عنوان: **MVC Scaffolding #3**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۰۴ ۸:۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, MVC, MVC Scaffolding

شاید کیفیت کدهای تولیدی یا کدهای View حاصل از MVC Scaffolding مورد تائید شما نباشد. در این قسمت به نحوه تغییر و سفارشی سازی این موارد خواهیم پرداخت.

آشنایی با ساختار اصلی MVC Scaffolding

پس از نصب MVC Scaffolding از طریق NuGet به پوشه Packages مراجعه نمائید. در اینجا پوشه‌های MvcScaffolding، T4Scaffolding و T4Scaffolding.Core ساختار اصلی این بسته را تشکیل می‌دهند. برای نمونه اگر پوشه T4Scaffolding\tools را باز کنیم، شاهد تعدادی فایل ps1 خواهیم بود که همان فایل‌های پاورشل هستند. مطابق طراحی NuGet، همواره فایلی با نام init.ps1 در ابتدا اجرا خواهد شد. همچنین در اینجا پوشه‌های T4Scaffolding\tools\EFRepository و T4Scaffolding\tools\EFDbContext نیز قرار دارند که حاوی قالب‌های اولیه کدهای مرتبط با الگوی مخزن و DbContext تولیدی می‌باشند.
در پوشه MvcScaffolding\tools، ساختار قالب‌های پیش فرض تولید Viewها و کنترلرهای تولیدی قرار دارند. در اینجا به ازای هر مورد، دو نگارش vb و cs قابل مشاهده است.

سفارشی سازی قالب‌های پیش فرض Viewهای MVC Scaffolding

برای سفارشی سازی قالب‌های پیش فرض از دستور کلی زیر استفاده می‌شود:

```
Scaffold CustomTemplate Name Template
```

مانند دستور زیر:

```
Scaffold CustomTemplate View Index
```

در اینجا View نام یک Scaffolder است و Index نام قالبی در آن.
اگر دستور فوق را اجرا کنیم، فایل جدیدی به نام CodeTemplates\Scaffolders\MvcScaffolding.RazorView\Index.cs.t4 به پروژه جاری اضافه می‌شود. از این پس کلیه فرامین اجرایی، از نسخه محلی فوق بجای نمونه‌های پیش فرض استفاده خواهند کرد.
در ادامه قصد داریم اندکی این قالب پیش فرض را جهت اعمال ویژگی DisplayName به هدر جدول تولیدی نمایش اطلاعات Tasks تغییر دهیم. در کلاس Task، خاصیت زمان موعود با ویژگی DisplayName مزین شده است. این نام نمایشی حین تولید فرم‌های ثبت و ویرایش اطلاعات بکار گرفته می‌شود، اما در زمان تولید جدول اطلاعات ثبت شده، به هدر جدول اعمال نمی‌گردد.

```
[DisplayName("Due Date")]
public DateTime? DueDate { set; get; }
```

برای تغییر و بهبود این مساله، فایل Index.cs.t4 را که پیشتر به پروژه اضافه کردیم باز کنید. کلاس ModelProperty را یافته و خاصیت جدید DisplayName را به آن اضافه کنید:

```
// Describes the information about a property on the model
class ModelProperty {
    public string Name { get; set; }
    public string DisplayName { get; set; }
    public string ValueExpression { get; set; }
    public EnvDTE.CodeTypeRef Type { get; set; }
    public bool IsPrimaryKey { get; set; }
    public bool IsForeignKey { get; set; }
    public bool IsReadOnly { get; set; }
}
```

در حالت پیش فرض فقط از خاصیت Name برای تولید هدر جدول در ابتدای فایل t4 در حال ویرایش استفاده می‌شود. در پایان فایل t4 جاری، متد زیر را اضافه کنید:

```
static string GetDisplayName(EnvDTE.CodeProperty prop)
{
  var displayAttr = prop.Attributes.OfType<EnvDTE80.CodeAttribute2>().Where(x => x.FullName ==
typeof(System.ComponentModel.DisplayNameAttribute).FullName).FirstOrDefault();
  if(displayAttr == null)
  {
    return prop.Name;
  }
  return displayAttr.Value.Replace("\"","");
}
```

در اینجا بررسی می‌شود که آیا ویژگی DisplayNameAttribute بر روی خاصیت در حال بررسی وجود دارد یا خیر. اگر خیر از نام خاصیت استفاده خواهد شد و اگر بلی، مقدار ویژگی نام نمایشی استخراج شده و بازگشت داده می‌شود.
اکنون برای اعمال متد GetDisplayName، متد GetEligibleProperties را یافته و به نحو زیر تغییر دهید:

```
results.Add(new ModelProperty {
Name = prop.Name,
DisplayName = GetDisplayName(prop),
ValueExpression = "Model." + prop.Name,
Type = prop.Type,
IsPrimaryKey = Model.PrimaryKeyName == prop.Name,
IsForeignKey = ParentRelations.Any(x => x.RelationProperty == prop),
IsReadOnly = !prop.IsWriteable()
});
```

در اینجا خاصیت DisplayName به لیست خروجی اضافه شده است.
اکنون قسمت هدر جدول تولیدی را در ابتدای فایل t4 یافته و به نحو زیر تغییر می‌دهیم تا از DisplayName استفاده کند:

```
<#
List<ModelProperty> properties = GetModelProperties(Model.ViewDataType, true);
foreach (ModelProperty property in properties) {
    if (!property.IsPrimaryKey && !property.IsForeignKey) {
#>
        <th>
            <#= property.DisplayName #>
        </th>
<#
    }
}
#>
```

در ادامه برای آزمایش تغییرات فوق، دستور ذیل را صادر می‌کنیم:

```
PM> Scaffold Controller -ModelType Task -ControllerName TasksController -DbContextType TasksDbContext -
Repository -Force
```

پس از اجرای دستور، به فایل Views\Tasks\Index.cshtml مراجعه نمائید. اینبار هدر خودکار تولیدی از Due Date بجای DueDate استفاده کرده است.

سفارشی سازی قالب‌های پیش فرض کنترلرهای MVC Scaffolding

در ادامه قصد داریم کدهای الگوی مخزن تهیه شده را اندکی تغییر دهیم. برای مثال با توجه به اینکه از تزریق وابستگی‌ها استفاده خواهیم کرد، نیازی به سازنده اولیه پیش فرض کنترلر که در بالای آن ذکر شده «در صورت استفاده از یک DI این مورد را حذف کنید»، نداریم. برای این منظور دستور زیر را اجرا کنید:

```
PM> Scaffold CustomTemplate Controller ControllerWithRepository
```

در اینجا قصد ویرایش قالب پیش فرض کنترلرهای تشکیل شده با استفاده از الگوی مخزن را داریم. نام ControllerWithRepository از فایل ControllerWithRepository.cs.t4 موجود در پوشه packages\MvcScaffolding\tools\Controller گرفته شده است.

به این ترتیب فایل جدید CodeTemplates\Scaffolders\MvcScaffolding.Controller\ControllerWithRepository.cs.t4 به پروژه جاری اضافه خواهد شد. در این فایل چند سطر ذیل را یافته و سپس حذف کنید:

```
// If you are using Dependency Injection, you can delete the following constructor
        public <#= Model.ControllerName #>() : this(<#= String.Join(", ", Repositories.Values.Select(x
=> "new " + x.RepositoryTypeName + "()")) #>)
        {
        }
```

برای آزمایش آن دستور زیر را صادر نمائید:

```
PM> Scaffold Controller -ModelType Task -ControllerName TasksController -DbContextType TasksDbContext -
Repository -Force -ForceMode ControllerOnly
```

چون تنها قصد تغییر کنترلر را داریم از پارامتر ForceMode با مقدار ControllerOnly استفاده شده است.

یا اگر نیاز به تغییر کدهای الگوی مخزن مورد استفاده است می‌توان از دستور ذیل استفاده کرد:

```
Scaffold CustomScaffolder EFRepository
```

به این ترتیب فایل جدید CodeTemplates\Scaffolders\EFRepository\EFRepositoryTemplate.cs.t4 جهت ویرایش به پروژه جاری اضافه خواهد شد.

لیست Scaffolderهای مهیا با دستور Get-Scaffolder قابل مشاهده است.

نظرات خوانندگان

نویسنده: محسن عباس آباد عربی
تاریخ: ۱۳۹۱/۱۱/۰۴ ۹:۵۲

مرسی از مطلب مفیدتون
یا علی.

نویسنده: حسینی
تاریخ: ۱۳۹۱/۱۱/۲۰ ۱:۵

سلام . ممنونم از مطلب مفیدتون...
سوالی که دارم اینه که برای سفارشی کردن **MVC Scaffolding به طوری که همانند این قسمت شامل الگوی واحد باشد باید چگونه عمل کرد ؟**

http://www.dotnettips.info/post/842/ef-code-first-12

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۱/۲۱ ۱۹:۵۴

در چهار سطر آخر این مقاله توضیح دادن. فایل قالب الگوی مخزن رو به پروژه اضافه کنید، بعد اون رو کمی ویرایش کرده و اینترفیس و پیاده سازی لایه سرویس رو اضافه کنید.

نویسنده: ح.م
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱:۲۲

زمانی که از Scaffold CustomTemplate استفاده می‌کنیم، چنانچه در الگوهای جدید، از کلمات فارسی استفاده شود، حتی زمانی که Encoding فایلها یونی کد است(Without signature) عبارات فارسی در خروجی به هم ریخته می‌شود، برای حل مشکل در فایل Web.config تگ زیر را در قسمت system.web درج کنید:

```
<globalization fileEncoding="utf-8" requestEncoding="utf-8" responseEncoding="utf-8"/>
```

نویسنده: ایمان اسلامی
تاریخ: ۱۳۹۲/۰۹/۱۵ ۱۴:۱۹

با تشکر از مطالب خوب شما
ممکنه در مورد
سفارشی کردن MVC Scaffolding به طوری که همانند این قسمت شامل الگوی واحد باشد
توضیح بیشتری بدید؟
اینکه چگونه با ویرایش EfRepository ، میشه الگوی واحد کار رو پیاده سازی کرد.

نویسنده: رضا
تاریخ: ۱۳۹۳/۰۱/۲۲ ۲۲:۱۷

من template رو تغییر دادم و DisplayNameها جایگزین PropertyNameها میشه ، ولیکن عبارات ساده مثل "Edit" رو اگر ویرایش کنم و معادل فارسی بزارم هیچ تاثیری نداره. کسی دلیلش رو میدونه ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۲۲ ۲۲:۲۶

encoding را به این نحو باید تنظیم کرد:

```
<#@ output extension=".cs" encoding="utf-8" #>
```

« نحوه استفاده از Text templateها در دات نت - قسمت سوم »