Для создания компонентов middleware используется делегат RequestDelegate, который выполняет некоторое действие и принимает контекст запроса:

```
public delegate  Task RequestDelegate(HttpContext context);
```

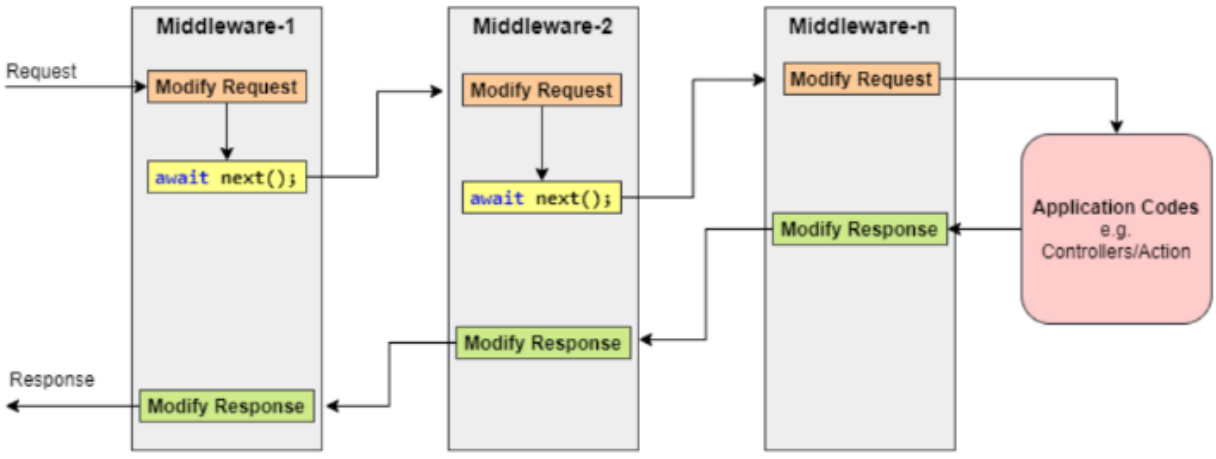
Что такое Middleware?

Middleware (промежуточное или связующее программное обеспечение) — это фрагмент кода в конвейере приложения, используемый для обработки запросов и ответов.

Например, у нас может быть middleware-компонент для аутентификации пользователя, middleware-компонент для обработки ошибок и еще один middleware-компонент для обслуживания статических файлов, таких как файлы JavaScript, CSS, разного рода изображения и т. д.

Middleware может быть встроенным как часть платформы .NET Core, добавляемым через пакеты NuGet или же написанным самим пользователем. Middleware-компоненты настраиваются в методе Configure класса запуска приложения (Startup). Метод Configure выстраивает конвейер обработки запросов в ASP.NET Core приложении. Он состоит из последовательности делегатов запросов, вызываемых один за другим.

На рисунке ниже показано, как запрос обрабатывается middleware-компонентами.



Как правило, каждое middleware обрабатывает входящие запросы и передает выполнение следующему middleware для дальнейшей обработки.

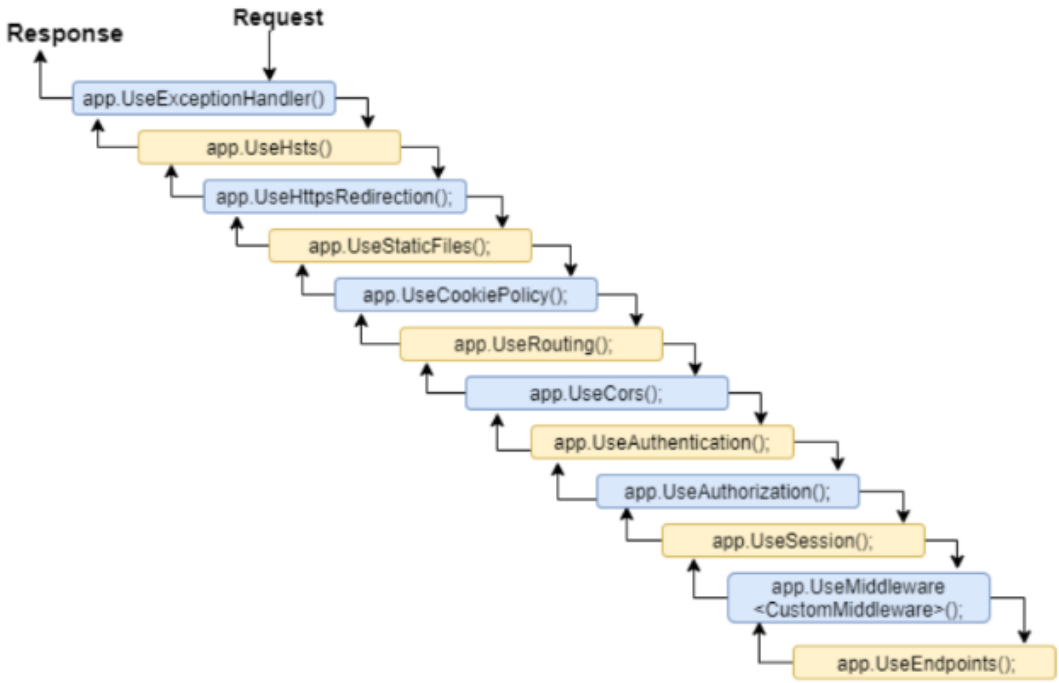
Но middleware-компонент также может решить не вызывать следующую часть middleware в конвейере. Это называется замыканием (short-circuiting) или завершением

конвейера запросов. Замыкание зачастую желательно, поскольку оно позволяет избежать ненужной работы. Например, если это запрос статического файла, такого как файл CSS, JavaScript, изображение и т. д., middleware-компонент для статических файлов может обработать и обслужить этот запрос, а затем замкнуть остальную часть конвейера.

Упорядочение Middleware

Middleware-компоненты выполняются в том порядке, в котором они добавляются в конвейер, по этому следует проявлять осторожность и добавлять middleware в правильном порядке, иначе приложение может работать не так, как вы ожидаете. Порядок расположения middleware важен для безопасности, производительности и функциональности.

Следующие middleware-компоненты предназначены для стандартных сценариев приложений и расположены в рекомендуемом порядке:



Первый middleware-компонент в конфигурации получил запрос, изменил его (при необходимости) и передал управление следующему middleware. Точно так же первый middleware-компонент выполняется последним при обработке ответа, если мы возвращаем обратно эхо. Вот почему делегаты обработки исключений должны вызываться на самых ранних этапах конвейера - чтобы они могли проверить результат и отобразить возможное исключение в удобном для браузера и клиента виде.

Методы Run, Use и Map

app.Run()

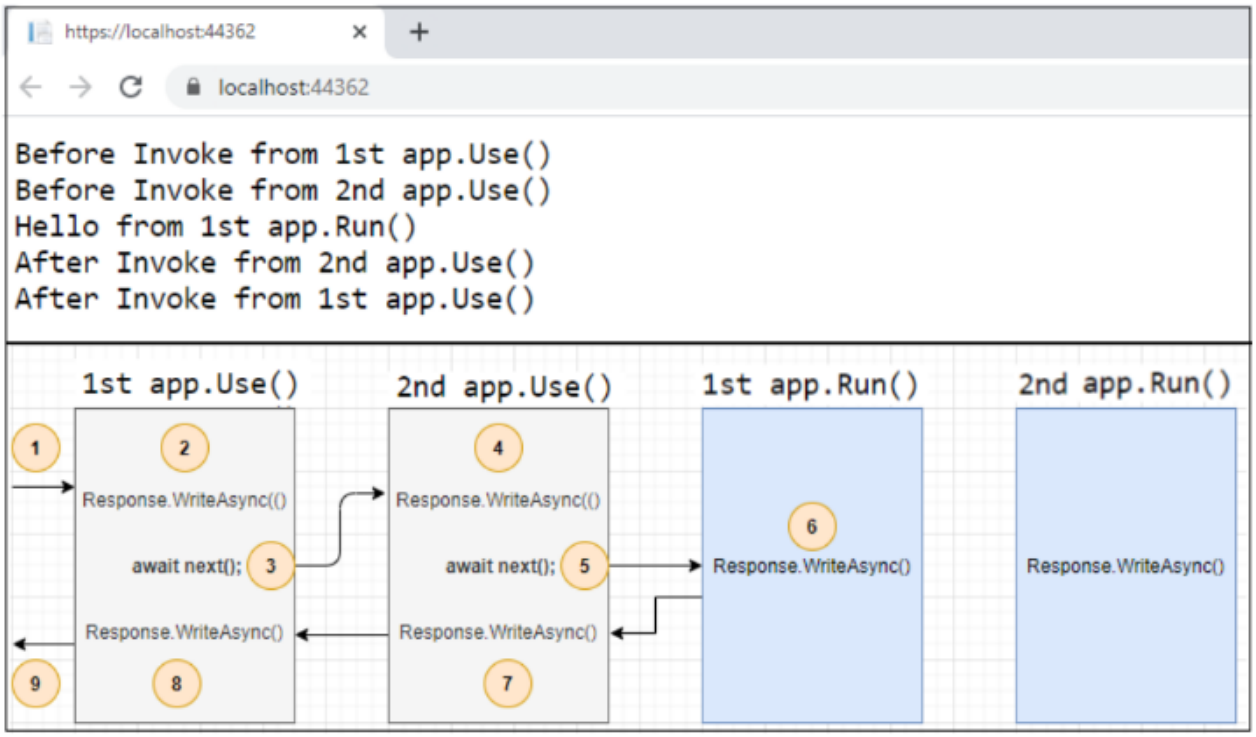
Этот метод добавляет middleware-компонент в виде Run[Middleware], который выполнится в конце конвейера. Как правило, он действует как замыкающее middleware и добавляется в конце конвейера запросов, поскольку не может вызывать следующий middleware-компонент.

```
app.Run(async (context) =>
{
        await context.Response.WriteAsync("Hello from 1st app.Run()\n");
});
```

app.Use()

Этот метод используется для конфигурирования нескольких middleware. В отличие от app.Run(), мы можем включить в него параметр next, который вызывает следующий делегат запроса в конвейере. Мы также можем замкнуть (завершить) конвейер, не вызывая параметр next.

```
app.Use(async (context, next) =>
{
        await context.Response.WriteAsync("Before Invoke from 1st
app.Use()\n");
         await next();
         await context.Response.WriteAsync("After Invoke from 1st
app.Use()\n");
 });
```



Первый делегат `app.Run()` завершает конвейер. В этом примере будет запущен только первый делегат (`«Hello from 1st app.Run()»`), а запрос никогда не достигнет второго метода `Run`.

app.Map()

Этот метод расширения используются как условное обозначение для ветвления конвейера. Map разветвляет конвейер запросов на основе пути запроса. Если путь запроса начинается с указанного пути, ветвь выполняется.

```
app.Map("/m1", HandleMapOne);
app.Map("/m2", appMap =>
{
        appMap.Run(
                async context => {
                        await context.Response.WriteAsync("Hello from 2nd
app.Map()");
                });
});
```

Создание собственного Middleware

Middleware обычно инкапсулируется в класс и предоставляется с помощью метода расширения. *Middleware* может быть создано с помощью класса с методом `InvokeAsync()` и параметром типа `RequestDelegate` в конструкторе. Тип `RequestDelegate` требуется для выполнения следующего middleware в последовательности.

Рассмотрим пример, в котором нам нужно создать собственное *middleware* для регистрации URL-адреса запроса в веб-приложении.

```
public class LogURLMiddleware
{
private readonly RequestDelegate _next;
private readonly ILogger<LogURLMiddleware> _logger;

public LogURLMiddleware(RequestDelegate next, ILoggerFactory
loggerFactory)
{
        _next = next;
        _logger = loggerFactory?.CreateLogger<LogURLMiddleware>() ?? throw
new ArgumentNullException(nameof(loggerFactory));
}
```

```
public async Task InvokeAsync(HttpContext context)
{
        _logger.LogInformation($"Request URL:
{Microsoft.AspNetCore.Http.Extensions.UriHelper.GetDisplayUrl(context.Requ
est)}");

        await this._next(context);
}
}
```

```
public static class LogURLMiddlewareExtensions
{
public static IApplicationBuilder UseLogUrl(this IApplicationBuilder app)
{
        return app.UseMiddleware<LogURLMiddleware>();
}
}
```

В методе Configure:

app.UseLogUrl();