عنوان: ASP.NET MVC #15
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۱/۲۶ ۲۳:۱۹:۰۰
آدرس: www.dotnettips.info
گروه‌ها: MVC

فیلترها در ASP.NET MVC

پایه قسمت‌های بعدی مانند مباحث امنیت، اعتبار سنجی کاربران، caching و غیره، مبحثی است به نام فیلترها در ASP.NET MVC. تابحال با سه فیلتر به نام‌های NonAction، ActionName و AcceptVerbs آشنا شده‌ایم. به این‌ها Action selector filters هم گفته می‌شود. زمانیکه قرار است یک درخواست رسیده به متدی در یک کنترلر خاص نگاشت شود، فریم ورک ابتدا به متادیتای اعمالی به متدها توجه کرده و بر این اساس درخواست را به متدی صحیح هدایت خواهد کرد. ActionName، نام پیش فرض یک متد را بازنویسی می‌کند و توسط AcceptVerbs اجرای یک متد، به افعالی مانند DELETE، GET، POST و امثال آن محدود می‌شود که در قسمت‌های قبل در مورد آن‌ها بحث شد.
علاوه بر این‌ها یک سری فیلتر دیگر نیز در ASP.NET MVC وجود دارند که آن‌ها نیز به شکل متادیتا به متدهای کنترلرها اعمال شده و کار نهایی‌اشان تزریق کدهایی است که باید پیش و پس از اجرای یک اکشن متد، اجرا شوند. 4 نوع فیلتر در ASP.NET MVC وجود دارند:

الف) IAuthorizationFilter
این نوع فیلترها پیش از اجرای هر متد یا فیلتر دیگری در کنترلر جاری اجرا شده و امکان لغو اجرای آن‌را فراهم می‌کنند. پیاده سازی پیش‌فرض آن توسط کلاس AuthorizeAttribute در فریم ورک وجود دارد.
بدیهی است این نوع اعمال را مستقیما داخل متدهای کنترلرها نیز می‌توان انجام داد (بدون نیاز به هیچگونه فیلتری). اما به این ترتیب حجم کدهای تکراری در سراسر برنامه به شدت افزایش می‌یابد و نگهداری آن‌را در طول زمان مشکل خواهد ساخت.

ب) IActionFilter
ActionFilterها پیش (OnActionExecuting) و پس از (OnActionExecuted) اجرای متدهای کنترلر جاری اجرا می‌شوند و همچنین پیش از ارائه خروجی نهایی متدها. به این ترتیب برای مثال می‌توان نحوه رندر یک View را تحت کنترل گرفت. این اینترفیس توسط کلاس ActionFilterAttribute در فریم ورک پیاده سازی شده است.

ج) IResultFilter
ResultFilter بسیار شبیه به ActionFilter است با این تفاوت که تنها پیش از (OnResultExecuting) بازگرداندن نتیجه متد و همچنین پس از (OnResultExecuted) اجرای متد، فراخوانی می‌گردد. کلاس ActionFilterAttribute موجود در فریم ورک، پیاده سازی پیش فرضی از آن‌را ارائه می‌دهد.

د) IExceptionFilter
ExceptionFilterها پس از اجرای تمامی فیلترهای دیگر، همواره اجرا خواهند شد؛ صرفنظر از اینکه آیا در این بین استثنایی رخ داده است یا خیر. بنابراین یکی از کاربردهای آن‌ها می‌تواند ثبت وقایع مرتبط با استثناهای رخداده باشد. پیاده سازی پیش فرض آن توسط کلاس HandleErrorAttribute در فریم ورک موجود است.

علت معرفی 4 نوع فیلتر متفاوت هم به مسایل امنیتی بر می‌گردد. می‌شد تنها موارد ب و ج معرفی شوند اما از آنجائیکه نیاز است مورد الف همواره پیش از اجرای متدی و همچنین تمامی فیلترهای دیگر فراخوانی شود، احتمال بروز اشتباه در نحوه و ترتیب معرفی این فیلترها وجود داشت. به همین دلیل روش معرفی صریح مورد الف در پیش گرفته شد. برای مثال فرض کنید که اگر از روی اشتباه فیلتر کش شدن اطلاعات پیش از فیلتر اعتبار سنجی کاربر جاری اجرا می‌شد چه مشکلات امنیتی ممکن بود بروز کند.

مثالی جهت درک بهتر ترتیب و نحوه اجرای فیلترها:

یک پروژه جدید خالی ASP.NET MVC را آغاز کنید. سپس فیلتر سفارشی زیر را به برنامه اضافه نمائید:

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

namespace MvcApplication12.CustomFilters
{
    public class LogAttribute : ActionFilterAttribute
    {
        public override void OnActionExecuting(ActionExecutingContext filterContext)
        {
            Log("OnActionExecuting", filterContext);
        }

        public override void OnActionExecuted(ActionExecutedContext filterContext)
        {
            Log("OnActionExecuted", filterContext);
        }

        public override void OnResultExecuting(ResultExecutingContext filterContext)
        {
            Log("OnResultExecuting", filterContext);
        }

        public override void OnResultExecuted(ResultExecutedContext filterContext)
        {
            Log("OnResultExecuted", filterContext);
        }

        private void Log(string stage, ControllerContext ctx)
        {
            ctx.HttpContext.Response.Write(
                string.Format("{0}:{1} - {2} < br/> ",
                ctx.RouteData.Values["controller"], ctx.RouteData.Values["action"], stage));
        }
    }
}
```

مرسوم است برای ایجاد فیلترهای سفارشی، همانند مثال فوق با ارث بری از پیاده سازی‌های توکار اینترفیس‌های چهارگانه یاد شده، کار شروع شود.
سپس یک کنترلر جدید را به همراه دو متد، به برنامه اضافه نمائید. برای هر کدام از متدها هم یک View خالی را ایجاد کنید. اکنون این ویژگی جدید را به هر کدام از این متدها اعمال نموده و برنامه را اجرا کنید.

```
using System.Web.Mvc;
using MvcApplication12.CustomFilters;

namespace MvcApplication12.Controllers
{
    public class HomeController : Controller
    {
        [Log]
        public ActionResult Index()
        {
            return View();
        }

        [Log]
        public ActionResult Test()
        {
            return View();
        }
    }
}
```

سپس ویژگی Log را از متدها حذف کرده و به خود کنترلر اعمال کنید:

```
[Log]
public class HomeController : Controller
```

در این حالت ویژگی اعمالی، پیش از اجرای متد درخواستی جاری اجرا خواهد شد یا به عبارتی به تمام متدهای قابل دسترسی کنترلر اعمال می‌گردد.

تقدم و تاخر اجرای فیلترهای هم‌خانواده

همانطور که عنوان شد، همیشه ابتدا AuthorizationFilter اجرا می‌شود و در آخر ExceptionFilter. سؤال: اگر در این بین مثلا دو نوع ActionFilter متفاوت به یک متد اعمال شدند، کدامیک ابتدا اجرا می‌شود؟
تمام فیلترها از کلاسی به نام FilterAttribute مشتق می‌شوند که دارای خاصیتی است به نام Order. بنابراین جهت مشخص سازی ترتیب اجرای فیلترها تنها کافی است این خاصیت مقدار دهی شود. برای مثال جهت اعمال دو فیلتر سفارشی زیر:

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

namespace MvcApplication12.CustomFilters
{
    public class AuthorizationFilterA : AuthorizeAttribute
    {
        public override void OnAuthorization(AuthorizationContext filterContext)
        {
            Debug.WriteLine("OnAuthorization : AuthorizationFilterA");
        }
    }
}
```

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

namespace MvcApplication12.CustomFilters
{
    public class AuthorizationFilterB : AuthorizeAttribute
    {
        public override void OnAuthorization(AuthorizationContext filterContext)
        {
            Debug.WriteLine("OnAuthorization : AuthorizationFilterB");
        }
    }
}
```

خواهیم داشت:

```
using System.Web.Mvc;
using MvcApplication12.CustomFilters;

namespace MvcApplication12.Controllers
{
    public class HomeController : Controller
    {
        [AuthorizationFilterA(Order = 2)]
        [AuthorizationFilterB(Order = 1)]
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

در اینجا با توجه به مقادیر order، ابتدا AuthorizationFilterB اجرا می‌گردد و سپس AuthorizationFilterA.
علاوه بر این‌ها محدوده اجرای فیلترها نیز بر بر این حق تقدم اجرایی تاثیر گذار هستند. برای مثال در پشت صحنه زمانیکه قرار است یک فیلتر جدید اجرا شود، وهله سازی آن به نحوه زیر است که بر اساس مقادیر order و FilterScope صورت می‌گیرد:

```
var filter = new Filter(actionFilter, FilterScope, order);
```

مقادیر FilterScope را در ادامه ملاحظه می‌نمائید:

```
namespace System.Web.Mvc {
      public enum FilterScope {
            First = 0,
            Global = 10,
            Controller = 20,
            Action = 30,
            Last = 100,
        }
}
```

به صورت پیش فرض، ابتدا فیلتری با محدوده اجرای کمتر، اجرا خواهد شد. در اینجا Global به معنای اجرای شدن در تمام کنترلرها است.

تعریف فیلترهای سراسری

برای اینکه فیلتری را عمومی و سراسری تعریف کنیم، تنها کافی است آن‌را در متد Application_Start فایل Global.asax.cs به نحو زیر معرفی نمائیم:

```
GobalFilters.Filters.Add(new AuthorizationFilterA() { Order = 2});
```

به این ترتیب AuthorizationFilterA، به تمام کنترلرها و متدهای قابل دسترسی آن‌ها در برنامه به صورت خودکار اعمال خواهد شد.
یکی از کاربردهای فیلترهای سراسری، نوشتن برنامه‌های پروفایلر است. برنامه‌هایی که برای مثال مدت زمان اجرای متدها را ثبت کرده و بر این اساس بهتر می‌توان کارآیی قسمت‌های مختلف برنامه را دقیقا زیرنظر قرار داد.

یک نکته
کلاس کنترلر در ASP.NET MVC نیز یک فیلتر است:

```
public abstract class Controller : ControllerBase, IActionFilter, IAuthorizationFilter, IDisposable,
IExceptionFilter, IResultFilter
```

به همین دلیل، امکان تحریف متدهای OnActionExecuting، OnActionExecuted و امثال آن که پیشتر ذکر شد، در یک کنترلر نیز وجود دارد.
کلاس کنترلر دارای محدوده اجرایی First و Order ایی مساوی Int32.MinValue است. به این ترتیب کنترلرها پیش از اجرای هر فیلتر دیگری اجرا خواهند شد.

ASP.NET MVC دارای یک سری فیلتر و متادیتای توکار مانند ValidateInpute، RequireHttps، HandleError، OutputCache و غیره است که توضیحات بیشتر آن‌ها به قسمت‌های بعد موکول می‌گردد.

نظرات خوانندگان

نویسنده: محمد شیران
تاریخ: ۱۳۹۱/۰۱/۲۷ ۰۸:۱۸:۰۷

آقای نصیری واقعا از شما سپاسگذاریم. میدونیم که جدا از تجربیات ارزندتون که ساعتها برای کسبش وقت و تلاش گذاشتین نوشتن هر کدوم از این مطالب هم خودش ساعتها وقت میگیره. نمیدونید چه خدمتی دارید به من و امثال من میکنید.
من خودم همین الان درگیر یه پروژه بزرگ MVC هستم. این سری مطالب شما واقعا بهم داره کمک میکنه.

نویسنده: Naser Tahery
تاریخ: ۱۳۹۱/۰۱/۲۹ ۲۳:۵۱:۳۱

ممنون از مطالب عالی
آیا این امکان هست داخل پوشه های مهم (پوشه هایی که توسط Areas تعریف شد)که نیاز به لاگین کردن کاربر داره یک فایل کانفیگ قرار داد و رول ها رو جهت دسترسی به فایل های داخل این پوشه معرفی کرد؟ که دیگه نیاز به این فیلتر اعتبار سنجی کاربر پشت هر اکشنی نباشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۲/۰۱ ۰۹:۲۶:۳۹

در فایل کانفیگ اصلی برنامه (در ریشه اصلی سایت) می‌تونید با مقدار دهی تگ location اینکار رو انجام بدید. ولی روش توصیه شده‌ای برای ASP.NET MVC نیست. استفاده از فیلتر Authorize روش امن‌تری است چون مستقل از تعاریف مسیریابی عمل می‌کند و امکان فراموشی آن پس از تغییرات در مسیریابی نیست.

نویسنده: asrin
تاریخ: ۱۳۹۱/۰۶/۲۹ ۱۸:۴۰

من دارم رو یه پروژه بزرگ با MVC کار می‌کنم.می خواستم ازتون بپرسم که گذاشتن این فیلترها در برنامه ضروریه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۲۹ ۱۹:۱۲

این قسمت پایه تئوری قسمت‌های بعدی است.
در قسمت‌های بعد با مباحث امنیتی و فیلترهای مربوطه آشنا خواهید شد که استفاده از آن‌ها در هر برنامه‌ای ضروری است.

نویسنده: sara zarmehr
تاریخ: ۱۳۹۱/۱۰/۱۷ ۲۱:۳۷

فیلتر سفارشی رو چگونه ساختید که namespace این گونه نوشته شده؟

```
namespace MvcApplication12.CustomFilters
{
    .....
    .....
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۱۷ ۲۱:۴۴

داخل یک پوشه به نام CustomFilters. سورس‌های کامل این سری در دسترس است .