عنوان: **بهینه سازی سرعت یافت ویوها با سفارشی سازی Lookup Caching در Razor View Engine**
نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۳/۰۵/۰۲ ۹:۴۰
آدرس: www.dotnettips.info
گروه‌ها: MVC, Razor, Performance, Caching

در این مقاله سعی داریم تا سرعت یافت و جستجوی Viewهای متناظر با هر اکشن را در View Engine، با پیاده سازی قابلیت Caching نتیجه یافت آدرس فیزیکی viewها در درخواست‌های متوالی، افزایش دهیم تا عملا بازده سیستم را تا حدودی بهبود ببخشیم.

طی مطالعاتی که بنده بر روی سورس MVC داشتم، به صورت پیش فرض، در زمانیکه پروژه در حالت Release اجرا می‌شود، نتیجه حاصل از یافت آدرس فیزیکی ویوهای متناظر با اکشن متدها در Application cache ذخیره می‌شود (HttpContext.Cache). این امر سبب اجتناب از عمل یافت چند باره بر روی آدرس فیزیکی ویوها در درخواست‌های متوالی ارسال شده برای رندر یک ویو خواهد شد.

نکته ای که وجود دارد این هست که علاوه بر مفید بودن این امر و بهبود سرعت در درخواست‌های متوالی برای اکشن متدها، این عمل با توجه به مشاهدات بنده از سورس MVC علاوه بر مفید بودن، تا حدودی هزینه بر هم هست و هزینه‌ای که متوجه سیستم می‌شود شامل مسائل مدیریت توکار حافظه کش توسط MVC است که مسائلی مانند سیاستهای مدیریت زمان انقضاء مداخل موجود در حافظه‌ی کش اختصاص داده شده به Lookup Cahching و مدیریت مسائل thread-safe و ... را شامل می‌شود.

همانطور که می‌دانید، معمولا تعداد ویوها اینقدر زیاد نیست که Caching نتایج یافت مسیر فیزیکی view ها، حجم زیادی از حافظه Ram را اشغال کند پس با این وجود به نظر می‌رسد که اشغال کردن این میزان اندک از حافظه در مقابل بهبود سرعت، قابل چشم پوشی است و سیاست‌های توکار نامبرده فقط عملا تاثیر منفی در روند Lookup Caching پیشفرض MVC خواهند گذاشت. برای جلوگیری از تاثیرات منفی سیاست‌های نامبرده و عملا بهبود سرعت Caching نتایج Lookup آدرس فیزیکی ویوها میتوانیم یک لایه Caching سطح بالاتر به View Engine اضافه کنیم .

خوشبختانه تمامی View Engineهای MVC شامل Web Forms و Razor از کلاس VirtualPathProviderViewEngine مشتق شده‌اند که نکته مثبت که توسعه Caching اختصاصی نامبرده را برای ما مقدور می‌کند. در اینجا خاصیت (Property) قابل تنظیم ViewLocationCache از نوع IViewLocationCache هست .

بنابراین ما یک کلاس جدید ایجاد کرده و از اینترفیس IViewLocationCache مشتق میکنیم تا به صورت دلخواه بتوانیم اعضای این اینترفیس را پیاده سازی کنیم .

خوب؛ بنابر این اوصاف، من کلاس یاد شده را به شکل زیر پیاده سازی کردم :

```
public class CustomViewCache : IViewLocationCache
    {

        private readonly static string s_key = "_customLookupCach" + Guid.NewGuid().ToString();
        private readonly IViewLocationCache _cache;

        public CustomViewCache(IViewLocationCache cache)
        {
            _cache = cache;
        }

        private static IDictionary<string, string> GetRequestCache(HttpContextBase httpContext)
        {
            var d = httpContext.Cache[s_key] as IDictionary<string, string>;
            if (d == null)
            {
                d = new Dictionary<string, string>();
                httpContext.Cache.Insert(s_key, d, null, Cache.NoAbsoluteExpiration, new TimeSpan(0,
15, 0));
            }
            return d;
        }
```

```
        public string GetViewLocation(HttpContextBase httpContext, string key)
        {
            var d = GetRequestCache(httpContext);
            string location;
            if (!d.TryGetValue(key, out location))
            {
                location = _cache.GetViewLocation(httpContext, key);
                d[key] = location;
            }
            return location;
        }

        public void InsertViewLocation(HttpContextBase httpContext, string key, string virtualPath)
        {
            _cache.InsertViewLocation(httpContext, key, virtualPath);
        }
    }
```

و به صورت زیر می‌توانید از آن استفاده کنید:

```
protected void Application_Start() {
    ViewEngines.Engines.Clear();
    var ve = new RazorViewEngine();
    ve.ViewLocationCache = new CustomViewCache(ve.ViewLocationCache);
    ViewEngines.Engines.Add(ve);
    ...
}
```

نکته: فقط به یاد داشته باشید که اگر View جدیدی اضافه کردید یا یک View را حذف کردید، برای جلوگیری از بروز مشکل، حتما و حتما اگر پروژه در مراحل توسعه بر روی IIS قرار دارد app domain را ری‌استارت کنید تا حافظه کش مربوط به یافت‌ها پاک شود (و به روز رسانی) تا عدم وجود آدرس فیزیکی View جدید در کش، شما را دچار مشکل نکند.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۰۵/۰۲ ۹:۵۶

ضمن تشکر از ایده‌ای که مطرح کردید. طول عمر httpContext.Items فقط محدوده به یک درخواست و پس از پایان درخواست از بین می‌ره. مثلا یکی از کاربردهاش ذخیره اطلاعات Unit of work در طول یک درخواست هست و بعد از بین رفتن خودکار آن. بنابراین در این مثال cache.GetViewLocation اصلی بعد از یک درخواست مجددا فراخوانی میشه، چون GetRequestCache نه فقط طول عمر کوتاهی داره، بلکه اساسا کاری به key متد GetViewLocation نداره. کار s_key تعریف شده عموما تعریف lock هست نه استفاده ازش به عنوان کلید دیکشنری. بنابراین اگر خود MVC از HttpContext.Cache استفاده کرده، کار درستی بوده، چون به ازای هر درخواست نیازی نیست مجددا محاسبه بشه.

نویسنده: سید مهران موسوی
تاریخ: ۱۳۹۳/۰۵/۰۲ ۱۲:۲۱

ممنون از توجهتون ، بله من اشتباها httpContext.Items رو به کار برده بودم . کد موجود در مقاله اصلاح شد

نویسنده: حامد سبزیان
تاریخ: ۱۳۹۳/۰۵/۰۲ ۱۸:۴

بهبودی حاصل نشده. در DefaultViewLocationCache خود MVC مسیرها از httpContext.Cache خوانده می‌شود، در کد شما هم از همان. استفاده از httpContext.Items در کد شما ممکن است اندکی بهینه بودن را افزایش دهد، به شرط استفاده بیش از یک بار از یک (چند) View در طول یک درخواست.

Optimizing ASP.NET MVC view lookup performance

همان طور که در انتهای مقاله اشاره شده است، استفاده از یک ConcurrentDictionary می‌تواند کارایی خوبی داشته باشد اما خوب استاتیک است و به حذف و اضافه شدن فیزیکی Viewها حساس نیست.