطه اجرا می‌شود ؟
کنترلر براساس مدل تعریف شده در ابتدای View انتخاب می‌شود ؟
Action بر اساس نام فایل ؟
2- در تمامی مثالها از برای نمایش Viewها از Modelهای ساده استفاده شده بود . در صورتی که بخواهم در یک view از Modeli استفاده کنم که یک Property آن لیستی از یک Model دیگر باشد به چه صورت است ؟
آیا باید viewها را همیشه به Modelهای جدا تفکیک کنیم ؟
3- هنگام سفارشی سازی model Binder در مثال شما مربوط به ورودی تاریخ شمسی و تبدیل آن به تاریخ میلادی کدی که در application_start اضافه شد مشخص می‌کرد در هر model که type فقط از نوع Datetime باشد اعمال گردد ؟
با توجه به مثال در صورتی که بخواهم فقط بر روی Create اعمال گردد در صورتی که معرفی binder را از Application_start حذف کنم بر روی تمامی Property ها اعمال میگردد . در صورتی هم که حذف نکنم باز هم بر روی تمامی Action ها اعمال می‌شود .

با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۱۱ ۱۴:۹

- از مقادیر پیش فرض استفاده می‌شود. زمانیکه شما می‌نویسید return view و پارامتری را صریحا مشخص نمی‌کنید، از نام متد فراخوان به عنوان view متناظر استفاده می‌گردد. در اینجا هم به همین نحو است. ارسال به کنترلر متناظر با View جاری و متد پیش فرضی که در routing تعریف شده است، صورت می‌گیرد. البته می‌شود تمام این‌ها را هم صریحا تعریف کرد.

- در این حالت می‌شود Model.property1.property2 الی آخر و کار می‌کند. فقط ابزارهای Scaffolding از این نوع خواص پشتیبانی نمی‌کنند، اما به این معنا نیست که کل فریم ورک مشکلی با آن دارد.
- بله. مثال من یک مثال عمومی بود. شما بر اساس contextهای دریافتی در حین پیاده سازی یک binder می‌تونید اون رو تا حد امکان سفارشی سازی کنید که مثلا برای یک کنترلر خاص یا یک مدل خاص فقط عمل کند.

نویسنده: hasani
تاریخ: ۱۳۹۱/۰۵/۱۶ ۲۱:۴۰

با سلام و عرض خسته نباشید خدمت مهندس نصیری
در قسمت سفارشی سازی model binder پیش فرض ASP.NET MVC و مثالی که برای تاریخ شمسی ارائه شد، اگر فیلد تاریخ Nullable باشد، با خطا موجه میشویم که برای رفع آن خط زیر هم باید به Application_Start اضافه شود:

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

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۱/۰۵/۲۵ ۲۲:۴۸

شما اشاره به چهار ویژگی یک ViewModel خوب کردید. ولی دوتا از این ویژگی‌هارو در مثالتون رعایت نکردین.
اول، گفتین هر ViewModel از یک View استفاده میکنه. که در مثالتون از خواص اضافی استفاده کردین. که شاید از دو یا چند View استفاده کرده است.
دوم، در متن گفتین در ViewModel منطقی درکار نیست. همچنین گفتین ViewModel از کنترلر ساختارشو نمیگیره و از View میگیره.
پس مثلا در مثالتون خواص FullName در ViewModel که Name+Family در View هست تشکیل شده چی هست؟ این منطق نیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۵ ۲۳:۱۴

- MyViewModel اول مطرح شده: عنوان شد که این مورد نباید استفاده شود.
MyViewModel دوم مطرح شده: این صحیح است و فیلدهایی که در آن مشاهده می‌کنید بر اساس ساختار View (ایی فرضی جهت توضیح بحث) تعریف شدن و تناظر یک به یک وجود دارد.
- بله ViewModel ساختار فیلدهای تشکیل دهنده‌اش رو (یعنی نحوه تعریف خواص/طراحی خودش رو) از کنترلر نمی‌گیره (بر مبنای ساختار View متناظر تعریف میشه). اما در کنترلر هست که مقدار دهی خواهد شد و نهایتا در اختیار View قرار می‌گیره. بنابراین فرق است بین طراحی ساختار یک کلاس بر اساس Viewایی متناظر و در ادامه نحوه مقدار دهی آن در یک کنترلر و نهایتا ارسال آن به View جهت استفاده. وجود منطق در اینجا یعنی پیاده سازی مستقیم؛ نه اینکه مجاز نیستیم یک نام ترکیبی بر اساس نیاز View تعریف/طراحی کنیم.
همچنین وجود فیلدهای محاسباتی خیلی جزئی با پیاده سازی اندک هیچ ایرادی در ViewModel ندارد و بسیار هم مورد استفاده است. مثلا جمع زدن نام‌ها؛ تبدیل تاریخ، محاسبه درصد مالیات و موارد جزئی از این دست که مستقیما در View صرفا جهت نمایش استفاده می‌شوند. منطق‌ها و پیاده سازی‌های بیش از یک سطر رو به لایه سرویس منتقل، در کنترلر فراخوانی و سپس در وهله سازی ViewModel ازش استفاده کنید.

نویسنده: امیر
تاریخ: ۱۳۹۱/۰۵/۲۸ ۲۳:۵۲

سلام
طبق گفته شما تمامی propertyهای موجود در model در صورت هم نام بودن با کنترل‌های موجود در view مقداردهی می‌شوند
حالا اگر بخوایم یک property در model را با یک کنترل در view مقداردهی کنیم که هم نام نیستند چه باید کرد
بعدش اینکه در razor چیزی به اسم masterpage همچنان وجود داره یا خیر؟ اگر وجود نداره راه حل جایگزین چیست؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۸ ۲۳:۵۶

- مانند قسمت 10 این سری از FormCollection استفاده کنید.
- در قسمت 14 به این موضوع پرداخته شده.

نویسنده: محسن
تاریخ: ۱۳۹۱/۱۱/۲۳ ۱۴:۱۸

من یک مثال جهت استفاده از model و viewmodel می‌خواستم که چگونه قابل اجرا است

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۳ ۱۵:۶

قسمتی در سایت هست تحت عنوان نیازمندی‌های آموزشی . درخواست خودتون رو در آنجا مطرح کنید. اگر یکی از نویسندگان سایت راغب بود، وقت کرد، شاید در این مورد یا موارد پیشنهادی دیگر، مطلبی تهیه و ارسال شد به شکل یک مقاله جدید در سایت. شما هم می‌تونید به این جمع اضافه بشید. خارج از این، مثال و مطلب اضافه‌تری در دسترس من و ما نیست.

نویسنده: پیمان مهربانی
تاریخ: ۱۳۹۱/۱۲/۱۵ ۱۸:۵۵

درود
پس از خواندن تمامی مطالب سایت ، تا اینجا به این نتیجه رسیدم که آقای نصیری به استفاده از ViewModel به عنوان پلی میان View و Model تاکید دارن.
استفاده از ViewModelها باعث افزایش امنیت در مقابل بعضی از حملات و افزایش Performance با دوری از SELECT * میشه.
اما مطلبی که هنوز برای من روشن نشده رهایی از Duplicate شدن منطق اعتبارسنجی مدل هاست که به صورت Attribute در ViewModelها مشخص می‌شوند.و این تعریف Validation Ruleها سیستم رو دچار Duplicated Business می‌کنه.آیا راهی هست که تمامی Validation Ruleها در Model انجام شه و به صورت AutoMap بر روی ViewModelهای متناظر اعمال شود؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۵ ۲۱:۶

روشی در اینجا با استفاده از AutoMapper برای کپی کردن ویژگی‌ها از مدل‌ها به ViewModelها پیشنهاد شده.
+ امکان خارج کردن تعاریف ویژگی‌ها از یک کلاس توسط MetaDataType نیز پیش بینی شده. بین این صورت می‌شود تعاریف ویژگی‌های تعریف شده را به چند کلاس مختلف هم اعمال کرد.

نویسنده: نوید
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۱:۴۲

با سلام
هنگامی که از ViewModelها برای ساختن یک view که ترکیبی از چند DomainModel هست استفاده میکنیم، چگونه باید بین فیلدهای درون یک ViewModel با فیلدهای DomainModelها تناظر یک به یک بر قرار کرد.
در چند مثال دیدم که از یک تابع جداگانه برای نگاشت بین اطلاعات این کلاس‌ها استفاده میکردند ولی نحوه این نگاشت رو توضیح نداده بودند (مانند http://blogs.msdn.com/b/simonince/archive/2010/01/26/view-models-in-asp-net-mvc.aspx)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۶ ۱۲:۶

برچسب AutoMapper را در سایت دنبال کنید.

نویسنده: ahmadb7
تاریخ: ۱۳۹۲/۰۳/۱۱ ۲۲:۳

در قسمت آپلود فایل به سرور

```
public ActionResult Upload(System.Web.HttpPostedFileBase file)
```

مقدار پارامتر file برابر با null می‌باشد و خطا ایجاد می‌شود چرا؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۱ ۲۲:۱۱

بررسی کنید که آیا مانند مثال آزمایش شده فوق، enctype فرم ذکر شده یا خیر، همچنین input file داخل فرم قرار دارد یا خیر.

نویسنده: Leila_gh
تاریخ: ۱۳۹۲/۰۴/۳۱ ۷:۱۷

با سلام و تشکر
مشکل من در قسمت آپلود فایل به سرور (مقدار پارامتر file برابر با null بود و خطا ایجاد می‌شد) از طریق این پست حل شد .

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۴/۳۱ ۹:۳۶

در مثالی که من در این قسمت مطرح کردم، نام کنترل ارسال فایل photo است، اما نام پارامتر اکشن متد متناظر با آن file قرار داده شده که باید یکی شوند. به این ترتیب model binder اینبار می‌داند که اطلاعات دریافتی را باید به چه پارامتری انتساب دهد تا مقدار آن دیگر نال نباشد.

نویسنده: محمد آزاد
تاریخ: ۱۳۹۲/۰۵/۰۱ ۱۳:۱۱

در قسمت آخر مطلبتون یک ModelBinder سفارشی برای DateTime نوشته شده که روش خوب و مناسبی هست برای جلوگیری از تکرار کد اضافی. اما یک مشکلی که من دارم اینه این روش شما برای گرفتن اطلاعات از کلاینت درست عمل میکنه اما مشکل اینجاست اگر من بخوام این فیلد تاریخ رو به کاربر نشون بدم چه کاری باید انجام بدم؟یعنی من این فیلد رو از دیتابیس خوندم چطور میشه اونو به مقدار شمسی تبدیل کنم و به کاربر نشون بدم ! چون مقدار تاریخ شمسی برای یک DateTime قابل قبول نیست(راهی ک من برای حل مشکل داشتم این بود که یک فیلد String در viewmodelقرار میدادم و مقادیر اونو وقتی اطلاعات از سمت کاربر میومد به مقدار میلادی تبدیل می‌کردم و در فیلد مربوطه قرار میدادم و وقتی اطلاعات رو از دیتابیس لود میکردم مقدار فیلد تاریخ رو به شمسی تبدیل می‌کردم و در فیلد string میذاشتم.)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۰۱ ۱۳:۵۸

- روش شما هم خوبه.
- یک روش متداول دیگر، ذخیره سازی تاریخ شمسی و میلادی با هم هست در یک جدول. این روش سازگاری بهتری با کوئری‌های datetime توکار بانک‌های اطلاعاتی داره. اما ... این کوئری‌ها یکجا مشکل پیدا می‌کنند و آن هم گروه بندی بر اساس ماه‌های شمسی است که تطابقی با ماه‌های میلادی ندارند و اگر قرار باشد برای آن کوئری ویژه‌ای بر اساس datetime میلادی نوشت، به کوئری بسیار پیچیده و کندی خواهیم رسید. اما گروه بندی بر اساس ماه‌های مقادیر فیلد شمسی ذخیره شده بسیار سریع است (مثلا تهیه گزارش لیست خلاصه جمع حقوقی تمامی کارکنان در یکسال، گروه بندی شده بر اساس ماه‌های شمسی؛ ... آن‌هم فقط با نوشتن یک کوئری ساده و نه بیشتر).
- روش دیگر نوشتن یک TextBoxFor سفارشی است که تاریخ میلادی دریافت کند و شمسی نمایش دهد. بعد از ترکیب Model binder فوق استفاده شود برای دریافت مقدار نهایی آن.

نویسنده: لیلا
تاریخ: ۱۳۹۲/۰۹/۱۰ ۹:۲

با تشکر
در هنگام create مقدار ModelState.IsValid ، برابر با false است ایا راهکار این لینک ، راهکار مناسبی است؟

```
ModelState.Remove("Id");
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۱۰ ۹:۱۳

خیر. باید بررسی کنید که علت آن چه چیزی بوده:

```
string messages = string.Join("; ", ModelState.Values
                                       .SelectMany(x => x.Errors)
                                       .Select(x => x.ErrorMessage));
```

نویسنده: مصطفی
تاریخ: ۱۳۹۲/۱۰/۲۱ ۱۲:۱۲

سلام
دوستان وقتی روی لینک details کلیک می‌کنم این ارور رو میده:

The model item passed into the dictionary is of type 'mvc1.Models.User', but this dictionary requires a model item of type 'System.Collections.Generic.IEnumerable`1[mvc1.Models.User]'. به نظر شما از چی میتونه باشه
با تشکر فراوان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۱ ۱۳:۳۲

- خروجی اکشن متد Details یک شیء User هست در اینجا. اما شما در View خودتان بجای شیء User، یک IEnumerable از آن‌را به عنوان نوع مدل تعریف کرده‌اید.
+ دریافت تمام مثال‌های MVC این سری: MVC_Samples

نویسنده: منصور
تاریخ: ۱۳۹۲/۱۱/۰۵ ۱۲:۹

سلام؛ ببخشید این ViewModel کجا تعریف میشن. مثلا اگر یک مدل اطلاعات کاربران داریم به سه تا فیلد نام و رمز عبور و IsAdmin. این ViewModel ی که شامل فقط فیلدهای نام و پسورد هست کجا و چطور تعریف میشن؟ ممنونم.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۰۵ ۱۳:۲۵

هرجا دوست داشتی. خواستی براش یک DLL جدا درست کن . خواستی داخل پوشه مدل‌های همین پروژه قرارش بده. استاندارد خاصی نداره.

نویسنده: kia
تاریخ: ۱۳۹۲/۱۱/۰۵ ۱۵:۵۵

با سلام؛ چند روز پیش متوجه وجود یک مشکل در فرم هایی که مقادیرشون به وسیله Ajax به سرور ارسال می‌شوند توی IE در

حالت Compatibility View شدم.

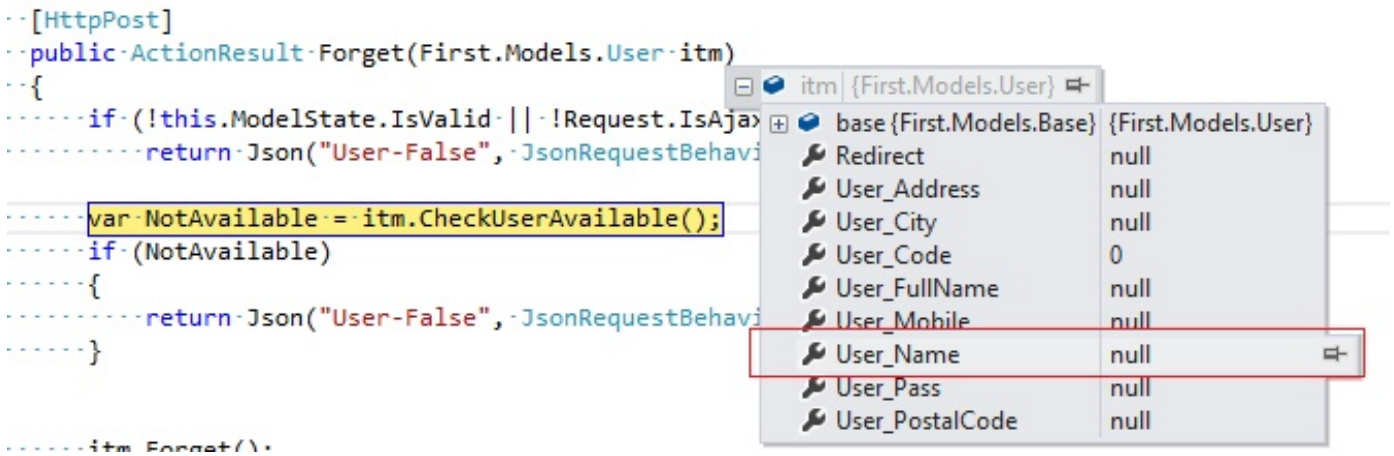
شرح مسئله :

View من به شکل زیره :

```
<% using (Html.BeginForm("Forget", "Account", FormMethod.Post, new { encType = "multipart/form-data",
id = "forgetForm", name = "forgetForm" }))
                  { %>
                <%: Html.AntiForgeryToken() %>
                <%: Html.ValidationSummary(true) %>

                                        <input id="User_Name" name="User_Name" type="text" data-original-
title="لطفاً نام کاربری انتخابی خود را وارد کنید" data-toggle="tooltip" data-placement="top">
                                        <button id="submitForgetForm" type="button"><i></i>  بازیابی کلمه
عبور   </button>
                <% } %>
```

اما مشکل اینجاست که این صفحه توی مرورگر موزیلا و کروم به خوبی کار می‌کنه اما توی اینترنت اکسپلورر در حالت Compatibility View وقتی تابع Forget() اجرا میشه ، مقدار itm.User_Name نال میشه.(تصویر زیر)



کجای کار اشتباه است ؟ **با تشکر**

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۰۵ ۱۷:۸

- id کنترل دریافت اطلاعات شما یک txt اولش اضافه داره نسبت به نام خاصیتی که تعریف کردید. model binder این‌ها رو به هم map نمی‌کنه. ضمنا اگر فرم شما اطلاعاتی رو آپلود نمی‌کنه، encType اون رو حذف کنید.
- دیباگ کنید مرورگر چه اطلاعاتی رو ارسال می‌کنه. از فیدلر هم می‌تونید برای دیباگ IE استفاده کنید.
- تنظیم jQuery.ajaxSettings.traditional = true رو هم تست کنید.
- ذکر contentType صحیح الزامی است.

نویسنده: ح مرداد
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱:۸

سلام،
چند عدد سوال داشتم:

-1 از آنجایی که بنده در مورد استفاده کمتر از منابع سرور و ... خیلی حساسم و در پی یافتن بهینه‌ترین روش کدینگ هستم ، سوالی برام پیش اومده :
آیا تعریف یک پراپرتی با دسترسی private از نوع EntityFrameWorkContext در سطح کلاس کنترلر (یا سطح کلاس سرویس یا کلا در سطح یک کلاس) و استفاده از آن در متدهای کلاس و استفاده نکردن از using در داخل متدها از نظر حرفه ای درست می‌باشد ؟
(این روش رو در چند جا مشاهده کردم و شک کردم که نکنه روش بنده که همیشه using می‌زنم بهینه نیست....)
{بهترین روش چیه ؟}

-2 اگر از استفاده غیرضروری از منابع سرور **صرف نظر کنیم** ؛ اگر ما ViewModel استفاده نکنیم و درون اکشن‌های ویرایش مثلا اینجوری کد بزنیم :

```
public ActionResult Edit(Member member)
{
var updatedItem = db.Members.FirstOrDefault(c => c.id == 1);

updatedItem.Name = member.Name;
updatedItem.Family = member.Family;

db.saveChanges();

return View();

}
```

آیا به دلیل استفاده نکردن از پراپرتی‌های غیر ضروری ، مشکل امنیتی برطرف میشه ؟
(بدین صورت اگر کاربر شیطونی کنه و مثلا فیلدای IsAdmin رو دستی بسازه و .... ازش استفاده نمیشه و مشکلی پیش نمیاد)
(نهایتا Model.IsValid هم می‌تونیم در اینجا استفاده کنیم)

مسلما موقع ثبت هم مقدار پروپرتی‌های حساس رو خودمون دستی پر می‌کنیم و اصلا کاری به ورودی دریافتی اکشن نخواهیم داشت.

```
public ActionResult Create(Member member)
{

If (Model.IsValid)
{

   db.Members.AddObject(new Member{ Name = member.Name , Family = member.Family , IsAdmin = False});
   db.saveChanges();

// ...

}

return View();

}
```

-3 سفارشی سازی پیام‌های خطای اعتبار سنجی فرم رو هم من که تست کردم ، همش انگلیسی پیام میده!
و متن فارسی منو نادیده می‌گیره ...
یک توضیح بیشتر اگر مرحمت کنین ، ممنون میشم.

با تشکر

نویسنده: ح مرداف
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱:۱۲

با سلام
آیا منظور شما اینه که تاریخ شمسی رو بصورت string توی دیتابیس بریزیم ؟
PersianDate : VARCHAR <<
Date : DATE <<

بدین صورت جهت استخراج تاریخ‌های شمسی و زدن کوئری به یک عدد پروسیجر بزرگ نیاز خواهیم داشت (مثلا بر اساس "/" اسپلیت کنه و بعد هر بخش رو int کنه و ....)
آیا چاره دیگری نداریم ؟ :(

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱:۲۰

- به صورت string شمسی «هم» ذخیره کنید. مابقی آن تابع substring است. نیازی به split ندارد. فقط ماه و روز تک رقمی باید یک صفر قبلش وارد شود، تا مرتب سازی درست انجام شود؛ یعنی بجای 1/1/92 باید 01/01/92 ذخیره شود (این روش از دوران فاکس‌پرو تحت داس مرسوم بوده) و فقط برای گروه بندی سریع بر اساس ماه‌های شمسی لازم است. برای سایر حالات DateTime میلادی کافی است.
- البته روش‌هایی مانند « افزودن یک DataType جدید برای نگه‌داری تاریخ خورشیدی » نیز هستند.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱:۳۲

- روش بهینه، استفاده از یک Context در طول درخواست است. در قسمت 12 سری EF به آن پرداخته شده. پیشنیاز آن مطالعه کامل مباحث IoC و تزریق وابستگی‌ها است.
- ViewModel یک روش است. روش‌های لیست سیاه و سفید هم هستند. این موارد هم بیشتر از این جهت معرفی شده‌اند چون با استفاده از ابزارهایی مانند AutoMapper می‌شود خواص مدل‌ها را خیلی سریع و بدون نوشتن تک تک آن‌ها به یکدیگر نگاشت کرد و یا متد توکار TryUpdateModel سعی می‌کند کل مدل را بر اساس اطلاعات دریافتی از کاربر، به روز رسانی کند.
- به اعتبارسنجی یک قسمت مجزا اختصاص داده شده‌است. جزئیات روش کار خودتان را با آن مقایسه کنید.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۰۹/۲۴ ۱:۴۸

با تشکر از مطلب مفیدتان
شما گفتید در mvc به جای viewstateها از برگشت دادن یک کلاس استفاده می‌کنیم ولی من الان اگر در اکشن نوع httppost فقط بنویسم

```
return view();
```

باز هم داده‌ها داخل فیلدها قرار دارن و هیچ فرقی با برگشت زدن مدل نداره

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۲۴ ۳:۶

- روش توصیه شده برای کار با فرم‌ها در حالت post back، استفاده از الگوی PRG است و بازگشت مستقیم View در این حالت توصیه نمی‌شود.
- متدهای ویژه‌ی **For** ، TextBox Editor و امثال آن، در Viewها و حالت Post (نه حالت Get)، مقدار خودشان را از ModelState کنترلر جاری دریافت می‌کنند (همان مقادیر و خواصی که در حالت Post به سرور ارسال شده‌اند) و اصلا کاری به model بازگشتی ندارند.
- اگر در این بین نیاز به تغییری در مقدار خاصیتی باشد، نیاز است حتما model بازگشت داده شود؛ ضمنا نکته‌ی ویژه‌ای هم برای به روز رسانی مقدار تغییر کرده وجود دارد (به علت استفاده از ModelState، تغییرات بر روی مدل دریافتی تا پیش از Remove

اعمال نمی‌شوند) :

```
[HttpPost]
public ActionResult Create(User user)
{
   ModelState.Remove("Name"); // به روز رسانی مقدار یک خاصیت در حالت ویژه ارسال به سرور
   user.Name = "new name";
   return View(user);
}
```

- اگر در ViewModel بازگشتی به View، نیاز است مثلا drop down ایی پر شود، مقادیر آن لیست یا لیست‌ها به همراه HTTP Post معمولی به سرور ارسال نمی‌شوند و نیاز است توسط return View model مقدار دهی مجدد شوند.
- اگر از متدهای ویژه‌ی For دار استفاده نکنید، باز هم نیاز است return View model انجام شود:

```
<input type="text" name="User.Email" id="User_Email" value="@Model.User.Email" />
```

در کل در حین آموزش مطلبی، ارائه عمومی‌ترین حالت ممکن، که با اکثر سناریوها کار می‌کند توصیه می‌شود.