Методы Use, Run и делегат RequestDelegate

Для конфигурации конвейера обработки запроса применяются методы **Run**, **Map** и **Use**. Рассмотрим вначале метод Run и для этого возьмем проект ASP.NET Core по типу Empty.

Изменим код класса Startup следующим образом:

```
public class Startup
{
        public void Configure(IApplicationBuilder app)
        {
                app.Run(async (context) =>
        {
                        await context.Response.WriteAsync("Hello World!");
                });
        }
}
```

Метод `Run` представляет собой простейший способ для добавления компонентов middleware в конвейер. Однако компоненты, определенные через метод Run, не вызывают никакие другие компоненты и дальше обработку запроса не передают.

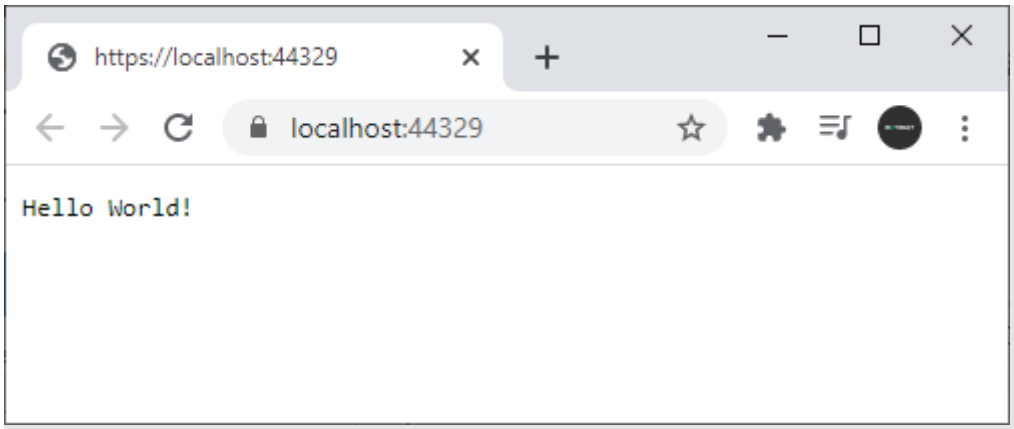
В качестве параметра метод Run принимает делегат RequestDelegate. Этот делегат имеет следующее определение:

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

Он принимает в качестве параметра контекст запроса HttpContext и возвращает объект Task. Поэтому в методе Run делегат в качестве параметра context принимает контекст запроса - объект HttpContext.

Данный метод определяет один единственный делегат запроса, который обрабатывает все запросы к приложению. Суть этого делегата заключается в отправке в ответ на запросы сообщения "Hello World!". Причем так как данный метод не передает обработку запроса далее по конвейеру, то его следует помещать в самом конце. До него же могут быть помещены другие методы.

В итоге при запуске проекта в браузере мы увидим приветствие:



Метод Use

Метод Use также добавляет компоненты middleware, которые также обрабатывают запрос, но в нем может быть вызван следующий в конвейере запроса компонент middleware. Например, изменим метод `Configure()` следующим образом:

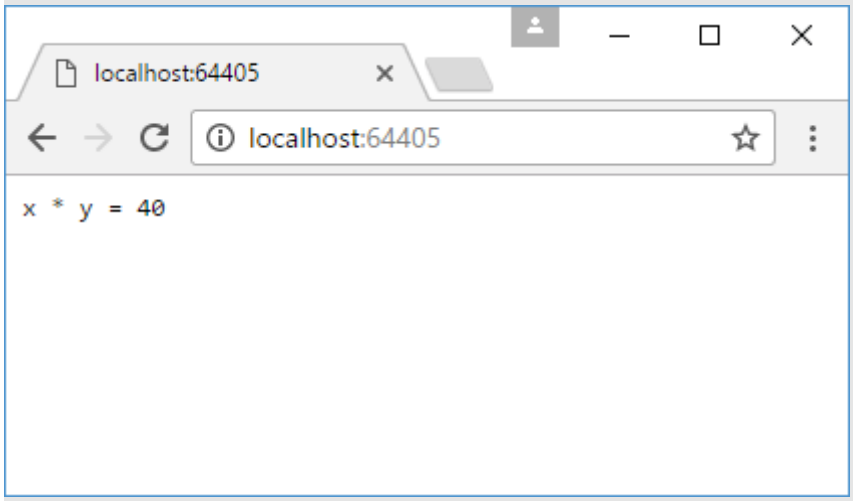
```
public void Configure(IApplicationBuilder app)
{
    int x = 5;
    int y = 8;
    int z = 0;
    app.Use(async (context, next) =>
    {
        z = x * y;
        await next.Invoke();
    });

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync($"x * y = {z}");
    });
}
```

В данном случае мы используем перегрузку метода Use, которая в качестве параметров принимает контекст запроса - объект HttpContext и делегат `Func<Task>`, который представляет собой ссылку на следующий в конвейере компонент middleware.

Метод app.Use реализует простейшую задачу - умножение двух чисел и затем передает обработку запроса следующим компонентам middleware в конвейере.

То есть при вызове `await next.Invoke()` обработка запроса перейдет к тому компоненту, который установлен в методе `app.Run()`.

В итоге в веб-браузере мы увидим следующее сообщение:



Если бы мы не использовали вызов `await next.Invoke()` или закомментировали бы его, то обращения к следующему компоненту в конвейере не произошло бы.

Однако в большинстве случаев мы можем использовать не просто методы Use, а методы расширений `app.UseXXX`, например, `UseStaticFiles()` или `UseMvc()`.

При использовании метода Use и передаче выполнения следующему делегату следует учитывать, что не рекомендуется вызывать метод `next.Invoke` после метода `Response.WriteAsync()`. Компонент middleware должен либо генерировать ответ с помощью Response.WriteAsync, либо вызывать следующий делегат посредством `next.Invoke`, но не выполнять оба этих действия одновременно. Так как согласно документации последующие изменения объекта Response могут привести к нарушению протокола, например, будет послано больше байт, чем указано в заголовке Content-Length, либо могут привести к нарушению тела ответа, например, футер страницы HTML запишется в CSS-файл.

То есть к примеру следующая обработка запроса не рекомендуется:

```
public void Configure(IApplicationBuilder app)
{
    app.Use(async (context, next) =>
    {
        await context.Response.WriteAsync("<p>Hello world!</p>");
        await next.Invoke();
    });
```

```
        app.Run(async (context) =>
    {
                // await Task.Delay(10000); можно поставить задержку
        await context.Response.WriteAsync("<p>Good bye, World...</p>");
    });
}
```

Выполнение app.Use

Если компоненты middleware в app.Use использует вызов `next.Invoke()` для передачи обработки дальше по конвейеру, то выполнение такого компонента фактически делится на две части: до `next.Invoke()` и после `next.Invoke()`. Например, определим в методе Configure следующий код:

```
public void Configure(IApplicationBuilder app)
{
        int x = 2;
    app.Use(async (context, next) =>
    {
                x = x * 2;      // 2 * 2 = 4
        await next.Invoke();    // вызов app.Run
        x = x * 2;      // 8 * 2 = 16
        await context.Response.WriteAsync($"Result: {x}");
        });

        app.Run(async (context) =>
    {
                x = x * 2;  //  4 * 2 = 8
        await Task.FromResult(0);
        });
}
```

Здесь определена переменная x, которая равна 2. Последующие вызовы компонентов middleware увеличивают ее значение в два раза. Каким образом будет происходить обработка запроса:

1. Вызов компонента app.Use
2. Увеличение переменной x в два раза: `x = x * 2;`. Теперь x равно 4.

3. Вызов `await next.Invoke()`. Управление переходит следующему компоненту в конвейере - к app.Run.

4. Увеличение переменной x в два раза: `x = x * 2;`. Теперь x равно 8.

5. Метод app.Run закончил свою работу, и управление обработкой возвращается к app.Use. Начинает выполняться та часть кода, которая идет после `await next.Invoke()`.

6. Увеличение переменной x в два раза: `x = x * 2;`. Теперь x равно 16.

7. Отправка ответа клиенту с помощью вызова `await context.Response.WriteAsync($"Result: {x}")`