عنوان: **ایجاد یک فیلتر سفارشی جهت تعیین Layout برای کنترلر و یا اکشن متد**
نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۲/۰۹/۱۹ ۲۳:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: MVC, ActionFilters

همانطور که می‌دانید در صورت عدم تعریف صریح layout در یک View، این تعریف از فایل Views_ViewStart.cshtml دریافت می‌گردد:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

برای معرفی صریح فایل layout، تنها کافی است مسیر کامل فایل layout را در یک View مشخص کنیم:

```
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>
```

حال ما می‌خواهیم یک فیلتر سفارشی را تعریف کنیم تا به براحتی امکان تعریف Layout در سطح کنترلر و هم در سطح اکشن متد به صورت Attribute را داشته باشد :

```
[SetLayoutAttribute("_MyLayout")]
public ActionResult Index()
{
      return View();
}
```

همچنین می‌توانیم این فیلتر سفارشی را در سطح کنترلر تعریف کنیم تا تمام اکشن متدهای داخل کنترلر از Layout مربوطه استفاده کنند:

```
[SetLayoutAttribute("_MyLayout")]
    public class HomeController : Controller
    {

        public ActionResult Index()
        {
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
```

برای تعریف چنین اکشن فیلتری کد زیر را می‌نویسیم :

```
public class SetLayoutAttribute : ActionFilterAttribute
```

```
{
    private readonly string _masterName;
    public SetLayoutAttribute(string masterName)
    {
        _masterName = masterName;
    }

    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        base.OnActionExecuted(filterContext);
        var result = filterContext.Result as ViewResult;
        if (result != null)
        {
            result.MasterName = _masterName;
        }
    }
}
```

همانطور که می‌دانید برای تعریف یک اکشن فیلتر سفارشی می‌بایست از کلاس ActionFilterAttribute ارث بری کنیم، حالا برای کلاسی که تعریف کرده ایم یک خصوصیت readonly را تعریف و سپس برابر با یک پارامتر با نام masterName که از طریق سازنده کلاس دریافت می‌شود قرار داده ایم. در نهایت متد OnActionExecuted را بازنویسی کرده ایم به این صورت که مقدار دریافتی توسط سازنده کلاس را برابر با نام Layout موردنظرمان است را به خاصیت MasterName اختصاص میدهیم.

عنوان: **حذف فضاهای خالی در خروجی صفحات ASP.NET MVC**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۹/۲۲ ۲۰:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, Performance, ActionFilters, HTTP Compression

صفحات خروجی وب سایت زمانی که رندر شده و در مرورگر نشان داده می‌شود شامل فواصل اضافی است که تاثیری در نمایش سایت نداشته و صرفا این کاراکترها فضای اضافی اشغال می‌کنند. با حذف این کاراکترهای اضافی می‌توان تا حد زیادی صفحه را کم حجم کرد. برای این کار در ASP.NET Webform کارهایی (^) انجام شده است.
روال کار به این صورت بوده که قبل از رندر شدن صفحه در سمت سرور خروجی نهایی بررسی شده و با استفاده از عبارات با قاعده الگوهای مورد نظر لیست شده و سپس حذف می‌شوند و در نهایت خروجی مورد نظر حاصل خواهد شد. برای راحتی کار و عدم نوشتن این روال در تمامی صفحات می‌تواند در مستر پیج این عمل را انجام داد. مثلا:

```
private static readonly Regex RegexBetweenTags = new Regex(@">\s+<", RegexOptions.Compiled);
        private static readonly Regex RegexLineBreaks = new Regex(@"\r\s+", RegexOptions.Compiled);

        protected override void Render(HtmlTextWriter writer)
        {
            using (var htmlwriter = new HtmlTextWriter(new System.IO.StringWriter()))
            {
                base.Render(htmlwriter);
                var html = htmlwriter.InnerWriter.ToString();

                html = RegexBetweenTags.Replace(html, "> <");
                html = RegexLineBreaks.Replace(html, string.Empty);
                html = html.Replace("//<![CDATA[", "").Replace("//]]>", "");
                html = html.Replace("// <![CDATA[", "").Replace("// ]]>", "");

                writer.Write(html.Trim());
            }
        }
```

در هر صفحه رویدادی به نام Render وجود دارد که خروجی نهایی را می‌توان در آن تغییر داد. همانگونه که مشاهده می‌شود عملیات یافتن و حذف فضاهای خالی در این متد انجام می‌شود.
این عمل در ASP.NET Webform به آسانی انجام شده و باعث حذف فضاهای خالی در خروجی صفحه می‌شود.
برای انجام این عمل در ASP.NET MVC روال کار به این صورت نیست و نمی‌توان مانند ASP.NET Webform عمل کرد.
چون در MVC از ViewPage استفاده می‌شود و ما مستقیما به خروجی آن دسترسی نداریم یک روش این است که می‌توانیم یک کلاس برای ViewPage تعریف کرده و رویداد Write آن را تحریف کرده و مانند مثال بالا فضای خالی را در خروجی حذف کرد. البته برای استفاده باید کلاس ایجاد شده را به عنوان فایل پایه جهت ایجاد صفحات در MVC فایل web.config معرفی کنیم. این روش در اینجا به وضوح شرح داده شده است.
اما هدف ما پیاده سازی با استفاده از اکشن فیلتر هاست. برای پیاده سازی ابتدا یک اکشن فیلتر به نام CompressAttribute تعریف می‌کنیم مانند زیر:

```
using System;
using System.IO;
using System.IO.Compression;
using System.Text;
using System.Text.RegularExpressions;
using System.Web;
using System.Web.Mvc;

namespace PWS.Common.ActionFilters
{
    public class CompressAttribute : ActionFilterAttribute
    {
         #region Methods (2)

        // Public Methods (1)

        /// <summary>
        /// Called by the ASP.NET MVC framework before the action method executes.
        /// </summary>
        /// <param name="filterContext">The filter context.</param>
```

```
        public override void OnActionExecuting(ActionExecutingContext filterContext)
        {
            var response = filterContext.HttpContext.Response;
            if (IsGZipSupported(filterContext.HttpContext.Request))
            {
                String acceptEncoding = filterContext.HttpContext.Request.Headers["Accept-Encoding"];
                if (acceptEncoding.Contains("gzip"))
                {
                    response.Filter = new GZipStream(response.Filter, CompressionMode.Compress);
                    response.AppendHeader("Content-Encoding", "gzip");
                }
                else
                {
                    response.Filter = new DeflateStream(response.Filter, CompressionMode.Compress);
                    response.AppendHeader("Content-Encoding", "deflate");
                }
            }
            // Allow proxy servers to cache encoded and unencoded versions separately
            response.AppendHeader("Vary", "Content-Encoding");
           //حذف فضاهای خالی
```

```
            response.Filter = new WhitespaceFilter(response.Filter);
        }
        // Private Methods (1)

        /// <summary>
        /// Determines whether [is G zip supported] [the specified request].
        /// </summary>
        /// <param name="request">The request.</param>
        /// <returns></returns>
        private Boolean IsGZipSupported(HttpRequestBase request)
        {
            String acceptEncoding = request.Headers["Accept-Encoding"];

            if (acceptEncoding == null) return false;
            return !String.IsNullOrEmpty(acceptEncoding) && acceptEncoding.Contains("gzip") ||
acceptEncoding.Contains("deflate");
        }

#endregion Methods
    }

    /// <summary>
    /// Whitespace Filter
    /// </summary>
    public class WhitespaceFilter : Stream
    {
#region Fields (3)

        private readonly Stream _filter;
        /// <summary>
        ///
        /// </summary>
        private static readonly Regex RegexAll = new Regex(@"\s+|\t\s+|\n\s+|\r\s+",
RegexOptions.Compiled);
        /// <summary>
        ///
        /// </summary>
        private static readonly Regex RegexTags = new Regex(@">\s+<", RegexOptions.Compiled);

#endregion Fields

#region Constructors (1)

        /// <summary>
        /// Initializes a new instance of the <see cref="WhitespaceFilter" /> class.
        /// </summary>
        /// <param name="filter">The filter.</param>
        public WhitespaceFilter(Stream filter)
        {
            _filter = filter;
        }

#endregion Constructors

#region Properties (5)
```

```
        //methods that need to be overridden from stream
        /// <summary>
        /// When overridden in a derived class, gets a value indicating whether the current stream
supports reading.
        /// </summary>
        /// <returns>true if the stream supports reading; otherwise, false.</returns>
        public override bool CanRead
        {
            get { return true; }
        }

        /// <summary>
        /// When overridden in a derived class, gets a value indicating whether the current stream
supports seeking.
        /// </summary>
        /// <returns>true if the stream supports seeking; otherwise, false.</returns>
        public override bool CanSeek
        {
            get { return true; }
        }

        /// <summary>
        /// When overridden in a derived class, gets a value indicating whether the current stream
supports writing.
        /// </summary>
        /// <returns>true if the stream supports writing; otherwise, false.</returns>
        public override bool CanWrite
        {
            get { return true; }
        }

        /// <summary>
        /// When overridden in a derived class, gets the length in bytes of the stream.
        /// </summary>
        /// <returns>A long value representing the length of the stream in bytes.</returns>
        public override long Length
        {
            get { return 0; }
        }

        /// <summary>
        /// When overridden in a derived class, gets or sets the position within the current stream.
        /// </summary>
        /// <returns>The current position within the stream.</returns>
        public override long Position { get; set; }

#endregion Properties

#region Methods (6)

// Public Methods (6)

        /// <summary>
        /// Closes the current stream and releases any resources (such as sockets and file handles)
associated with the current stream. Instead of calling this method, ensure that the stream is properly
disposed.
        /// </summary>
        public override void Close()
        {
            _filter.Close();
        }

        /// <summary>
        /// When overridden in a derived class, clears all buffers for this stream and causes any
buffered data to be written to the underlying device.
        /// </summary>
        public override void Flush()
        {
            _filter.Flush();
        }

        /// <summary>
        /// When overridden in a derived class, reads a sequence of bytes from the current stream and
advances the position within the stream by the number of bytes read.
        /// </summary>
        /// <param name="buffer">An array of bytes. When this method returns, the buffer contains the
specified byte array with the values between <paramref name="offset" /> and (<paramref name="offset" />
+ <paramref name="count" /> - 1) replaced by the bytes read from the current source.</param>
        /// <param name="offset">The zero-based byte offset in <paramref name="buffer" /> at which to
begin storing the data read from the current stream.</param>
        /// <param name="count">The maximum number of bytes to be read from the current stream.</param>
```

```
        /// <returns>
        /// The total number of bytes read into the buffer. This can be less than the number of bytes
requested if that many bytes are not currently available, or zero (0) if the end of the stream has been
reached.
        /// </returns>
        public override int Read(byte[] buffer, int offset, int count)
        {
            return _filter.Read(buffer, offset, count);
        }

        /// <summary>
        /// When overridden in a derived class, sets the position within the current stream.
        /// </summary>
        /// <param name="offset">A byte offset relative to the <paramref name="origin" />
parameter.</param>
        /// <param name="origin">A value of type <see cref="T:System.IO.SeekOrigin" /> indicating the
reference point used to obtain the new position.</param>
        /// <returns>
        /// The new position within the current stream.
        /// </returns>
        public override long Seek(long offset, SeekOrigin origin)
        {
            return _filter.Seek(offset, origin);
        }

        /// <summary>
        /// When overridden in a derived class, sets the length of the current stream.
        /// </summary>
        /// <param name="value">The desired length of the current stream in bytes.</param>
        public override void SetLength(long value)
        {
            _filter.SetLength(value);
        }

        /// <summary>
        /// When overridden in a derived class, writes a sequence of bytes to the current stream and
advances the current position within this stream by the number of bytes written.
        /// </summary>
        /// <param name="buffer">An array of bytes. This method copies <paramref name="count" /> bytes
from <paramref name="buffer" /> to the current stream.</param>
        /// <param name="offset">The zero-based byte offset in <paramref name="buffer" /> at which to
begin copying bytes to the current stream.</param>
        /// <param name="count">The number of bytes to be written to the current stream.</param>
        public override void Write(byte[] buffer, int offset, int count)
        {
            string html = Encoding.Default.GetString(buffer);

            //remove whitespace
            html = RegexTags.Replace(html, "> <");
            html = RegexAll.Replace(html, " ");

            byte[] outdata = Encoding.Default.GetBytes(html);

            //write bytes to stream
            _filter.Write(outdata, 0, outdata.GetLength(0));
        }

#endregion Methods
    }
}
```

در این کلاس فشرده سازی (gzip و deflate نیز اعمال شده است) در متد OnActionExecuting ابتدا در خط 24 بررسی می‌شود که آیا درخواست رسیده gzip را پشتیبانی می‌کند یا خیر. در صورت پشتیبانی خروجی صفحه را با استفاده از gzip یا deflate فشرده سازی می‌کند. تا اینجای کار ممکن است مورد نیاز ما نباشد. اصل کار ما (حذف کردن فضاهای خالی) در خط 42 اعمال شده است. در واقع برای حذف فضاهای خالی باید یک کلاس که از Stream ارث بری دارد تعریف شده و خروجی کلاس مورد نظر به فیلتر درخواست ما اعمال شود.
در کلاس WhitespaceFilter با تحریف متد Write الگوهای فضای خالی موجود در درخواست یافت شده و آنها را حذف می‌کنیم. در نهایت خروجی این کلاس که از نوع استریم است به ویژگی فیلتر صفحه اعمال می‌شود.

برای معرفی فیلتر تعریف شده می‌توان در فایل Global.asax در رویداد Application_Start به صورت زیر فیلتر مورد نظر را به فیلترهای MVC اعمال کرد.

```
GlobalFilters.Filters.Add(new CompressAttribute());
```

برای آشنایی بیشتر فیلترها در ASP.NET MVC را مطالعه نمایید.

پ.ن: جهت سهولت، در این کلاس ها، صفحات فشرده سازی و همزمان فضاهای خالی آنها حذف شده است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۲ ۲۰:۵۹

با تشکر. من چندبار سعی کردم از روش حذف فواصل خالی استفاده کنم ولی هربار از خیرش گذشتم به این دلایل:
- در مرورگرهای قدیمی گاها باعث کرش و بسته شدن آنی برنامه می‌شد.
- کدهای جاوا اسکریپت یا CSS اگر داخل صفحه قرار داشتند، مشکل پیدا می‌کردند.
- گاهی از همین فضاهای خالی برای اندکی ایجاد فاصله بین عناصر ممکن است استفاده شود. این‌ها با حذف فواصل خالی به هم می‌ریزند.
- بعضی مرورگرها علاقمند هستند که doctype ابتدای یک فایل HTML، حتما در یک سطر مجزا ذکر شود.
- زمانیکه شما codeایی در صفحه تعریف می‌کنید، برای پردازش صحیح تگ PRE توسط مرورگر، مهم است که سطر جدیدی وجود داشته باشد، یا فاصله بین عناصر حفظ شود. در غیراینصورت کد نمایش داده شده به هم می‌ریزد.
- الگوریتم‌های فشرده سازی اطلاعات مانند GZIP یا Deflate، حداقل کاری را که به خوبی انجام می‌دهند، فشرده سازی فواصل خالی است.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۹/۲۲ ۲۱:۵

بله کاملا حق با شماست و مشکل زمانی زیاد می‌شود که در صفحه کد js داشته باشیم و یکی از خطوط با استفاده از // کامنت کنیم.
با فشرده سازی دستورات بعدی کامنت خواهد شد و تا حدودی ممکن است صفحه از کار بیفتد.
که باید حتی الامکان از این نوع کامنت‌ها استفاده نشود.
در هر صورت از نظر شما متشکرم

نویسنده: میثم هوشمند
تاریخ: ۱۳۹۲/۰۹/۲۳ ۰:۳۱

با سلام
می‌شود این استثناها را در فشرده سازی لحاظ کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۳ ۱:۱۲

البته فشرده سازی متفاوت است با حذف فواصل خالی بین تگ‌ها و سطرهای جدید. در حذف فواصل مثلا می‌شود تگ Pre را لحاظ نکرد:

```
var regex = new Regex(@"(?<=\s)\s+(?![^<>]*</pre>)");
```

عنوان: **لغو اجرای یک اکشن فیلتر برای یک اکشن خاص در MVC**
نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۳/۱۱/۰۷ ۲۲:۵
آدرس: www.dotnettips.info
گروه‌ها: MVC, ASP.Net MVC, Attributes, ActionFilters

بسیار پیش می‌آید که یک کنترلر را به یک اکشن فیلتر خاص مزین کنیم. در این صورت تمامی اکشن‌های موجود در کنترلر مربوطه مجاب به اجرای اکشن فیلتر مورد نظر می‌شوند. اما بسیار پیش می‌آید که نخواهیم یک اکشن خاص در کنترلر مذکور اکشن فیلتر مورد نظر را اجرا کند.
یک راهکار ساده اما (به نظر شخصی من) غیر منطقی این است که تک تک اکشن هایی را که می‌خواهیم اکشن فیلتر مورد نظر را اجرا کنند، مزین کنیم و اکشن‌هایی که نمی‌خواهیم اکشن فیلتر مورد نظر را اجرا کنند به اکشن فیلتر مورد نظر مزین نمی‌کنیم. اما فرض کنید تعداد اکشن‌های ما زیاد باشند؛ به نظر این روش غیر منطقی و غیر بهینه است.

یکی از مشکلاتی که در یکی از پروژه‌ها حدود سه روز وقت من را گرفت همین کار بود. زمانی که از یک تصویر امنیتی جهت مقابله با ربوت‌های استفاده میکردم به این مشکل برخورد کردم. کنترلر من به یک اکشن فیلتر فشرده سازی محتوا مزین بود. در نتیجه اکشنی که تصویر امنیتی را تولید میکرد نیز اکشن فیلتر فشرده سازی را اجرا می‌کرد و پس از فشرده سازی باعث می‌شد که تصویر امنیتی نشان داده نشود، چون فرمت تصویری آن بهم ریخته بود. یک راهکار را که پس از جستجو به آن رسیدم، جهت استفاده‌ی دوستان مطرح می‌کنم.

برای اینکه از اجرای چنین اکشن فیلتر هایی جلوگیری کنیم نیاز است کمی اکشن فیلتر مورد نظر را دستکاری کنیم.
برای این کار باید مراحل زیر را انجام داد:
1- ابتدا یک Attribute خالی را تعریف می‌کنیم.
2- سپس اکشن فیلتر دلخواهی را تعریف کرده و در زمان اجرا بررسی می‌کنیم اگر متد (اکشن) مورد نظر با Attribute تعریفی در مرحله یک مزین شده بود، در نتیجه اکشن فیلتر را اجرا نمی‌کنیم.
3- هر اکشنی را که نمی‌خواهیم اکشن فیلتر تعریفی مرحله 2 را اجرا کند، آن را به Attribute مرحله یک مزین میکنیم.
به این ترتیب می‌توانیم از اجرای اکشن فیلتر دلخواه روی متدها یا اکشن‌های دلخواه جلوگیری کنیم. در ادامه نحوه‌ی تعریف آنها را در زیر مشاهده میکنید.

1- تعریف یک Attribute دلخواه مثلا با نام DisableCompression

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = false, Inherited =
true)]
public sealed class DisableCompression : Attribute { }
```

2- تعریف اکشن فیلتر دلخواه مثلا با نام CompressionFilter

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, Inherited = true, AllowMultiple =
false)]
public class CompressionFilter : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        bool disabled = filterContext.ActionDescriptor.IsDefined(typeof(DisableCompression), true) ||

filterContext.ActionDescriptor.ControllerDescriptor.IsDefined(typeof(DisableCompression), true);
        if (disabled)
            return;

        // کدهای دلخواه اکشن فیلتر مورد نظر
    }
}
```

3- در این مرحله هر اکشنی را که نمی‌خواهیم اکشن فیلتر CompressionFilter را اجرا کند به Attribute با نام

DisableCompression مزین میکنیم.

```
[CompressionFilter]
public abstract class BaseController : Controller
{
}

public class SomeController : BaseController
{
    public ActionResult WantThisActionCompressed()
    {
        // code
    }

    [DisableCompression]
    public ActionResult DontWantThisActionCompressed()
    {
        // code
    }
}
```

با این کار اکشن WantThisActionCompressed اکشن فیلتر CompressionFilter را اجرا می‌کند اما اکشن DontWantThisActionCompressed چون مزین به DisableCompression شده‌است، پس در نتیجه اکشن فیلتر CompressionFilter بر روی آن اجرا نخواهد شد.