< Previous Next >

ASP.NET MVC- Filters

In ASP.NET MVC, a user request is routed to the appropriate controller and action method. However, there may be circumstances where you want to execute some logic before or after an action method executes. ASP.NET MVC provides filters for this purpose.

ASP.NET MVC Filter is a custom class where you can write custom logic to execute before or after an action method executes. Filters can be applied to an action method or controller in a declarative or programmatic way. Declarative means by applying a filter attribute to an action method or controller class and programmatic means by implementing a corresponding interface.

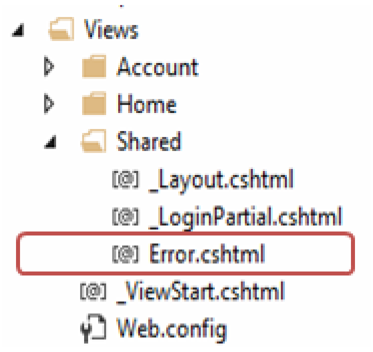
MVC provides different types of filters. The following table list filter types, built-in filters for the type and interface which must be implemented to create a custom filter class.

Filter Type	Description	Built-in Filter	Interface
Authorization filters	Performs authentication and authorizes before executing action method.	[Authorize], [RequireHttps]	IAuthorizationFilter
Action filters	Performs some operation before and after an action method executes.		IActionFilter

Filter Type	Description	Built-in Filter	Interface
Result filters	Performs some operation before or after the execution of view result.	[OutputCache]	IResultFilter
Exception filters	Performs some operation if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline.	[HandleError]	IExceptionFilter

To understand the filter in detail, let's take an example of built-in Exception filter.

An exception filter executes when there is an unhandled exception occurs in your application. HandleErrorAttribute ([HandlerError]) class is a built-in exception filter class in MVC framework. This built-in HandleErrorAttribute class renders Error.cshtml included in the Shared folder by default, when an unhandled exception occurs.



Error.cshtml

The following example demonstrates built-in exception filter HandErrorAttribute.

Example: Authorization Filter

```
[HandleError]
public class HomeController : Controller
{
    public ActionResult Index()
```

```
        {
            //throw exception for demo
            throw new Exception("This is unhandled exception");

            return View();
        }

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

        public ActionResult Contact()
        {
            return View();
        }
    }
```

In the above example, we have applied `[HandleError]` attribute to HomeController. So now it will display Error page if any action method of HomeController would throw unhandled exception. Please note that unhandled exception is an exception which is not handled by the try-catch block.

TIPS

Every attribute class must end with Attribute e.g. HanderErrorAttribute. Attribute must be applied without Attribute suffix inside square brackets [ ] like [HandelError].

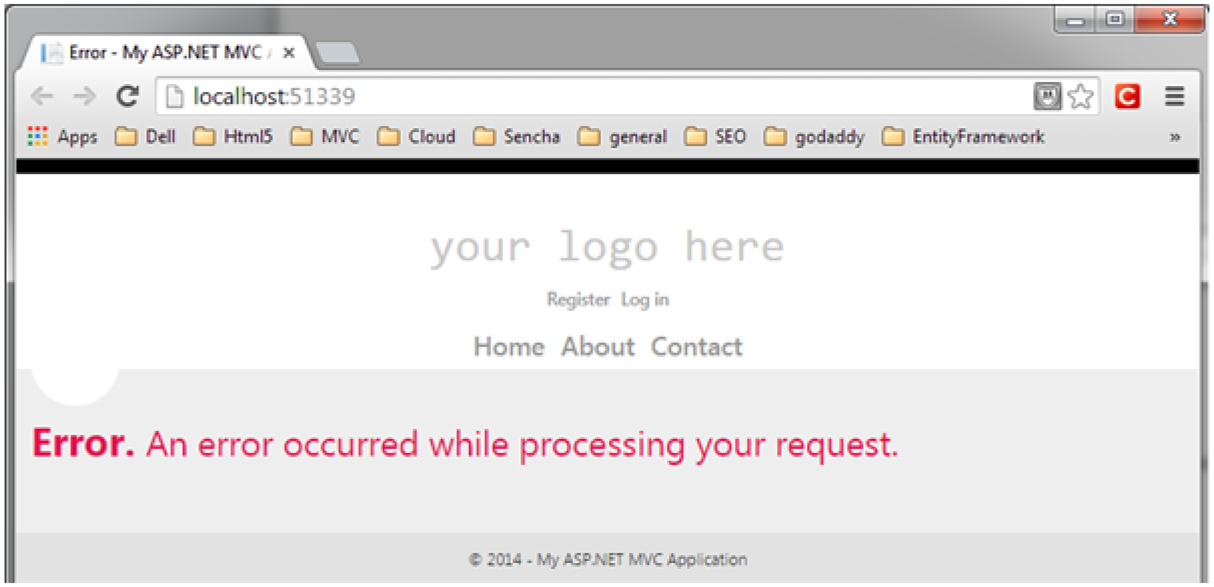
Filters applied to the controller will automatically be applicable to all the action methods of a controller.

Please make sure that CustomError mode is on in System.web section of web.config, in order for HandleErrorAttribute work properly.

Example: SetCustomError in web.config

```
<customErrors mode="On" />
```

Now, if you run the application. You would get following error page because we throw exception in Index action method for the demo purpose.



Error. An error occurred while processing your request.

© 2014 - My ASP.NET MVC Application

HandleError Demo

Thus, HandleError attribute will display common error page for any unhandled exception occurred in HomeController.

Register Filters

Filters can be applied at three levels.

1. Global Level

You can apply filters at global level in the Application_Start event of Global.asax.cs file by using default FilterConfig.RegisterGlobalFilters() mehtod. Global filters will be applied to all the controller and action methods of an application.

The [HandleError] filter is applied globaly in MVC Application by default in every MVC application created using Visual Studio as shown below.

Example: Register Global Filters

```
// MvcApplication class contains in Global.asax.cs file
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    }
}

// FilterConfig.cs located in App_Start folder
public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
    }
}
```

2. Controller level

Filters can also be applied to the controller class. So, filters will be applicable to all the action method of Controller class if it is applied to a controller class.

Example: Action Filters on Controller

```
[HandleError]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

}
```

3. Action method level

You can apply filters to an individual action method also. So, filter will be applicable to that particular action method only.

Example: Filters on Action Method

```
public class HomeController : Controller
{
    [HandleError]
    public ActionResult Index()
    {
        return View();
        }

    }
```

The same way, you can apply multiple built-in or custom filters globally or at controller or action method level for different purpose such as [Authorize],[RequireHttps], [ChildActionOnly],[OutputCache],[HandleError].

Filter Order

As mentioned above, MVC includes different types of filters and multiple filters can be applied to a single controller class or action method. So, filters run in the following order.

1. Authorization filters
2. Action filters
3. Response filters
4. Exception filters

Create Custom Filter

You can create custom filter attributes by implementing an appropriate filter interface for which you want to create a custom filter and also derive a FilterAttribute class so that you can use that class as an attribute.

For example, implement IExceptionFilter and FilterAttribute class to create custom exception filter. In the same way implement an IAuthorizatinFilter interface and FilterAttribute class to create a custom authorization filter.

Example: Custom Exception Filter

```
class MyErrorHandler : FilterAttribute, IExceptionFilter
{
    public override void IExceptionFilter.OnException(ExceptionContext filterContext)
    {
        Log(filterContext.Exception);

        base.OnException(filterContext);
    }

    private void Log(Exception exception)
    {
        //log exception here..

    }
}
```

Alternatively, you can also derive a built-in filter class and override an appropriate method to extend the functionality of built-in filters.

Let's create custom exception filter to log every unhandled exception by deriving built-in HandleErrorAttribute class and overriding OnException method as shown below.

Example: Custom Exception Filter

```
class MyErrorHandler : HandleErrorAttribute
{
    public override void OnException(ExceptionContext filterContext)
    {
        Log(filterContext.Exception);
```

```
            base.OnException(filterContext);
        }

        private void Log(Exception exception)
        {
            //log exception here..

        }
    }
```

Now, you can apply MyErrorHandler attribute at global level or controller or action method level, the same way we applied the HandleError attribute.

Example: Custom Action Filters to Controller

```
[MyErrorHandler]
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

}
```

Points to Remember :

1) MVC Filters are used to execute custom logic before or after executing action method.

2) MVC Filter types:

  1) Authorization filters

  2) Action filters

  3) Result filters

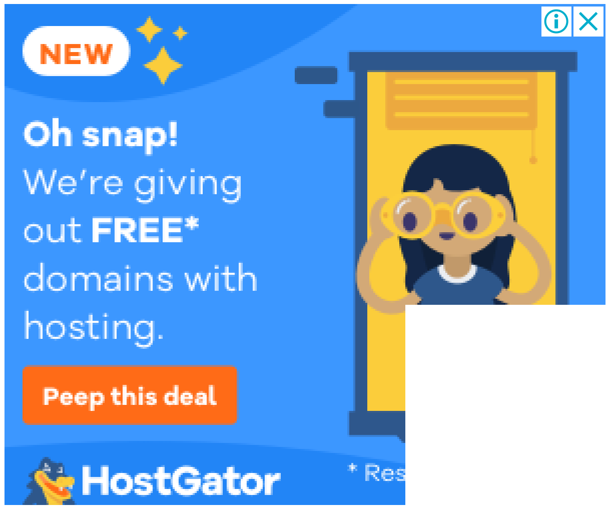
  4) Exception filters

3) Filters can be applied globally in FilterConfig class, at controller level or action method level.

4) Custom filter class can be created by implementing FilterAttribute class and corresponding interface.

Share Tweet

Share Whatsapp



Previous Next

TUTORIALSTEACHER.COM

TutorialsTeacher.com is optimized for learning web technologies step by step. Examples might be simplified to improve reading and basic understanding. While using this site, you agree to have read and accepted our terms of use and privacy policy.

feedback@tutorialsteacher.com

TUTORIALS

- ASP.NET Core
- ASP.NET MVC
- IoC
- Web API
- C#
- LINQ
- Entity Framework
- AngularJS 1
- Node.js
- D3.js
- JavaScript
- jQuery
- Sass
- Https

E-MAIL LIST

Subscribe to TutorialsTeacher email list and get latest updates, tips & tricks on C#, .Net, JavaScript, jQuery, AngularJS, Node.js to your inbox.

Email address GO

We respect your privacy.

HOME PRIVACY POLICY TERMS OF USE ADVERTISE WITH US