

Exploring types of

hii

# Filters in .NET

hii

Action Filter

Resource Filter

Exception Filter

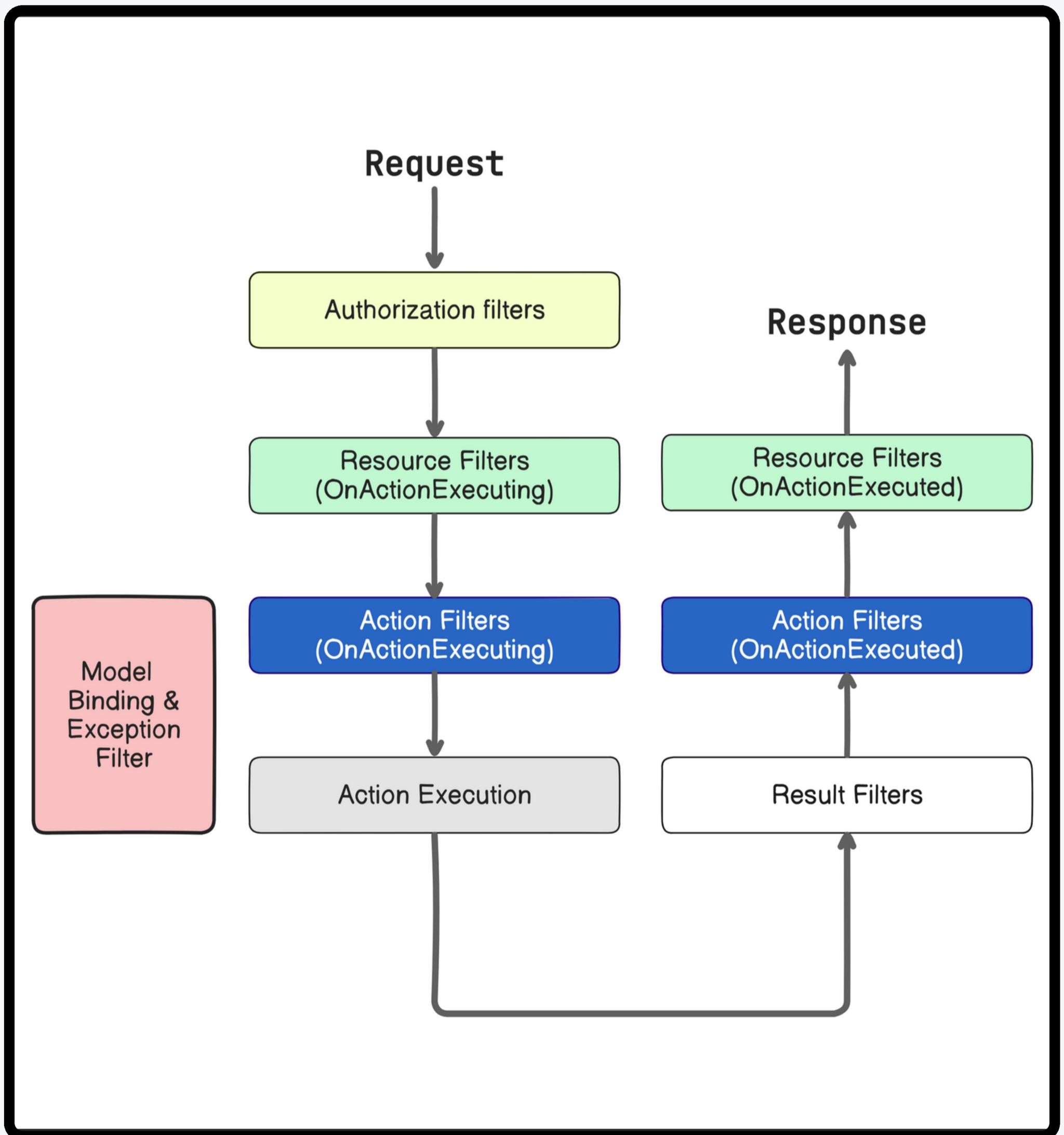
Result Filter

Authorization Filter

There are primarily 5  
types of filters  
in .NET



# Request & Response Pipeline with .NET Filters



# OVERVIEW



FILTER NAME	EXECUTION ORDER	USE CASE
<b>Authorization Filters</b>	Before anything else (before resource filters).	Authentication & authorization.
<b>Resource Filters</b>	After authorization, before model binding & action execution.	Caching, logging, performance.
<b>Action Filters</b>	Before and after action execution.	Modify action execution, logging.
<b>Exception Filters</b>	Only when an unhandled exception occurs.	Global exception handling.
<b>Result Filters</b>	Before and after the result (view or JSON response) is processed.	Modify or log responses.



# Authorization Filter

- Implemented using `IAuthorizationFilter` or `IAsyncAuthorizationFilter`.
- Executes first in the pipeline, before any other filters or processing.
- Ensures users are authenticated and authorized before accessing a resource.
- Short-circuits the request if authorization fails, returning `401 Unauthorized` or `403 Forbidden`.

```
1 public class CustomAuthorizationFilter : Attribute, IAuthorizationFilter
2 {
3     public void OnAuthorization(AuthorizationFilterContext context)
4     {
5         var user = context.HttpContext.User;
6         if (!user.Identity.IsAuthenticated)
7         {
8             context.Result = new UnauthorizedResult(); // 401 Unauthorized
9         }
10    }
11 }
12
13 // Apply filter at the controller or action level
14 [CustomAuthorizationFilter]
15 public class SecureController : Controller
16 {
17     public IActionResult SecureData()
18     {
19         return View();
20     }
21 }
```



# Action Filter

- Implemented using `IActionFilter` or `IAsyncActionFilter`.
- Executes before and after the action method in the pipeline.
- Used for logging, validation, modifying action parameters, and measuring execution time.
- Allows modifying action behavior dynamically before it executes or after it returns a response.

```
1 public class ExecutionTimeActionFilter : Attribute, IActionFilter
2 {
3     private Stopwatch _stopwatch;
4
5     public void OnActionExecuting(ActionExecutingContext context)
6     {
7         _stopwatch = Stopwatch.StartNew();
8         Console.WriteLine("Action execution started.");
9     }
10
11    public void OnActionExecuted(ActionExecutedContext context)
12    {
13        _stopwatch.Stop();
14        Console.WriteLine($"Action executed in {_stopwatch.ElapsedMilliseconds} ms");
15    }
16 }
17
18 // Applying the filter on a controller
19 [ExecutionTimeActionFilter]
20 public class ProductsController : Controller
21 {
22     public IActionResult List()
23     {
24         return View();
25     }
26 }
```



# Exception Filters

- Implemented using `IExceptionFilter` or `IAsyncExceptionFilter`.
- Executes only if an unhandled exception occurs in the pipeline.
- Used for centralized error handling, logging, and custom error responses.
- Can replace the default error response with a custom message or status code.

```
1 public class CustomExceptionFilter : Attribute, IExceptionFilter
2 {
3     public void OnException(ExceptionContext context)
4     {
5         Console.WriteLine($"Exception occurred: {context.Exception.Message}");
6
7         context.Result = new ContentResult
8         {
9             Content = "An unexpected error occurred. Please try again later.",
10            StatusCode = 500
11        };
12        context.ExceptionHandled = true; // Mark exception as handled
13    }
14 }
15
16 // Applying the filter on a controller
17 [CustomExceptionFilter]
18 public class OrderController : Controller
19 {
20     public IActionResult ProcessOrder()
21     {
22         throw new Exception("Order processing failed!");
23     }
24 }
```



# Result Filters

- Implemented using `IResultFilter` or `IAsyncResultFilter`.
- Executes before and after the result (view or JSON response) is processed.
- Used for modifying response data, adding headers, or logging response details.
- Ensures that response data is consistent, formatted, or secured before being sent to the client.

```
1 public class CustomResultFilter : Attribute, IResultFilter
2 {
3     public void OnResultExecuting(ResultExecutingContext context)
4     {
5         Console.WriteLine("Before result is processed.");
6     }
7
8     public void OnResultExecuted(ResultExecutedContext context)
9     {
10        Console.WriteLine("After result is processed.");
11    }
12 }
13
14 // Applying the filter on a controller
15 [CustomResultFilter]
16 public class ReportsController : Controller
17 {
18     public IActionResult Summary()
19     {
20         return View();
21     }
22 }
```



# Resource Filters

- Implemented using `IResourceFilter` or `IAsyncResourceFilter`.
- Executes after authorization filters but before model binding and action execution.
- Used for caching, logging, and modifying requests before the action method runs.
- Can short-circuit requests if a cached response is available, improving performance.

```
2
3  public class CustomResourceFilter : Attribute, IResourceFilter
4  {
5      public void OnResourceExecuting(ResourceExecutingContext context)
6      {
7          Console.WriteLine("Before executing action - Resource Filter.");
8      }
9
10     public void OnResourceExecuted(ResourceExecutedContext context)
11     {
12         Console.WriteLine("After executing action - Resource Filter.");
13     }
14 }
15
16 // Applying the filter on a controller
17 [CustomResourceFilter]
18 public class HomeController : Controller
19 {
20     public IActionResult Index()
21     {
22         return View();
23     }
24 }
```

