عنوان: **Best Practice هایی برای ASP.NET MVC - قسمت اول**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۱۲ ۱۸:۵
آدرس: www.dotnettips.info
برچسب‌ها: ASP.Net, MVC, Tips

در سایت جاری مطالب زیادی درباره ASP.NET MVC نوشته شده است. این مطلب و قسمت بعدی آن مروری خواهد داشت بر Best Practiceها در ASP.NET MVC.

استفاده از NuGet Package Manager برای مدیریت وابستگی‌ها

درباره اهمیت NuGet برای مصرف کنندگان قبلا این مطلب نوشته شده است.
بجای صرف وقت برای اینکه بررسی کنیم آیا این نسخه‌ی جدید کتابخانه‌ی X یا اسکریپت jQuery آمده است یا خیر، می‌توان این وظیفه را به NuGet سپرد. علاوه بر این NuGet مزیت‌های دیگری هم دارد؛ مثلا تیم‌های برنامه نویسی می‌توانند کتاب خانه‌های مشترک خودشان را در مخزن‌های سفارشی NuGet قرار دهند و توزیع و Versioning آن‌را به NuGet بسپارند.

تکیه بر Abstraction (انتزاع)

Abstraction در طراحی سیستم‌ها منجر به تولید نرم افزار هایی Loosely coupled با قابلیت نگهداری بالا و همچنین فراهم شدن زمینه برای نوشتن Unit Test می‌شود.
اگر به مطالب قبلی وب سایت برگردیم در مطلب چرا ASP.NET MVC گفته شد که :
2) دستیابی به کنترل بیشتر بر روی اجزای فریم ورک :
در طراحی ASP.NET MVC همه‌جا interfaceها 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 interfaceها یک مزیت دیگر را هم به همراه دارد و آن هم ساده سازی mocking (تقلید) آن‌ها است جهت ساده سازی نوشتن آزمون‌های واحد.

از کلمه‌ی کلیدی New استفاده نکنید

هر جا ممکن است کار وهله سازی اشیاء را به لایه و حتی Framework دیگری بسپارید. هر زمان اشیاء نرم افزار خودتان را با کلمه‌ی new وهله سازی می‌کنید اصل Abstraction را فراموش کرده اید. هر زمان اشیاء نرم افزار را مستقیم وهله سازی می‌کنید در نظر داشته باشید می‌توانید آنها را به صورت وابستگی تزریق کنید.
در همین رابطه مطالب زیر را از دست ندهید :
تزریق وابستگی (dependency injection) به زبان ساده
تزریق وابستگی (Dependency Injection) و توسعه پذیری

از HttpContext مستقیما استفاده نکنید (از HttpContextBase استفاده کنید)

از .NET 4 به بعد فضای نامی تعریف شده که در بر دارنده‌ی کلاس‌های انتزاعی (Abstraction) خیلی از قسمت‌های اصلی ASP.NET است. یکی از مواردی که در توسعه‌ی ASP.NET معمولا زیاد استفاده می‌شود، شیء HttpContext است . استفاده از HttpContextBase را به استفاده از HttpContext ترجیح دهید. اهمیت این موضوع در راستای اهمیت انتزاع (Abstraction) می‌باشد.

از "رشته‌های جادویی" اجتناب کنید

استفاده از رشته‌های جادویی در خیلی از جاها کار را ساده می‌کند؛ بعضی وقت‌ها هم به آنها نیاز است اما مشکلات زیادی دارند :

رشته‌ها معنای باطنی ندارند (مثلا : دشوار است که از روی نام یک ID مشخص کنم این ID چطور به ID دیگری مرتبط است و یا اصلا ربط دارد یا خیر)
با اشتباهات املایی یا عدم رعایت حروف بزرگ و کوچک ایجاد مشکل می‌کنند.
به Refactoring واکنش خوبی نشان نمی‌دهند. (برای درک بهتر این مطلب را بخوانید.)
برای درک بهتر 2، یک مثال بررسی می‌شود؛ اولی از رشته‌های جادویی برای دسترسی به داده در ViewData استفاده می‌کند و دومی refactor شده‌ی مثال اول است که از strongly type مدل برای دسترسی به همان داده استفاده می‌کند.

```
<p>
    <label for="FirstName">First Name:</label>
    <span id="FirstName">@ViewData["FirstName"]</span>
</p>
```

مثال دوم :

```
<p>
    <label for="FirstName">First Name:</label>
    <span id="FirstName">@Model.FirstName</span>
</p>
```

مثلا مثال دوم مشکلات رشته‌های جادویی را ندارد.
در رابطه با Magic strings این مطلب را مطالعه بفرمایید.

از نوشتن HTML در کدهای "Backend" خودداری کنید

با توجه به اصل جداسازی وابستگی‌ها (Separation of Concerns) وظیفه‌ی کنترلرها و دیگر کدهای backend رندر کردن HTML نیست. (ساده سازی کنترلر ها) البته در نظر داشته باشید که قطعا تولید HTML در متدهای کمکی کلاس هایی که "تنها" وظیفه‌ی آنها کمک به Viewها جهت تولید کد هست ایرادی ندارد. این کلاس‌ها بخشی از View در نظر گرفته می‌شوند نه کدهای "backend".

در Viewها "Business logic" انجام ندهید

معکوس بند قبلی هم کاملا صدق می‌کند ، منظور این است که Viewها تا جایی که ممکن است باید حاوی کمترین Business logic ممکن باشند. در واقع تمرکز Viewها باید استفاده و نحوه‌ی نمایش داده ای که برای آن‌ها فراهم شده باشد نه انجام عملیات روی آن.

استفاده از Presentation Model را به استفاده مستقیم از Business Objectها ترجیح دهید

در مطالب مختلف وب سایت اشاره به اهمین ViewModelها شده است. برای اطلاعات بیشتر بند ج آموزش 11 از سری آموزش‌های ASP.NET MVC را مطالعه بفرمایید.

Ifهای شرطی را در Viewها را در متدهای کمکی کپسوله کنید

استفاده از شرطها در View کار توسعه دهنده را برای یک سری اعمال ساده می‌کند اما می‌تواند باعث کمی کثیف کاری هم شود.
مثلا:

```
@if(Model.IsAnonymousUser) {
    <img src="@Url.Content("~/content/images/anonymous.jpg")" />
} else if(Model.IsAdministrator) {
    <img src="@Url.Content("~/content/images/administrator.jpg")" />
```

```
} else if(Model.Membership == Membership.Standard) {
    <img src="@Url.Content("~/content/images/member.jpg")" />
} else if(Model.Membership == Membership.Preferred) {
    <img src="@Url.Content("~/content/images/preferred_member.jpg")" />
}
```

می‌توان این کد که تا حدودی شامل منطق تجاری هم هست را در یک متد کمکی کپسوله کرد :

```
public static string UserAvatar(this HtmlHelper<User> helper)
{
    var user = helper.ViewData.Model;
    string avatarFilename = "anonymous.jpg";
    if (user.IsAnonymousUser)
    {
        avatarFilename = "anonymous.jpg";
    }
    else if (user.IsAdministrator)
    {
        avatarFilename = "administrator.jpg";
    }
    else if (user.Membership == Membership.Standard)
    {
        avatarFilename = "member.jpg";
    }
    else if (user.Membership == Membership.Preferred)
    {
        avatarFilename = "preferred_member.jpg";
    }
    var urlHelper = new UrlHelper(helper.ViewContext.RequestContext);
    var contentPath = string.Format("~/content/images/{0}", avatarFilename)
    string imageUrl = urlHelper.Content(contentPath);
    return string.Format("<img src='{0}' />", imageUrl);
}
```

اکنون برای نمایش آواتار کاربر به سادگی می‌توان نوشت :

```
@Html.UserAvatar()
```

به این ترتیب کد ما تمیزتر شده ، قابلیت نگهداری آن بالاتر رفته ، منطق تجاری یک بار و در یک قسمت نوشته شده از این کد در جاهای مختلف سایت می‌توان استفاده کرد و اگر لازم به تغییر باشد با تغییر در یک قسمت همه جا اعمال می‌شود.

منتظر قسمت بعدی باشید.

نظرات خوانندگان

نویسنده: محسن.د
تاریخ: ۱۳۹۱/۰۸/۱۲ ۱۹:۴۵

نکات جالبی بود .
یک نکته که به شخصه اون رو تجربه کرده ام (بخصوص در مورد استفاده از best practiceها) وضعیتی است که اصطلاحاً بهش overkilling میگن (اگر اشتباه نکنم) ، یعنی بعضی وقتا دیگه زیاده روی میشه . مثلا در به کاربردن اینترفیس‌ها و یا loose coupling .
برای مثال در مورد تشخیص زمان استفاده از اینترفیس در این کتاب ، یکی از بهترین راهکارها استفاده از آزمون واحد معرفی شده .
بسیار عالی میشه در صورت امکان در این مورد هم نکاتی را ذکر کنید .

نویسنده: افشار محبی
تاریخ: ۱۳۹۱/۰۸/۲۴ ۸:۲۶

خیلی مفید و عملی بود. ممنون.

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۱/۰۸/۲۴ ۱۴:۱۹

عالی بود ممنون،
بهتر نیست به جای این if و elseها از Dictionary استفاده کنیم؟

نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۸/۲۴ ۱۴:۲۰

خواهش می‌کنم.
چرا همینطوره.
کدها همه مثال هستند و هدف این مطلب چیز دیگری بوده.