ASP.NET MVC provides a simple way to inject your piece of code or logic either before or after an action is executed. This is achieved by decorating the controllers or actions with ASP.NET MVC attributes or custom attributes. An attribute or custom attribute implements the ASP.NET MVC filters(filter interface) and can contain your piece of code or logic. You can make your own custom filters or attributes either by implementing ASP.NET MVC filter interface or by inheriting and overriding methods of ASP.NET MVC filter attribute classes

When to use Filters?

Typically, Filters are used to perform the following common functionalities in your ASP.NET MVC application.

1. Custom Authentication
2. Custom Authorization(User based or Role based)
3. Error handling or logging
4. User Activity Logging
5. Data Caching

Types of Filters

The ASP.NET MVC framework provides five types of filters.

1. Authentication filters (MVC5)
2. Authorization filters
3. Action filters
4. Result filters
5. Exception filters

Authentication Filters

This filter is introduced with ASP.NET MVC5. The IAuthenticationFilter interface is used to create CustomAuthentication filter. The definition of this interface is given below-

1. public interface IAuthenticationFilter
2. {
3. void OnAuthentication(AuthenticationContext filterContext);
5. void OnAuthenticationChallenge(AuthenticationChallengeContext filterContext);
6. }

You can create your CustomAuthentication filter attribute by implementing IAuthenticationFilter as shown below-

1. public class CustomAuthenticationAttribute : ActionFilterAttribute, IAuthenticationFilter
2. {
3. public void OnAuthentication(AuthenticationContext filterContext)
4. {
5. *//Logic for authenticating a user*
6. }
7. *//Runs after the OnAuthentication method*
8. public void OnAuthenticationChallenge(AuthenticationChallengeContext filterContext)
9. {
10. *//TODO: Additional tasks on the request*
11. }
12. }

Authorization Filters

The ASP.NET MVC Authorize filter attribute implements the IAuthorizationFilter interface. The definition of this interface is given below-

1. public interface IAuthorizationFilter
2. {
3. void OnAuthorization(AuthorizationContext filterContext);
4. }

The AuthorizeAttribute class provides the following methods to override in the CustomAuthorize attribute class.

1. public class AuthorizeAttribute : FilterAttribute, IAuthorizationFilter
2. {
3. protected virtual bool AuthorizeCore(HttpContextBase httpContext);
4. protected virtual void HandleUnauthorizedRequest(AuthorizationContext filterContext);
5. public virtual void OnAuthorization(AuthorizationContext filterContext);
6. protected virtual HttpValidationStatus OnCacheAuthorization(HttpContextBase httpContext);
7. }

In this way you can make your CustomAuthorize filter attribute either by implementing IAuthorizationFilter interface or by inheriting and overriding above methods of AuthorizeAttribute class.

Action Filters

Action filters are executed before or after an action is executed. The IActionFilter interface is used to create an Action Filter which provides two methods OnActionExecuting and OnActionExecuted which will be executed before or after an action is executed respectively.

1. public interface IActionFilter
2. {
3. void OnActionExecuting(ActionExecutingContext filterContext);
4. void OnActionExecuted(ActionExecutedContext filterContext);
5. }

Result Filters

Result filters are executed before or after generating the result for an action. The Action Result type can be ViewResult, PartialViewResult, RedirectToRouteResult, RedirectResult, ContentResult, JsonResult, FileResult and EmptyResult which derives from the ActionResult class. Result filters are called after the Action filters. The IResultFilter interface is used to create an Result Filter which provides two methods OnResultExecuting and OnResultExecuted which will be executed before or after generating the result for an action respectively.

1. public interface IResultFilter
2. {
3. void OnResultExecuted(ResultExecutedContext filterContext);
4. void OnResultExecuting(ResultExecutingContext filterContext);
5. }

Exception Filters

Exception filters are executed when exception occurs during the actions execution or filters execution. The IExceptionFilter interface is used to create an Exception Filter which provides OnException method which will be executed when exception occurs during the actions execution or filters execution.

1. public interface IExceptionFilter
2. {
3. void OnException(ExceptionContext filterContext);
4. }

ASP.NET MVC HandleErrorAttribute filter is an Exception filter which implements IExceptionFilter. When HandleErrorAttribute filter receives the exception it returns an Error view located in the Views/Shared folder of your ASP.NET MVC application.

Order of Filter Execution

All ASP.NET MVC filter are executed in an order. The correct order of execution is given below:

1. Authentication filters
2. Authorization filters
3. Action filters
4. Result filters

Configuring Filters

You can configure your own custom filter into your application at following three levels:

1. Global level

By registering your filter into Application\_Start event of Global.asax.cs file with the help of FilterConfig class.

* 1. protected void Application\_Start()
  2. {
  3. FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
  4. }

1. Controller level

By putting your filter on the top of the controller name as shown below-

* 1. [Authorize(Roles="Admin")]
  2. public class AdminController : Controller
  3. {
  4. *//*
  5. }

1. Action level

By putting your filter on the top of the action name as shown below-

* 1. public class UserController : Controller
  2. {
  3. [Authorize(Users="User1,User2")]
  4. public ActionResult LinkLogin(string provider)
  5. {
  6. *// TODO:*
  7. return View();
  8. }
  9. }