عنوان: **اختصاصی کردن Razor برای C# در MVC با استفاده از Extension Method**
نویسنده: محمد محمدصادقی
تاریخ: ۱۳۹۳/۰۴/۲۵ ۱۳:۱۰
آدرس: www.dotnettips.info
گروه‌ها: MVC

در این مقاله ما میخواهیم RazorViewEngine را با استفاده از یک Extension Method به گونه ای تنظیم کنیم که فقط به دنبال Viewهایی که مربوط به C# هستند بگردد. در ابتدای مقاله توضیح خلاصه ای درباره Extension Method خواهیم داشت و سپس نحوه اختصاصی کردن Razor برای C# را خواهیم دید.
Extension Methodها بسیار کارآمد هستند و نحوه ایجاد و استفاده از آنها بسیار راحت است. به گونه ای که میتوان آنها را حتی برای کلاس‌های از قبل تعریف شده .Net نیز ایجاد کرد و در سرتاسر برنامه از آن استفاده کرد.
با مثالی ساده نحوه ایجاد و استفاده از Extension Method را میبینیم. در این مثال ما سعی داریم متدی برای کلاس string در .Net بنویسیم که دو رشته را به هم بچسباند.
1. ابتدا کلاسی در دایرکتوری دلخواه ایجاد میکنیم.

```
namespace ApplicationTest01.Utilities
{
    public class StringHelper
    {
    }
}
```

2. سپس متد مورد نظر را مینویسیم:

```
namespace ApplicationTest01.Utilities
{
    public class StringHelper
    {
        public string StringConcatenate(string firstPhrase, string secondPhrase)
        {
            return firstPhrase + secondPhrase;
        }
    }
}
```

3. کلاس و متدی که در آن تعریف میکنیم بایستی public و static باشند. namespace کلاس را نیز به namespace کلاس string در .Net (یعنی System) تغییر میدهیم.

```
namespace System
{
    public static class StringHelper
    {
        public static string StringConcatenate(string firstPhrase, string secondPhrase)
        {
            return firstPhrase + secondPhrase;
        }
    }
}
```

4. در Extension Methodها ورودی اول تابع به پارامتری اشاره دارد که قرار است هنگام استفاده از آن (یا صدا زدن آن) متد، عملیات مورد نظر را روی آن اجرا کند. به همین جهت عبارت this را به پارامتر ورودی اول تابع میدهیم.

```
namespace System
{
    public static class StringHelper
    {
        public static string StringConcatenate(this string firstPhrase, string secondPhrase)
        {
            return firstPhrase + secondPhrase;
        }
    }
}
```

برای استفاده از این متد کافیست پس از یک عبارت string متد را فراخوانی کنیم:

```
public ActionResult Index()
        {
            "1234".StringConcatenate("567");
            string s1 = "dotnet";
            string s2 = "tips";
            s1.StringConcatenate(s2);
            return View();
        }
```

همانطور که میبینید در ظاهر تابع فقط یک ورودی دارد ولی ما دو ورودی برای آن در نظر گرفتیم. در واقع ورودی اول تابع قبل از "." (دات) آمده است و عبارت this به آن اشاره دارد.
برای اختصاصی کردن RazorViewEngine برای #C، مشابه روند فوق یک Extension Method ایجاد میکنیم که namespace کلاس آن با namespace کلاس RazorViewEngine (یعنی System.Web.Mvc) یکی باشد. خروجی Extension Method ما از نوع RazorViewEngine میباشد:

```
namespace System.Web.Mvc
{
    public static class EngineFilter
    {
        public static RazorViewEngine DisableVbhtml(this RazorViewEngine engine)
        {
            return engine;
        }
    }
}
```

از آن جایی که کلاس RazorViewEngine برای شناسایی viewها شامل Peroperty هایی از جنس []string می‌باشد، ابتدا متدی ساده به نام FilterOutVbhtml برای فیلتر کردن stringهای حاوی عبارت "vbhtml" مینویسیم.

```
namespace System.Web.Mvc
{
    public static class EngineFilter
    {
        public static RazorViewEngine DisableVbhtml(this RazorViewEngine engine)
        {
            return engine;
        }

        private static string[] FilterOutVbhtml(string[] source)
        {
            return source.Where(s => !s.Contains("vbhtml")).ToArray();
        }
    }
}
```

در ادامه، در بدنه متد DisableVbhtml پروپرتی‌های RazorViewEngine را فرا خوانی کرده و با استفاده از متد FilterOutVbhtml آنها را فیلتر میکنیم.

```
namespace System.Web.Mvc
{
    public static class EngineFilter
    {
        public static RazorViewEngine DisableVbhtml(this RazorViewEngine engine)
        {
            engine.AreaViewLocationFormats = FilterOutVbhtml(engine.AreaViewLocationFormats);
            engine.AreaMasterLocationFormats = FilterOutVbhtml(engine.AreaMasterLocationFormats);
            engine.AreaPartialViewLocationFormats =
FilterOutVbhtml(engine.AreaPartialViewLocationFormats);
            engine.ViewLocationFormats = FilterOutVbhtml(engine.ViewLocationFormats);
            engine.MasterLocationFormats = FilterOutVbhtml(engine.MasterLocationFormats);
            engine.PartialViewLocationFormats = FilterOutVbhtml(engine.PartialViewLocationFormats);
            engine.FileExtensions = FilterOutVbhtml(engine.FileExtensions);
            return engine;
        }
```

```
        private static string[] FilterOutVbhtml(string[] source)
        {
            return source.Where(s => !s.Contains("vbhtml")).ToArray();
        }
    }
}
```

سپس در فایل Global.asax در Application_Start یکبار ViewEngineها را حذف میکنیم -Engine مربوط به aspx به صورت پیش فرض فعال میباشد- و سپس RazorViewEngine را به همراه فراخوانی Extension Method خودمان به ViewEngineها اضافه میکنیم.

```
ViewEngines.Engines.Clear();
ViewEngines.Engines.Add(new RazorViewEngine().DisableVbhtml());
```

امیدوارم مطالب فوق برای شما مفید و کارآمد باشد.

منبع: stackoverflow.com

نظرات خوانندگان

نویسنده: محمد آزاد
تاریخ: ۱۳۹۳/۰۴/۲۶ ۱:۳۰

دوست عزیز با تشکر از مقاله شما.اما می‌خواستم بدون مورد استفادش چیه و کجا به کارمون میاد؟

نویسنده: محمد محمدصادقی
تاریخ: ۱۳۹۳/۰۴/۲۶ ۱۱:۸

در حالت عادی RazorViewEngine هنگام Load کردن یک View به دنبال Viewهای مربوط به #C و VB و حتی aspx میگردد:

```
The view 'Index' or its master was not found
~/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
```

ولی اگر اصلاحات فوق برای RazorViewEngine انجام گیرد، Engine فقط به دنبال Viewهای مربوط به #C میگردد:

```
The view 'Index' or its master was not found
~/Views/Home/Index.cshtml
~/Views/Shared/Index.cshtml
```

در واقع این کار به افزایش سرعت Razor کمک میکند و در Scale بالا میتونه موثر باشه

سوال دیگه هست در خدمتم

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۳/۰۴/۲۶ ۱۹:۵

دوست عزیز برای جلوگیری از Lookuping سایر Viewهای مرتبط با ViewEngineهای دیگه (پسوند aspx) نیاز به این همه کار اضافه نیست . کافیه سایر ViewEngine هارو حذف کنید.

```
protected void Application_Start() {
    ViewEngines.Engines.Clear();
    ViewEngines.Engines.Add(new RazorViewEngine());
    ...
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۲۶ ۱۹:۵۰

این مطلب در حقیقت تکمیلی است بر « بهبود سرعت نمایش صفحات در ASP.NET MVC با حذف View Engines اضافی ». در حالت

RazorViewEngine تنها، هم فایل‌های cs و هم vb پردازش می‌شوند. در مطلب جاری پردازش فایل‌های vb آن هم فیلتر شده‌اند (توسط متد DisableVbhtml) و فقط فایل‌های cs باقی مانده‌اند.

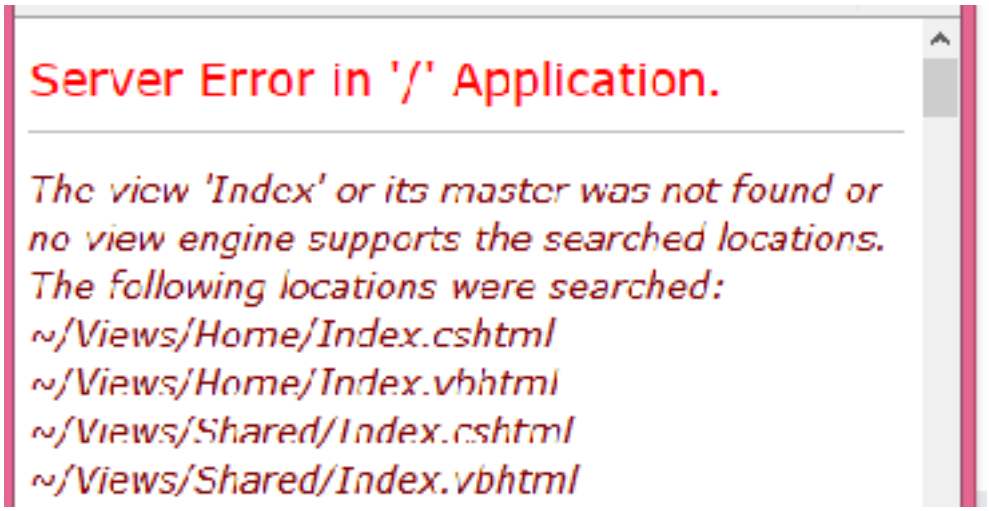
نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۳/۰۴/۲۷ ۱۹:۴۵

طبق مطالعات بنده روی سورس MVC خوشبختانه تمامی ViewEngineهای ارائه شده توسط این FrameWork از کلاس VirtualPathProviderViewEngine مشتق شدن ، این کلاس Lookuping ویو هارو عهده دار هست. برای اینکه ما جلوی Lookuping پسوند vbhtml رو بگیریم کافیه در هنگام تعریف ViewEngine به صورت زیر بنویسیم :

```
protected void Application_Start()
{
ViewEngines.Engines.Clear();
var veiwEngine = new RazorViewEngine();
veiwEngine.FileExtensions = new string[] { "cshtml" };
ViewEngines.Engines.Add(veiwEngine);
....
}
```

نویسنده: محمد محمدصادقی
تاریخ: ۱۳۹۳/۰۴/۲۸ ۱۶:۴

من کدهای شمارو اجرا کردم و خروجی زیر رو گرفتم



در تصویر بالا razor جلوی Lookuping پسوندهای vbhtml رو نگرفته. لطفا شما هم امتحان کنید ببینید خروجی دیگه ای میگیرید یا همین خروجی برای شما هم میاد؟