Exploring types of

Filters in .NET

Action Filter

Resource Filter

Exception Filter

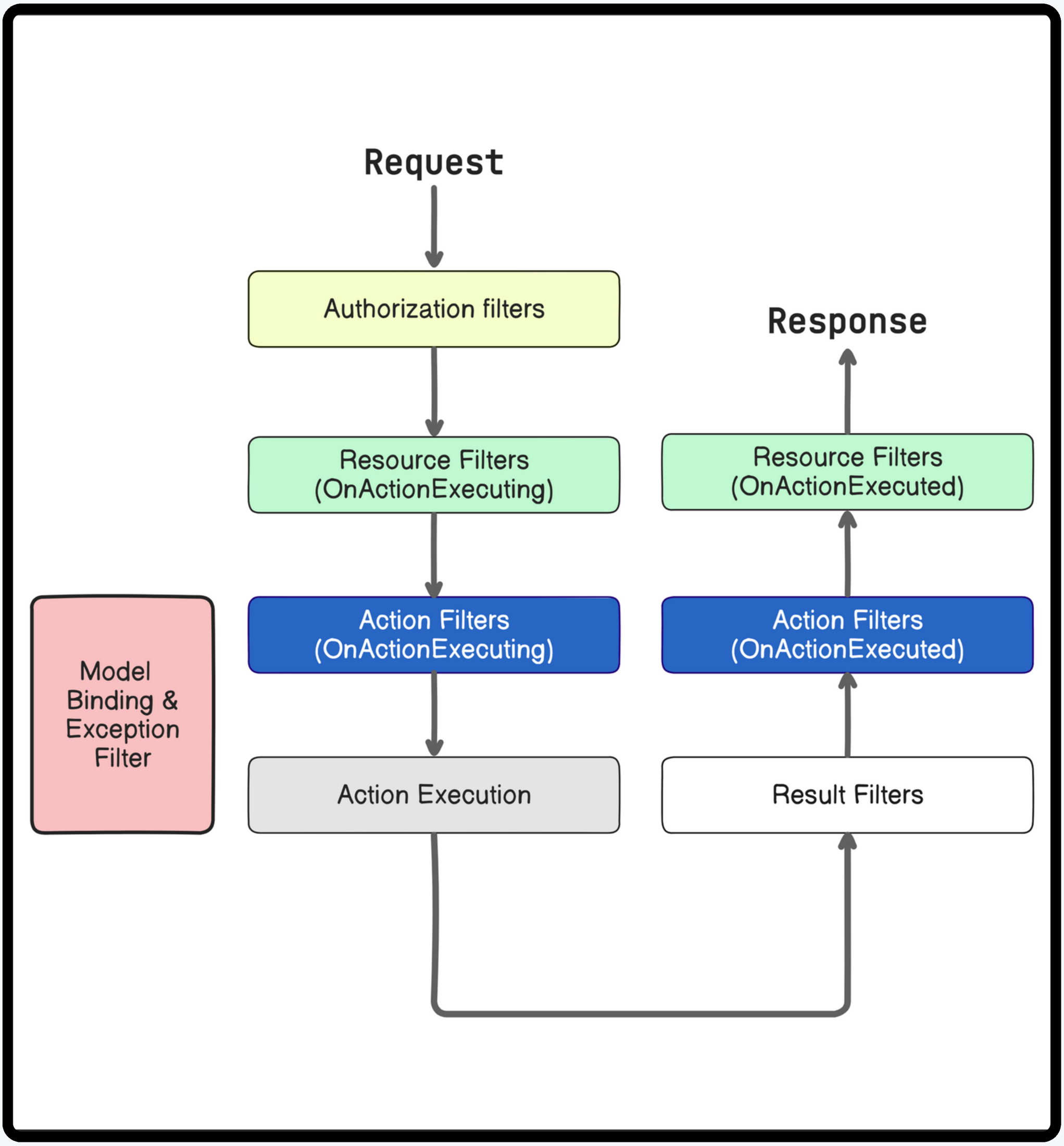
Result Filter

Authorization Filter

There are primarily 5 types of filters in .NET

SHREYANS PADMANI

Request & Response Pipeline with .NET Filters



SHREYANS PADMANI

OVERVIEW

FILTER NAME	EXECUTION ORDER	USE CASE
Authorization Filters	Before anything else (before resource filters).	Authentication & authorization.
Resource Filters	After authorization, before model binding & action execution.	Caching, logging, performance.
Action Filters	Before and after action execution.	Modify action execution, logging.
Exception Filters	Only when an unhandled exception occurs.	Global exception handling.
Result Filters	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.

```
public class CustomAuthorizationFilter : Attribute, IAuthorizationFilter
{
    public void OnAuthorization(AuthorizationFilterContext context)
    {
        var user = context.HttpContext.User;
        if (!user.Identity.IsAuthenticated)
        {
            context.Result = new UnauthorizedResult(); // 401 Unauthorized
        }
    }
}

// Apply filter at the controller or action level
[CustomAuthorizationFilter]
public class SecureController : Controller
{
    public IActionResult SecureData()
    {
        return View();
    }
}
```

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.

```
public class ExecutionTimeActionFilter : Attribute, IActionFilter
{
    private Stopwatch _stopwatch;

    public void OnActionExecuting(ActionExecutingContext context)
    {
        _stopwatch = Stopwatch.StartNew();
        Console.WriteLine("Action execution started.");
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        _stopwatch.Stop();
        Console.WriteLine($"Action executed in {_stopwatch.ElapsedMilliseconds} ms");
    }
}

// Applying the filter on a controller
[ExecutionTimeActionFilter]
public class ProductsController : Controller
{
    public IActionResult List()
    {
        return View();
    }
}
```

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.

```
public class CustomExceptionFilter : Attribute, IExceptionFilter
{
    public void OnException(ExceptionContext context)
    {
        Console.WriteLine($"Exception occurred: {context.Exception.Message}");

        context.Result = new ContentResult
        {
            Content = "An unexpected error occurred. Please try again later.",
            StatusCode = 500
        };
        context.ExceptionHandled = true; // Mark exception as handled
    }
}

// Applying the filter on a controller
[CustomExceptionFilter]
public class OrderController : Controller
{
    public IActionResult ProcessOrder()
    {
        throw new Exception("Order processing failed!");
    }
}
```

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.

```
public class CustomResourceFilter : Attribute, IResourceFilter
{
    public void OnResourceExecuting(ResourceExecutingContext context)
    {
        Console.WriteLine("Before executing action - Resource Filter.");
    }

    public void OnResourceExecuted(ResourceExecutedContext context)
    {
        Console.WriteLine("After executing action - Resource Filter.");
    }
}

// Applying the filter on a controller
[CustomResourceFilter]
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```