Aspect oriented programming with ASP.NET Core

This post is about implementing simple AOP (Aspect Oriented Programming) with ASP.NET Core. AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does so by adding additional behavior to existing code (an advice) without modifying the code itself. An example of crosscutting concerns is “logging,” which is frequently used in distributed applications to aid debugging by tracing method calls. AOP helps you to implement logging without affecting you actual code.

To add AOP to the controllers, first you need to create ActionFilter class, which should inherit from `ActionFilterAttribute` class and need to override methods. Here is minimal implementation of Logging filter.

```
public class LoggingActionFilter : ActionFilterAttribute
{
    private readonly ILogger _logger;
    public LoggingActionFilter(ILoggerFactory loggerFactory)
    {
        _logger = loggerFactory.CreateLogger("LoggingActionFilter");
    }
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        _logger.LogInformation("ClassFilter OnActionExecuting");
        base.OnActionExecuting(context);
    }

    public override void OnActionExecuted(ActionExecutedContext context)
    {
        _logger.LogInformation("ClassFilter OnActionExecuted");
        base.OnActionExecuted(context);
    }

    public override void OnResultExecuting(ResultExecutingContext context)
    {
        _logger.LogInformation("ClassFilter OnResultExecuting");
        base.OnResultExecuting(context);
    }

    public override void OnResultExecuted(ResultExecutedContext context)
    {
        _logger.LogInformation("ClassFilter OnResultExecuted");
        base.OnResultExecuted(context);
    }
}
```

You can get various parameters and values from the `Context` object, which is passed to the overriden methods. And you can use it in the Controllers as attribute like this.

```
[ServiceFilter(typeof(LoggingActionFilter))]
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
```

```
    }
}
```

Also you need to inject it using ASP.NET Core DI to use it in ServiceFilter attribute. You can do this like this.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddScoped<LoggingActionFilter>();
}
```

Here is the log output from Console.