عنوان: **ASP.NET MVC #19**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۱ ۱۴:۱۶:۰۰
آدرس: www.dotnettips.info
گروه‌ها: MVC

مروری بر امکانات Caching اطلاعات در ASP.NET MVC

در برنامه‌های وب، بالاترین حد کارآیی برنامه‌ها از طریق بهینه سازی الگوریتم‌ها حاصل نمی‌شود، بلکه با بکارگیری امکانات Caching سبب خواهیم شد تا اصلا کدی اجرا نشود. در ASP.NET MVC این هدف از طریق بکارگیری فیلتری به نام OutputCache میسر می‌گردد:

```
using System.Web.Mvc;

namespace MvcApplication16.Controllers
{
    public class HomeController : Controller
    {
        [OutputCache(Duration = 60, VaryByParam = "none")]
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

همانطور که ملاحظه می‌کنید، OutputCache را به یک اکشن متد یا حتی به یک کنترلر نیز می‌توان اعمال کرد. به این ترتیب HTML نهایی حاصل از View متناظر با اکشن متد جاری فراخوانی شده، Cache خواهد شد. سپس زمانیکه درخواست بعدی به سرور ارسال می‌شود، نتیجه دریافت شده، همان اطلاعات Cache شده قبلی است و عملا در سمت سرور کدی اجرا نخواهد شد. در اینجا توسط پارامتر Duration، مدت زمان معتبر بودن کش حاصل، برحسب ثانیه مشخص می‌شود. VaryByParam مشخص می‌کند که اگر متدی پارامتری را دریافت می‌کند، آیا باید به ازای هر مقدار دریافتی، مقادیر کش شده متفاوتی ذخیره شوند یا خیر. در اینجا چون متد Index پارامتری ندارد، از مقدار none استفاده شده است.

مثال یک

یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. سپس کنترلر جدید Home را نیز به آن اضافه نمائید:

```
using System;
using System.Web.Mvc;

namespace MvcApplication16.Controllers
{
    public class HomeController : Controller
    {
        [OutputCache(Duration = 60, VaryByParam = "none")]
        public ActionResult Index()
        {
            ViewBag.ControllerTime = DateTime.Now;
            return View();
        }
    }
}
```

همچنین کدهای View متد Index را نیز به نحو زیر تغییر دهید:

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<p>@ViewBag.ControllerTime</p>
<p>@DateTime.Now</p>
```

در اینجا نمایش دو زمان دریافتی از کنترلر و زمان محاسبه شده در View را مشاهده می‌کنید. هدف این است که بررسی کنیم آیا فیلتر OutputCache بر روی این دو مقدار تاثیری دارد یا خیر.
برنامه را اجرا نمائید. سپس چند بار صفحه را Refresh کنید. مشاهده خواهید کرد که هر دو زمان یاد شده تا 60 ثانیه، تغییری نخواهند کرد و حاصل نهایی از Cache خواهنده می‌شود.
کاربرد یک چنین حالتی برای مثال نمایش اطلاعات بازدیدهای یک سایت است. نباید به ازای هر کاربر وارد شده به سایت، یکبار به بانک اطلاعاتی مراجعه کرد و آمار جدیدی را تهیه نمود. یا برای نمونه اگر جایی قرار است اطلاعات وضعیت آب و هوا نمایش داده شود، بهتر است این اطلاعات، مثلا هر نیم ساعت یکبار به روز شود و نه به ازای هر بازدید جدید از سایت، توسط صدها بازدید کننده همزمان. یا برای مثال کش کردن خروجی فید RSS یک بلاگ به مدت چند ساعت نیز ایده خوبی است. از این لحاظ که اگر اطلاعات بلاگ شما روزی یکبار به روز می‌شود، نیازی نیست تا به ازای هر برنامه فیدخوان، یکبار اطلاعات از بانک اطلاعاتی دریافت شده و پروسه رندر نهایی فید صورت گیرد. منوهای پویای یک سایت نیز در همین رده قرار می‌گیرند. دریافت اطلاعات منوهای پویای سایت به ازای هر درخواست رسیده کاربری جدید، کار اشتباهی است. این اطلاعات نیز باید کش شوند تا بار سرور کاهش یابد. البته تمام این‌ها زمانی میسر خواهند شد که اطلاعات سمت سرور کش شوند.

مثال دو

همان مثال قبلی را در اینجا جهت بررسی پارامتر VaryByParam به نحو زیر تغییر می‌دهیم:

```
using System;
using System.Web.Mvc;

namespace MvcApplication16.Controllers
{
    public class HomeController : Controller
    {
        [OutputCache(Duration = 60, VaryByParam = "none")]
        public ActionResult Index(string parameter)
        {
            ViewBag.Msg = parameter ?? string.Empty;
            ViewBag.ControllerTime = DateTime.Now;
            return View();
        }
    }
}
```

در اینجا یک پارامتر به متد Index اضافه شده است. مقدار آن به ViewBag.Msg انتساب داده شده و سپس در View ، در بین تگ‌های h2 نمایش داده خواهد شد. همچنین یک فرم ساده هم جهت ارسال parameter به متد Index اضافه شده است:

```
@{
    ViewBag.Title = "Index";
}

<h2>@ViewBag.Msg</h2>

<p>@ViewBag.ControllerTime</p>
<p>@DateTime.Now</p>
```

```
@using (Html.BeginForm())
{
    @Html.TextBox("parameter")
    <input type="submit" />
}
```

اکنون برنامه را اجرا کنید. در TextBox نمایش داده شده یکبار مثلا بنویسید Test1 و فرم را به سرور ارسال نمائید. سپس مقدار Test2 را وارد کرده و ارسال نمائید. در بار دوم، خروجی صفحه همانند زمانی است که مقدار Test1 ارسال شده است. علت این است که مقدار VaryByParam به none تنظیم شده است و صرفنظر از ورودی کاربر، همان اطلاعات کش شده قبلی بازگشت داده خواهد شد. برای رفع این مشکل، متد Index را به نحو زیر تغییر دهید، به طوریکه مقدار VaryByParam به نام پارامتر متد جاری اشاره کند:

```
[OutputCache(Duration = 60, VaryByParam = "parameter")]
public ActionResult Index(string parameter)
```

در ادامه مجددا برنامه را اجرا کنید. اکنون یکبار مقدار Test1 را به سرور ارسال کنید. سپس مقدار Test2 را ارسال نمائید. مجددا همین دو مرحله را با مقادیر Test1 و Test2 تکرار کنید. مشاهده خواهید کرد که اینبار اطلاعات بر اساس مقدار پارامتر ارسالی کش شده است.

تنظیمات متفاوت OutputCache

الف) VaryByParam : اگر مساوی none قرار گیرد، همواره همان مقدار کش شده قبلی نمایش داده می‌شود. اگر مقدار آن به نام پارامتر خاصی تنظیم شود، اطلاعات کش شده بر اساس مقادیر متفاوت پارامتر دریافتی، متفاوت خواهند بود. در اینجا پارامترهای متفاوت را با یک «,» می‌توان از هم جدا ساخت. اگر تعداد پارامترها زیاد است می‌توان مقدار VaryByParam را مساوی با * قرار داد. در این حالت به ازای مقادیر متفاوت دریافتی پارامترهای مختلف، اطلاعات مجزایی در کش قرار خواهد گرفت. این روش آخر آنچنان توصیه نمی‌شود چون سربار بالایی دارد و حجم بالایی از اطلاعات بر اساس پارامترهای مختلف، باید در کش قرار گیرند.
ب) Location : مکان قرارگیری اطلاعات کش شده را مشخص می‌کند. مقدار آن نیز بر اساس یک enum به نام OutputCacheLocation مشخص می‌گردد. در این حالت برای مثال می‌توان مکان‌های Server، Client و ServerAndClient را مقدار دهی نمود. مقدار Downstream به معنای کش شدن اطلاعات بر روی پروکسی سرورهای بین راه و یا مرورگرها است. پیش فرض آن Any است که ترکیبی از Server و Downstream می‌باشد.
اگر قرار است اطلاعات یکسانی به تمام کاربران نمایش داده شود، مثلا محتوای لیست یک منوی پویا، محل قرارگیری اطلاعات کش باید سمت سرور باشد. اگر نیاز است به ازای هر کاربر محتوای اطلاعات کش شده متفاوت باشد، بهتر است محل سمت کلاینت را مقدار دهی نمود.
ج) VaryByHeader : اطلاعات، بر اساس هدرهای مشخص شده، کش می‌شوند. برای مثال مرسوم است که از Accept-Language در اینجا استفاده شود تا اطلاعات مثلا فرانسوی کش شده، به کاربر آلمانی تحویل داده نشود.
د) VaryByCustom : در این حالت نام یک متد استاتیک تعریف شده در فایل global.asax.cs باید مشخص گردد. توسط این متد کلید رشته‌ای اطلاعاتی که قرار است کش شود، بازگشت داده خواهد شد.
ه) SqlDependency : در این حالت اطلاعات تا زمانیکه تغییری در جداول بانک اطلاعاتی SQL Server صورت نگیرد، کش خواهد شد.
و) Nostore : به پروکسی سرورهای بین راه و همچنین مرورگرها اطلاع می‌دهد که اطلاعات را نباید کش کنند. اگر قسمت اعتبار سنجی این سری را به خاطر داشته باشید، چنین تعریفی در قسمت Remote validation بکارگرفته شد:

```
[OutputCache(Location = OutputCacheLocation.None, NoStore = true)]
```

و یا می‌توان برای اینکار یک فیلتر سفارشی را نیز تهیه کرد:

```
using System;
using System.Web.Mvc;

namespace MvcApplication16.Helper
{
    /// <summary>
    /// Adds "Cache-Control: private, max-age=0" header,
    /// ensuring that the responses are not cached by the user's browser.
    /// </summary>
    public class NoCachingAttribute : ActionFilterAttribute
    {
        public override void OnActionExecuted(ActionExecutedContext filterContext)
        {
            base.OnActionExecuted(filterContext);
            filterContext.HttpContext.Response.CacheControl = "private";
            filterContext.HttpContext.Response.Cache.SetMaxAge(TimeSpan.FromSeconds(0));
        }
    }
}
```

کار این فیلتر اضافه کردن هدر «Cache-Control: private, max-age=0» به Response است.

استفاده از فایل Web.Config برای معرفی تنظیمات Caching

یکی دیگر از تنظیمات ویژگی OutputCache، پارامتر CacheProfile است که امکان تنظیم آن در فایل web.config نیز وجود دارد. برای نمونه تنظیمات زیر را به قسمت system.web فایل وب کانفیگ برنامه اضافه کنید:

```
<system.web>
    <caching>
      <outputCacheSettings>
        <outputCacheProfiles>
          <add name="Aggressive" location="ServerAndClient" duration="300"/>
          <add name="Mild" duration="100"  location="Server" />
        </outputCacheProfiles>
      </outputCacheSettings>
    </caching>
```

سپس مثلا برای استفاده از پروفایلی به نام Aggressive، خواهیم داشت:

```
[OutputCache(CacheProfile = "Aggressive", VaryByParam = "parameter")]
public ActionResult Index(string parameter)
```

استفاده از ویژگی به نام donut caching

تا اینجا به این نتیجه رسیدیم که OutputCache، کل خروجی یک View را بر اساس پارامترهای مختلفی که دریافت می‌کند، کش خواهد کرد. در این بین اگر بخواهیم تنها قسمت کوچکی از صفحه کش نشود چه باید کرد؟ برای حل این مشکل قابلیتی به نام cache substitution که به donut caching هم معروف است (چون آن‌را می‌توان به شکل یک donut تصور کرد!) در ASP.NET MVC

قابل استفاده است.

```
@{ Response.WriteSubstitution(ctx => DateTime.Now.ToShortTimeString()); }
```

همانطور که ملاحظه می‌کنید برای تعریف یک چنین اطلاعاتی باید از متد Response.WriteSubstitution در یک view استفاده کرد. در این مثال، نمایش زمان جاری معرفی شده، صرف نظر از وضعیت کش صفحه جاری، کش نخواهد شد.

عکس آن هم ممکن است. فرض کنید که صفحه جاری شما از سه partial view تشکیل شده است. هر کدام از این partial viewها نیز مزین به OutpuCache هستند. اما صفحه اصلی درج کننده اطلاعات این سه partial view فاقد ویژگی Output کش است. در این حالت تنها اطلاعات این partial viewها کش خواهند شد و سایر قسمت‌های صفحه با هر بار درخواست از سرور، مجددا بر اساس اطلاعات جدید به روز خواهند شد. حالت توصیه شده نیز همین مورد است و متد Response.WriteSubstitution را صرفا جهت اطلاعات عمومی درنظر داشته باشید.

استفاده از امکانات Data Caching به صورت مستقیم

مطالبی که تا اینجا عنوان شدند به کش کردن اطلاعات Response اختصاص داشتند. اما امکانات Caching موجود، به این مورد خلاصه نشده و می‌توان اطلاعات و اشیاء را نیز کش کرد. برای مثال اطلاعات «با سطح دسترسی عمومی» دریافتی از بانک اطلاعاتی توسط یک کوئری را نیز می‌توان کش کرد. جهت انجام اینکار می‌توان از متدهای HttpRuntime.Cache.Insert و یا HttpContext.Cache.Insert استفاده کرد. استفاده از HttpContext.Cache.Insert حین نوشتن Unit tests دردسر کمتری دارد و mocking آن ساده است؛ از این جهت که بر اساس HttpContextBase تعریف شده‌است.
در ادامه یک کلاس کمکی نوشتن اطلاعات در cache و سپس بازیابی آن‌را ملاحظه می‌کنید:

```
using System;
using System.Web;
using System.Web.Caching;

namespace MvcApplication16.Helper
{
    public static class CacheManager
    {
        public static void CacheInsert(this HttpContextBase httpContext, string key, object data, int
durationMinutes)
        {
            if (data == null) return;
            httpContext.Cache.Add(
                key,
                data,
                null,
                DateTime.Now.AddMinutes(durationMinutes),
                TimeSpan.Zero,
                CacheItemPriority.AboveNormal,
                null);
        }

        public static T CacheRead<T>(this HttpContextBase httpContext, string key)
        {
            var data = httpContext.Cache[key];
            if (data != null)
                return (T)data;
            return default(T);
        }

        public static void InvalidateCache(this HttpContextBase httpContext, string key)
        {
            httpContext.Cache.Remove(key);
        }
    }
}
```

و برای استفاده از آن در یک اکشن متد، ابتدا نیاز است فضای نام این کلاس تعریف شود و سپس برای نمونه متد HttpContext.CacheInsert در دسترس خواهد بود. HttpContext یکی از خواص تعریف شده در شیء کنترلر است که با ارث بری کنترلرها از آن، همواره در دسترس می‌باشد.

در اینجا برای نمونه اطلاعات یک لیست جنریک دریافتی از بانک اطلاعاتی را مثلا 10 دقیقه (بسته به پارامتر durationMinutes آن) می‌توان کش کرد و سپس توسط متد CacheRead آن‌را دریافت نمود. اگر متد CacheRead نال برگرداند به معنای خالی بودن کش است. بنابراین یکبار اطلاعات را از بانک اطلاعاتی دریافت نموده و سپس آن‌را کش خواهیم کردیم.

البته هستند ORMهایی که یک چنین کارهایی را به صورت توکار پشتیبانی کنند. به مکانیزم آن، Second level cache هم گفته می‌شود؛ به علاوه امکان استفاده از پروایدرهای دیگری را بجز کش IIS برای ذخیره سازی موقتی اطلاعات نیز فراهم می‌کنند.

همچنین باید دقت داشت این اعداد مدت زمان، هیچگونه ضمانتی ندارند. اگر IIS احساس کند که با کمبود منابع مواجه شده است، به سادگی شروع به حذف اطلاعات موجود در کش خواهد کرد.

نکته امنیتی مهم!

به هیچ عنوان از OutputCache در صفحاتی که نیاز به اعتبار سنجی دارند، استفاده نکنید و به همین جهت در قسمت کش کردن اطلاعات، بر روی «اطلاعاتی با سطح دسترسی عمومی» تاکید شد.

فرض کنید کارمندی به صفحه مشاهده فیش حقوقی خودش مراجعه کرده است. این ماه هم اضافه حقوق آنچنانی داشته است. شما هم این صفحه را به مدت سه ساعت کش کرده‌اید. آیا می‌توانید تصور کنید اگر همین گزارش کش شده با این اطلاعات، به سایر کارمندان نمایش داده شود چه قشقرقی به پا خواهد شد؟!

بنابراین هیچگاه اطلاعات مخصوص به یک کاربر اعتبار سنجی شده را کش نکنید و «تنها» اطلاعاتی نیاز به کش شدن دارند که عمومی باشند. برای مثال لیست آخرین اخبار سایت؛ لیست آخرین مدخل‌های فید RSS سایت؛ لیست اطلاعات منوی عمومی سایت؛ لیست تعداد کاربران مراجعه کننده به سایت در طول یک روز؛ گزارش آب و هوا و کلیه اطلاعاتی با سطح دسترسی عمومی که کش شدن آن‌ها مشکل ساز نباشد.

به صورت خلاصه هیچگاه در کدهای شما چنین تعریفی نباید مشاهده شود:

```
[Authorize]
[OutputCache(Duration = 60)]
public ActionResult Index()
```

نظرات خوانندگان

نویسنده: Naser Tahery
تاریخ: ۱۳۹۱/۰۲/۱۱ ۲۳:۰۹:۲۷

آقای نصیری اگر ما اطلاعاتی داشته باشیم که هفته ای یک بار به روز میشوند چه باید کرد؟
در کل آیا میشود مدت زمان کشینگ را به ساعت بدهیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۱۱ ۲۳:۳۶:۳۲

در حالت تئوری، بله می‌شود. مثلا پارامتر duration که بر حسب ثانیه است را مقدار دهی کرد (یک تبدیل واحد ساده است). یا در حالت متد CacheInsert ذکر شده نیز به همین ترتیب. اما در عمل IIS پروسه اجرایی سایت رو بر اساس تنظیمات Application pool در مدت زمان‌های مشخصی اصطلاحا Recycle می‌کنه؛ یعنی برنامه ری استارت میشه و کش از دست خواهد رفت. البته این زمان در تنظیمات IIS قابل تغییر است: (^)

نویسنده: مهدی غیاثی
تاریخ: ۱۳۹۱/۰۶/۲۷ ۱:۱۶

آقای نصیری، آیا راهی هست که این caching به ازای هر کاربر انجام بشه؟ مثلا یک صفحه که پردازش سنگینی داره و به ازای هر کاربر هم اون صفحه خروجی متفاوتی داره، برای یک مدت کش بشه... آیا راهی در خود ASP.NET MVC هست یا اینکه باید خودم چنین ویژگی‌ای رو پیاده سازی کنم؟
(من از سیستم Membership اصلی خود ASP.NET استفاده می‌کنم)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۷ ۹:۱۳

می‌شود از VaryByParam استفاده کرد (مثال دوم فوق)

```
[OutputCache(Duration = 60, VaryByParam = "userId")]
public ActionResult Index(string userId)
```

البته کش کردن صفحاتی که نیاز به اعتبارسنجی دارند اشتباه است (نکته مهم انتهای بحث).
- از کلاس CacheManager مطرح شده در انتهای بحث استفاده کنید. کلید آن‌را مساوی یک عبارت منحصر به فرد مانند شماره کاربری به علاوه نام صفحه قرار دهید. مقدار آن را حاصل عملیات سنگینی که مد نظر دارید.

نویسنده: سعید یزدانی
تاریخ: ۱۳۹۱/۱۱/۲۲ ۲۰:۴۶

در مورد متد CacheInsert بالا چندتا سوال داشتم:
1) ما چرا از httpcontext استفاده کردیم .
2) نحوه‌ی فراخوانی ان به چه شکل است .
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۱/۲۲ ۲۱:۱

هر کلاس کنترلر از کلاس پایه‌ای به نام Controller مشتق می‌شود. یک سری خاصیت و متد هم در این کلاس پایه وجود دارند. یکی از این‌ها دقیقا this.HttpContext است که برای آن یک متد الحاقی به نام CacheInsert تهیه شده. بنابراین نحوه فراخوانی آن در یک اکشن متد به صورت this.HttpContext.CacheInsert. **this** است که this آن در مطلب فوق عنوان نشده و ضروری هم

نیست.

نویسنده: سعید یزدانی
تاریخ: ۱۳۹۱/۱۱/۲۲ ۲۲:۲۰

مرسی از توضیح کاملتون
در واقع این متد یک extention از نوع httpcontext هست

نویسنده: وحید
تاریخ: ۱۳۹۲/۱۰/۱۷ ۰:۵۱

با سلام اگر بخوام در یک لایه دیگر از insertCache یا readCache استفاده کنم میشه؟چون من در یک کلاس دیگه امتحان کردم وقتی httpcontext.current را میزنم در لیست آن موجود نیست با تشکر از شما

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۷ ۱:۱۶

بله. اسمبلی استاندارد System.Web.dll را باید به آن پروژه اضافه کنید (از منوی پروژه گزینه افزودن ارجاعات).

نویسنده: وحید
تاریخ: ۱۳۹۲/۱۰/۱۷ ۸:۵۷

ممنون از شما
منظور من اینه که کلاسی در لایه دیگر داشته باشم که بتواند با HttpContext.Current.User.Identity.IsAuthenticated دستور مشخص کنم که اگر کاربر Authenticate حالا نام و نام خانوادگی آن را با readcach بخواند در غیر اینصورت از db بخواند و در کش درج شود و هر جا خواستم این متد را استفاده کنم. که این عمل در لایه دیگر انجام نمی‌شود. یعنی وقتی میزنم httpcontext.current.Readcach آن را نمی‌شناسد حتی بنده System.Web.dll را هم اضافه نمودم ممنون از شما.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۷ ۹:۴۲

چون به صورت متد الحاقی تعریف شده، باید فضای نام مثلا MvcApplication16.Helper (مانند مطلب فوق) نیز در ابتدای فایل شما ذکر شود. در غیر اینصورت این extension method جدید، شناسایی نخواهد شد.

نویسنده: nima
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱۴:۵۷

با سلام؛ یک مشکل برای من پیش میاد. من از هیچ کشی استفاده نمیکنم اما صفحات کاملا کش میشه. حتی نام کنترلر و ویوها رو هم عوض میکنم که برنامه ارور بده اما برنامه باز هم اجرا میشه. با تغییر MapRoute باز هم برنامه از کش خارج نمیشه.
میشه لطفا راهنمایی کنید ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱۷:۳۹

ممکن هست در بین راه یک web cache proxy قرار داشته باشد (مثلا در ISP شما).
استفاده از ویژگی OutputCache به صورت زیر، کش مرورگر و کش سرورهای واسط را غیرفعال می‌کند:

```
[OutputCache(NoStore = true, Duration = 0, VaryByParam = "*")]
```

نویسنده: احمد
تاریخ: ۱۳۹۳/۰۷/۲۴ ۱۵:۵۴

من یه اکشن دارم که داخل این اکشن گفتم اگر id مقدار داشت view1 بازگشت داده شود و در صورتی که نداشت view2 بازگشت داده شود، حالا لازمه که وقتی میخواد view2 بازگشت داده شود، اطلاعات کش بشه. من اومدم هرکدوم از این بخش‌ها رو redirect کردم به اکشن دیگه تا اونجا پردازش انجام بشه و بتونم اکشن مورد نظرمو کش کنم، ولی مشکل اینجاست که وقتی return view میشه آدرس مرورگر تغییر میکنه و همون آدرس درخواست شده نیست. چطور میتونم اکشن مورد نظرمو کش کنم ولی آدرس تغییر نکنه؟ آیا این روش مناسبیه من انتخاب کردم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۲۴ ۱۹:۸

خیر. از VaryByParam = "id" استفاده کنید (در متن فوق بیشتر در مورد آن بحث شده؛ یعنی به ازای idهای مختلف، کش‌های مختلف و متناظر خروجی نهایی متد). در این حالت نیازی به return redirect نیست که آدرس صفحه را تغییر می‌دهد. حالت return view امکان مسیردهی فایل view را هم دارد، برای انتخاب فایل‌های مختلف؛ مثلا: ("return View("~/Views/Items/Details.cshtml

نویسنده: علیرضا
تاریخ: ۱۳۹۳/۰۸/۰۹ ۱۱:۱۵

من متد تولید منو رو در یک helper نوشتم. چگونه می‌توانم آن را کش کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۸/۰۹ ۱۱:۴۹

- یک partial view به نام مثلا SidebarMenu ایجاد می‌کنید ؛ جهت رندر قسمت منو با هر محتوا و روشی که صلاح می‌دانید.
- سپس یک کنترلر مخصوص آن ایجاد می‌کنید که این partial view را ارائه دهد:

```
public partial class SidebarMenuController : Controller
    {
        const int Min15 = 900;

        [ChildActionOnly]
        [OutputCache(Duration = Min15)]
        public virtual ActionResult Index()
        {
            return PartialView(viewName: MVC.Shared.Views._SidebarMenu);
        }
    }
```

به عمد توسط ChildActionOnly مزین شده تا مستقیما قابل فراخوانی نباشد.
- در آخر در فایل layout خواهید داشت:

```
@{Html.RenderAction(result: MVC.SidebarMenu.Index());}
```

یک نکته‌ی تکمیلی:
مسیردهی‌های خاص فوق از T4MVC استفاده می‌کنند.