ASP.NET Core MVC

(HTTP, HTML, REST API, Request Pipeline, Middleware, Environments)

Андрей Голяков

Hypertext Transfer Protocol - HTTP/1.1

Оригинальное и полное описание можно найти здесь: RFC2616.

Этот протокол описывает взаимодействие между двумя компьютерами (клиентом и сервером), построенное на базе сообщений, называемых запрос (Request) и ответ (Response). Каждое сообщение состоит из трех частей:

1. Стартовая строка
2. Заголовки
3. Тело

При этом обязательной является только стартовая строка.

Типы HTTP запросов

Стартовая строка

Формат стартовой строки запроса: `METHOD URI HTTP/VERSION`
Пример стартовой строки запроса: `GET http://ya.ru/ HTTP/1.1`

- METHOD — метод HTTP-запроса
- URI — идентификатор ресурса (Uniform Resource Identifier — единообразный идентификатор ресурса)
- VERSION — версия протокола (на данный момент актуальна версия 1.1)

Заголовки

Это набор пар имя-значение, разделенных двоеточием. В заголовках передается различная служебная информация: кодировка сообщения, название и версия браузера, адрес, с которого пришел клиент (Referrer) и так далее.

Тело сообщения

Это, собственно, передаваемые данные. В ответе передаваемыми данными, как правило, является html-страница, которую запросил браузер, а в запросе, например, в теле сообщения передается содержимое файлов, загружаемых на сервер.

HTTP

метод	GET	POST
заголовки	User-Agent: Mozilla...	Content Length: 11
тело запроса		Hello World

Запрос

Ответ

код статуса	200 OK	201 Created
заголовки	Content Length: 964	Content Type: text
тело	<html>...</html>	Hello World

Fiddler

Инструмент для просмотра деталей локальных HTTP запросов www.telerik.com/download/fiddler

```
GET http://ya.ru/ HTTP/1.1
Host: ya.ru
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)…
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*…
Accept-Encoding: gzip, deflate
Accept-Language: en,ru;q=0.9

HTTP/1.1 302 Found
Date: Tue, 30 Apr 2019 15:42:00 GMT
Cache-Control: no-cache,no-store,max-age=0,must-revalidate
Location: https://ya.ru/
Expires: Tue, 30 Apr 2019 15:42:01 GMT
Last-Modified: Tue, 30 Apr 2019 15:42:01 GMT
Set-Cookie: yandexuid=4992688631556638920; Expires=Fri,27-Apr-2029 15:42:00 GMT;
            Domain=.ya.ru; Path=/
Content-Length: 0
```

HTTP-Методы

HTTP-метод запроса указывает серверу на то, какое действие мы хотим произвести с ресурсом.

Для разграничения действий с ресурсами на уровне HTTP-методов и были придуманы следующие варианты:

- GET — запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.
- POST — используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.
- PUT — заменяет (обновляет) все текущие представления ресурса данными запроса.
- DELETE — удаляет указанный ресурс.
- PATCH — используется для частичного изменения ресурса.
- а также HEAD, CONNECT, COPY, OPTIONS, LINK, UNLINK, OPTIONS, PURGE, LOCK, UNLOCK, PROPFIND, TRAC, VIEW (некоторые из них хорошо описаны здесь: MDN HTTP Methods).

REST: REpresentational State Transfer

REST - это набор принципов построения веб-приложений.

Вообще REST охватывает более широкую область, нежели HTTP — его можно применять и в других сетях с другими протоколами. REST описывает принципы взаимодействия клиента и сервера, основанные на понятиях «ресурса» и «глагола» (можно понимать их как подлежащее и сказуемое).

В случае HTTP ресурс определяется своим URI, а глагол — это HTTP-метод.

Например, мы хотим оперировать со статьёй сайта, имеющую некий ID:

• Добавить статью с ID 47	POST	http://some.site/article/47
• Прочитать статью с ID 47	GET	http://some.site/article/47
• Изменить статью с ID 47	PATCH	http://some.site/article/47
• Заменить статью с ID 47	PUT	http://some.site/article/47
• Удалить статью с ID 47	DELETE	http://some.site/article/47

Проблемы?

Есть небольшая проблема с применением REST на практике. Проблема эта называется HTML.

Запросы PUT и DELETE можно отправлять через XMLHttpRequest, или через небраузерное обращение к серверу (скажем, через curl или даже через telnet).

Однако, нельзя сделать HTML-форму, отправляющую полноценный PUT- или DELETE-запрос.

Дело в том, спецификация HTML не позволяет создавать формы, отправляющие данные иначе, чем через GET или POST.

Проблему можно обходить добавляя скрытые поля, однако, надо понимать, что это всего-лишь имитация, которую должны поддерживать оба — как клиент, так и сервер.

Поскольку мы будем писать приложение для веба, а браузер не умеет посылать запросы методами отличными от GET и POST, нам придётся воспользоваться дополнительным инструментом.

Postman

Postman

https://www.getpostman.com

Поскольку браузера нам будет явно недостаточно для проверки всех типов запросов, для проверки нашего API мы будем использовать Postman.

Это бесплатный инструмент, позволяющий отправлять все виды HTTP-запросов и просматривать ответы сервера.

Самостоятельная работа

Создаём новое приложение

ASP.NET Core Web Application

Empty

Request Pipeline и Middleware

Когда приложению приходит HTTP-запрос, что-то должно перехватить и обработать его чтобы в итоге вернуть HTTP-ответ.

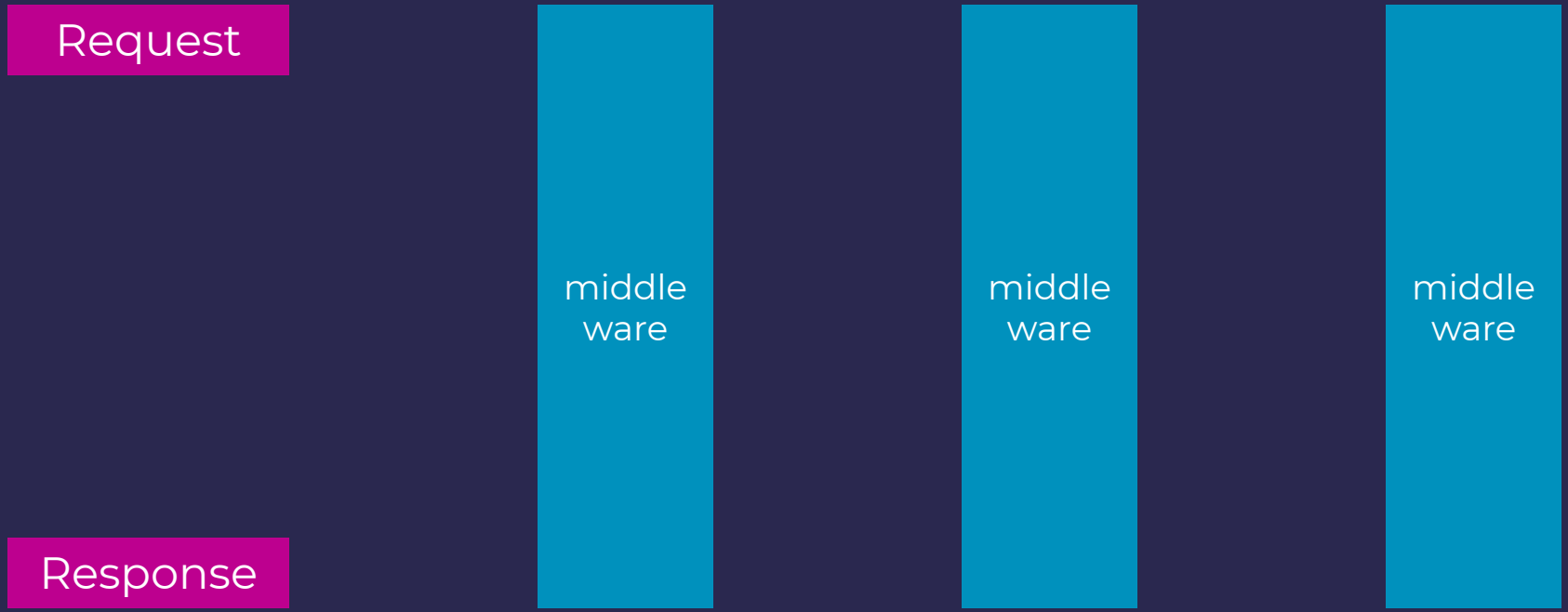
Части кода, которые обрабатывают HTTP-запросы и возвращают HTTP-ответы формируют Request Pipeline — конвейер запросов.

Мы можем добавлять в этот конвейер Middleware — промежуточные элементы (связующий код).

Примером таких middleware может быть система аутентификации или авторизации, система диагностики, система логирования.

И сам MVC также является точно таким же middleware, который также может быть добавлен в request pipeline.

Request Pipeline и Middleware

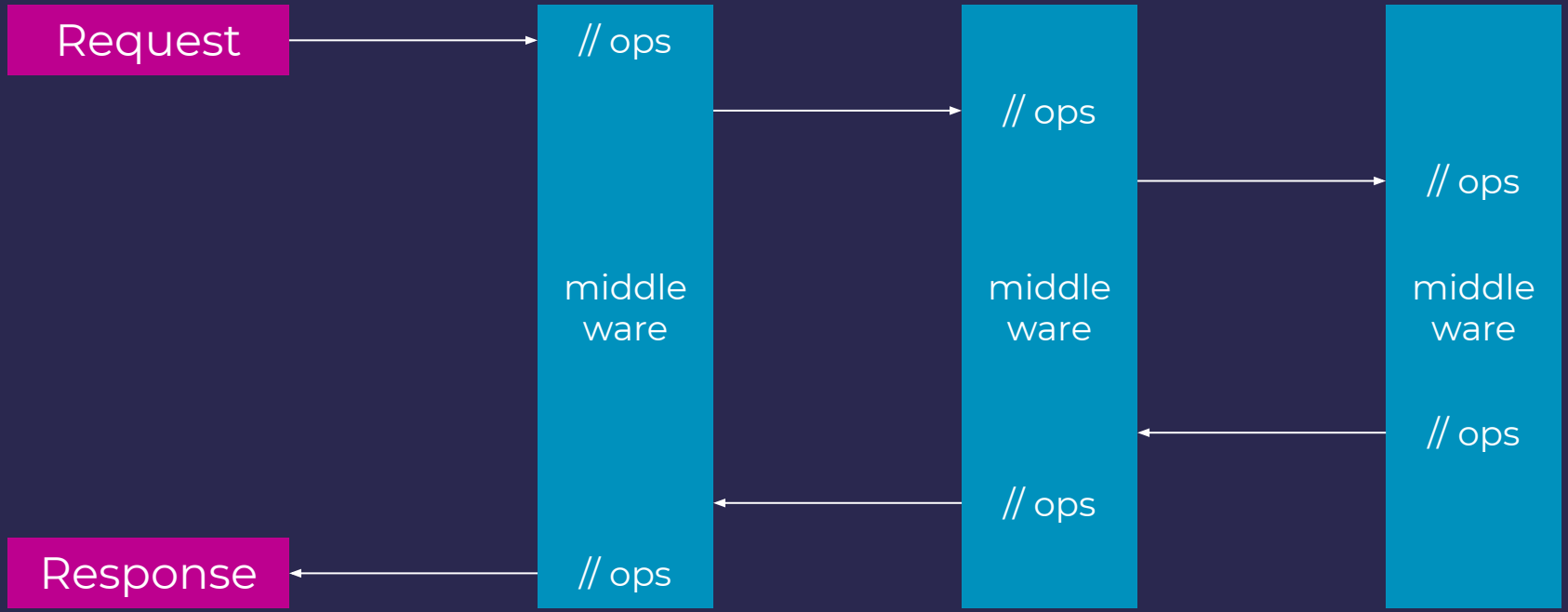


Request Pipeline и Middleware

Request Pipline ASP.NET Core:

- Состоит из последовательности делегатов, выполняющихся от одного middleware к другому.
- Каждый из них имеет возможность выполнить операции перед и после следующего делегата и дополнить или изменить Response.

Request Pipeline и Middleware



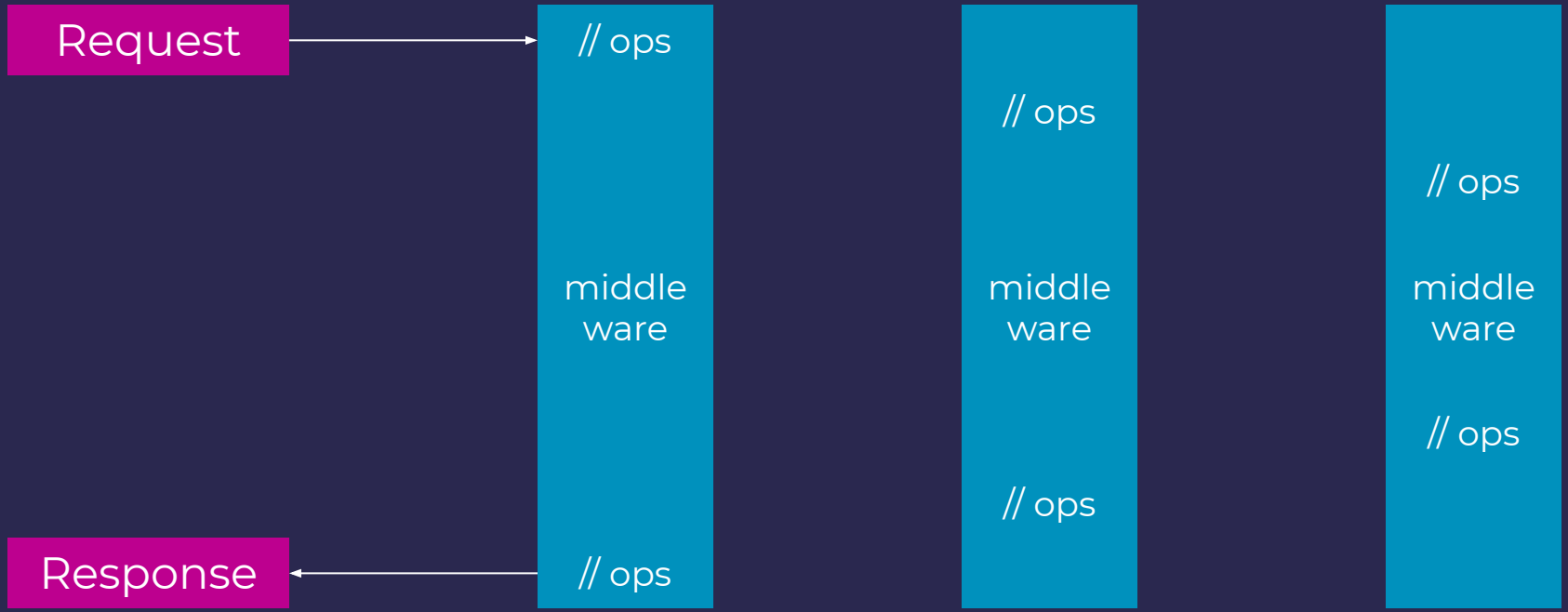
Request Pipeline и Middleware

Важно понимать, что каждый компонент middleware решает, передать ли Request дальше по цепочке middleware или нет. Таким образом, порядок, в котором мы добавляем middleware имеет важное значение.

Хорошим примером здесь будет компонент middleware, отвечающий за авторизацию. Если пользователь не авторизован для доступа к запрашиваемому ресурсу, middleware сам ответит кодом ошибки доступа 403 Forbidden и не будет передавать запрос дальше по цепочке.

В данном примере авторизационный middleware должен стоять раньше по цепочке конвейера, чтобы отфильтровывать неавторизованные запросы.

Request Pipeline и Middleware



Самостоятельная работа

Давайте посмотрим, Как мы можем добавить middleware в request pipeline.

В этом примере мы сконфигурируем ASP.NET Core Request Pipeline.

Мы добавим вывод диагностики в удобном для разработчика виде для случая, если в коде сгенерировалось исключение. И мы хотим, чтобы этот вывод происходил только в окружении разработчика (Development Environment).

Настройка Middleware в Request Pipeline

Когда я сказал: “Мы добавим…”, на самом деле, я немного преувеличил, так как разработчики Visual Studio уже добавили необходимый код в пустой шаблон ASP.NET Core Web Application. Так что, давайте посмотрим, как он выглядит (файл Startup.cs метод Configure):

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // ...

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    // ...
}
```

Давайте посмотрим, как это работает:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run((context) =>
    {
        throw new Exception("Example exception");
    });

    //app.Run(async (context) =>
    //{
    //   await context.Response.WriteAsync("Hello World!");
    //});
}
```

Environments (окружения или среды исполнения)

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    // ...
}
```

Это условие обеспечивает нам показ расширенной информации об исключении только для окружения разработчика (Development Environment).

И, думаю, это удобное место, чтобы поговорить о различных окружениях. Как видно, мы можем запускать различный код в зависимости от окружения, в котором работает наше приложение.

Environments (окружения или среды исполнения)

Давайте посмотрим на вкладку Debug свойств проекта.

В пункте Environment Variables установлена переменная окружения `ASPNETCORE_ENVIRONMENT`, которой выставлено значение `Development`.

ASP.NET Core использует эту переменную окружения для того, чтобы определить, в какой среде (или в каком окружении) он запущен. По соглашению используется 3 возможные среды выполнения:

- Development
- Staging
- Production

Переменные окружения задаются в ОС напрямую, они внешние по отношению к приложению. В ОС Windows для работы с переменными окружения используется команды `set` и `setx`:

- Посмотреть: `set ASPNETCORE_ENVIRONMENT` или `echo %ASPNETCORE_ENVIRONMENT%`
- Задать: `set ASPNETCORE_ENVIRONMENT=Staging`

*Следует учитывать контекст, в котором вы задаёте переменные окружения.

Самостоятельная работа (Environments)

Нажмите кнопку Windows ❖, наберите `cmd` и нажимте `Enter`, чтобы открыть консоль.

В консоли перейдите в папку `bin\Debug\netcoreapp2.2` относительно папки вашего проекта. Посмотрите, чему равна переменная окружения `ASPNETCORE_ENVIRONMENT`:

```
set ASPNETCORE_ENVIRONMENT
```

Скорее всего вы увидите `Environment variable ASPNETCORE_ENVIRONMENT not defined`. Установите эту переменную окружения в значение `Development`:

```
set ASPNETCORE_ENVIRONMENT=Development
```

Запустите ваше приложение:

```
dotnet имя_проекта.dll
```

Откройте в браузере URL http://localhost:5000, и убедитесь, что вы видите информацию об исключении.

Теперь остановите приложение `[Ctrl+C]`, поменяйте переменную окружения на `Production`, снова запустите приложение, в браузере (`F5`) убедитесь в отсутствии информации об исключении.

Домашняя работа

Спасибо за внимание.