NodeJS. Основы создания веб-приложений

Тема 2. Модель приложения на платформе Node.js
Объекты Request и Response. Создание простого веб-сервера. Отдача статического контента, MIME-типы. Выдача HTTP-заголовков. CORS.

Платформа Node.js использует идею **модуля** как основное средство структурирования программного кода. Модули являются строительными блоками приложений и библиотек, часто называемых пакетами (термин «пакет» нередко используется взамен термина «модуль», потому что стало обычным делом, когда пакет состоит из единственного модуля). Модули должны быть относительно небольшими. Они расширяют функциональность минималистичного **ядра**.

Приложение на платформе Node.js работает под управлением циклической структуры (цикл событий, Event Loop), которая проверяет наличие невыполненных заданий. Приложение выполняется в одном потоке. Использование единственного потока не исключает возможности одновременного выполнения нескольких заданий, связанных с вводом/выводом. Выполнение задач распределено по времени, а не разделено на несколько потоков.

Современные операционные системы предоставляют так называемый интерфейс уведомления о событиях. Его называют также «демультиплексор событий»; и он обеспечивает:

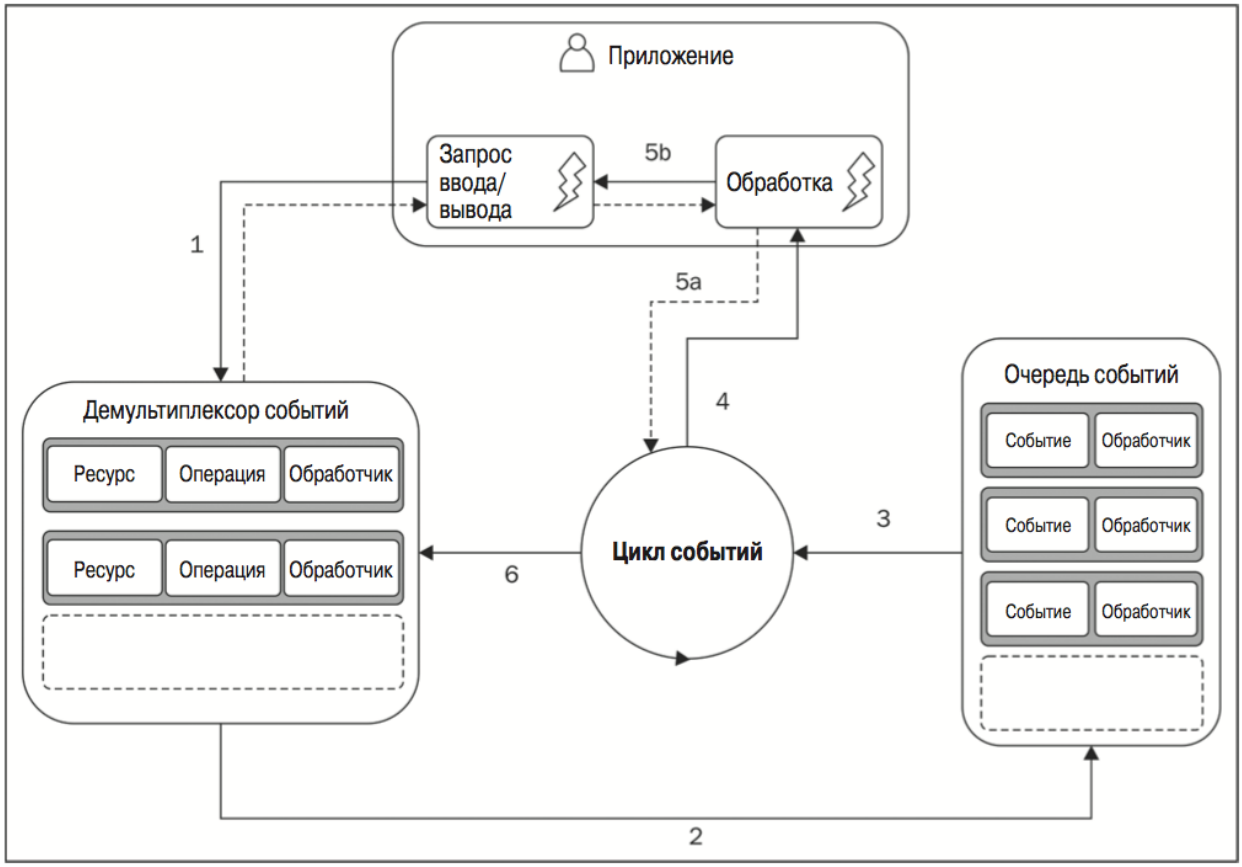
- сборку и постановку в очередь событий ввода/вывода, поступающих из набора наблюдаемых ресурсов;
- блокировку появления новых, доступных для обработки событий.

Т.е. программист на Node инициирует операцию, например, считывание файла. Этот запрос передаётся на уровень ниже, где его реализацией занимается отдельный процесс, который запишет в «очередь» «сигнал» о том, что он завершился. Получается очередь из сигналов. Эту очередь нужно обойти и выполнить сопоставленный каждому сигналу код-обработчик. Node обрабатывает все события в очереди по порядку. Добравшись до определённого события, Node вызывает сопоставленный коллбэк и передает ему всю информацию, связанную с событием. Разумеется, код коллбэка написан на JavaScript и выполняется не одновременно с каким-то другим кодом приложения, потому что JavaScript однопоточен.

(Примечание: начиная с версии 10.5.0 в Node реализована многопоточность на уровне языка, однако эта функциональность экспериментальная и требует работы с воркерами.)

Цикл событий – это расхожее название алгоритма, с помощью которого происходит слежение за очередью событий. Также это краткий способ сказать «Организация вызовов коллбэков, привязанных к событиям». Любой, писавший код для DOM, по факту пользовался этим алгоритмом, но у Цикла Событий есть свои тонкости, которые может быть весьма полезно знать для эффективной разработки и более глубокого понимания.

Приложение завершится автоматически, когда в демультиплексоре событий не останется отложенных операций и событий в очереди.



Чтобы создать простейшее приложение, которое реализует функциональность веб-сервера, т.е. приложение, отдающее ответы по протоколу HTTP, можно воспользоваться встроенным модулем http.

Этот модуль содержит и методы get и request для реализации функциональности клиента.

Можно спуститься на уровень ниже и использовать модуль net, но это оставим для самостоятельных экспериментов (класс HTTP.Server является реализацией класса Net.Server на базе TCP; протокол TCP предоставляет транспортный уровень, а HTTP — прикладной уровень коммуникаций).

Функцию Server модуля http можно использовать как конструктор с new или просто вызвать и передать ей в качестве аргумента коллбэк от двух аргументов: req(uest) и res(ponse)

```
require('http')
.Server((req, res) => res.end('Hello!'))
.listen(9999, () => console.log(process.pid));
```

Если s содержит ссылку на созданный сервер, то с помощью записи типа:

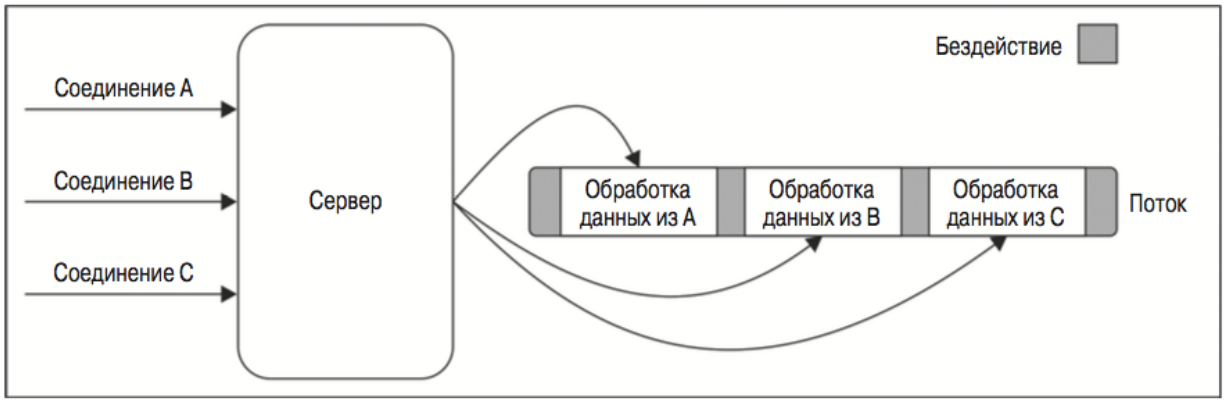
s.on('request', (req, res) => s.close() && res.end('ok'));

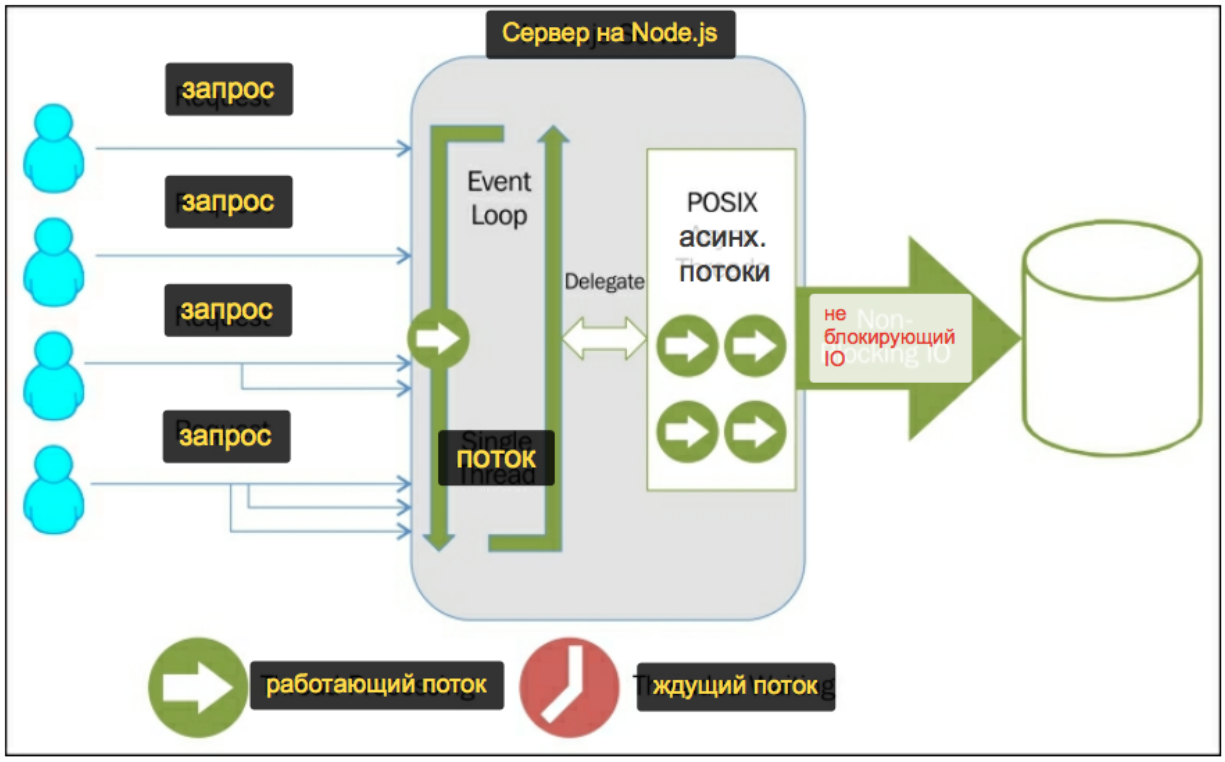
мы можем достичь аналогичного эффекта, что и передавая коллбэк.
Событие request возникает, когда приходит запрос от клиента. В режиме Keep-Alive может быть несколько запросов в рамках одного соединения. Каждый раз приходящий запрос вызывает событие request и привязанный к нему коллбэк.

Метод close позволяет отказаться от обслуживания новых соединений. Существующий набор соединений при этом не закрывается.

Коллбэк, используемый для ответов на веб-запросы, получает два параметра: запрос и ответ. Второй параметр (response) представляет собой объект типа http.ServerReponse. Это поток для записи, поддерживающий несколько функций, включая функцию response.writeHead() для создания заголовка ответа, response.write() для записи данных ответа и response.end() для завершения ответа. Первый параметр, request, представляет собой результат класса IncomingMessage, который является потоком для чтения.

Благодаря использованию демультиплексора можно сильно сократить время бездействия сервера: в очередь событий ставятся I/O действия, которые нужны для ответов на запросы по соединениям. Это касается, конечно, неблокирующих I/O операций.





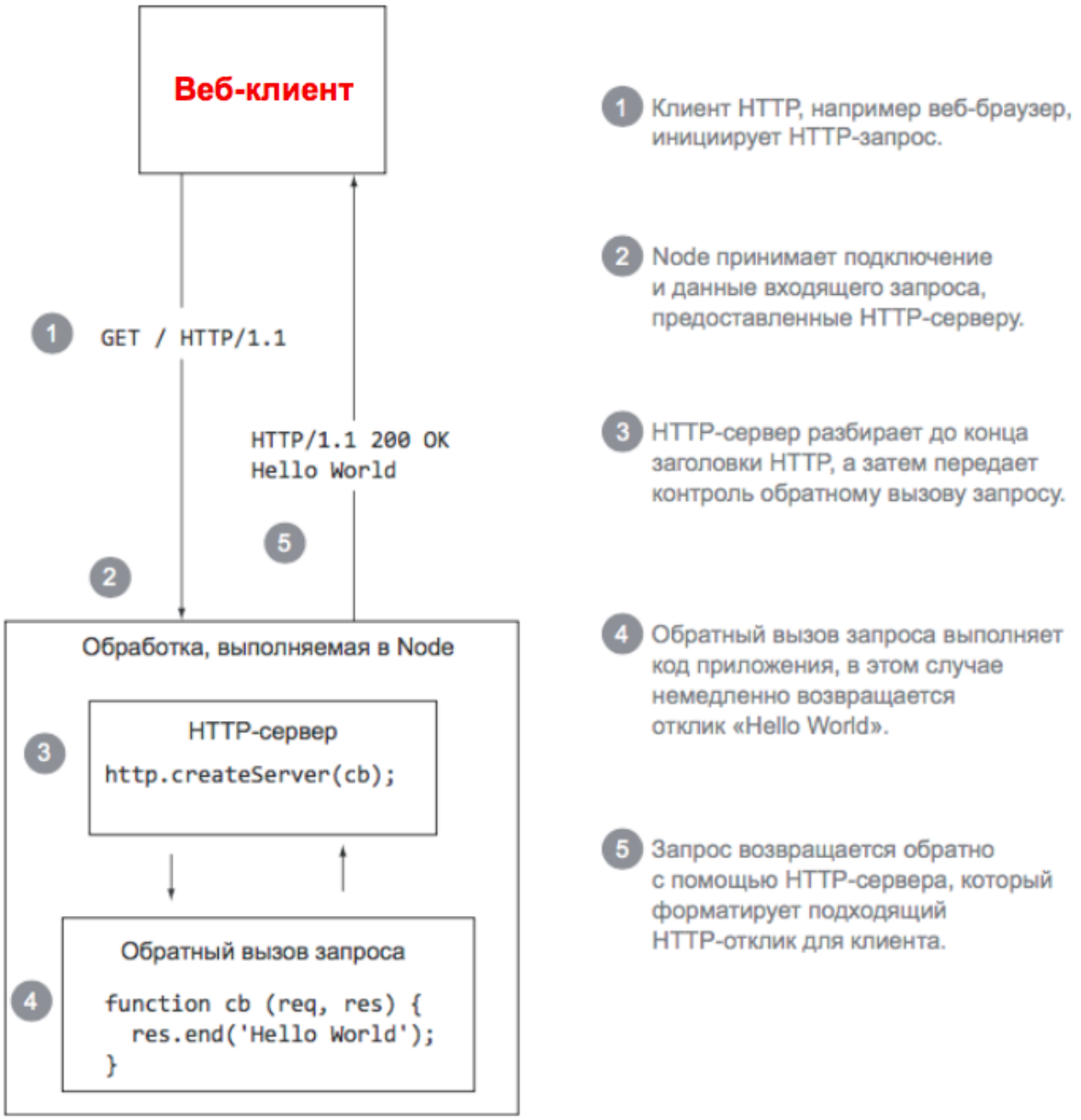
В браузере потоки под капотом создаются в рамках Web API, куда входят setTimeout и работа с DOM. В Node это не Web API, а libuv.

Кратко подведём итог. Цикл Событий – это алгоритм организации работы с возникающими задачами, при которой команды, их запускающие, могут выполняться без ожидания прихода результатов выполнения, а за результатами следит специальный менеджер, который запускает нужный для обработки этих результатов код в подходящее время между следующими командами.

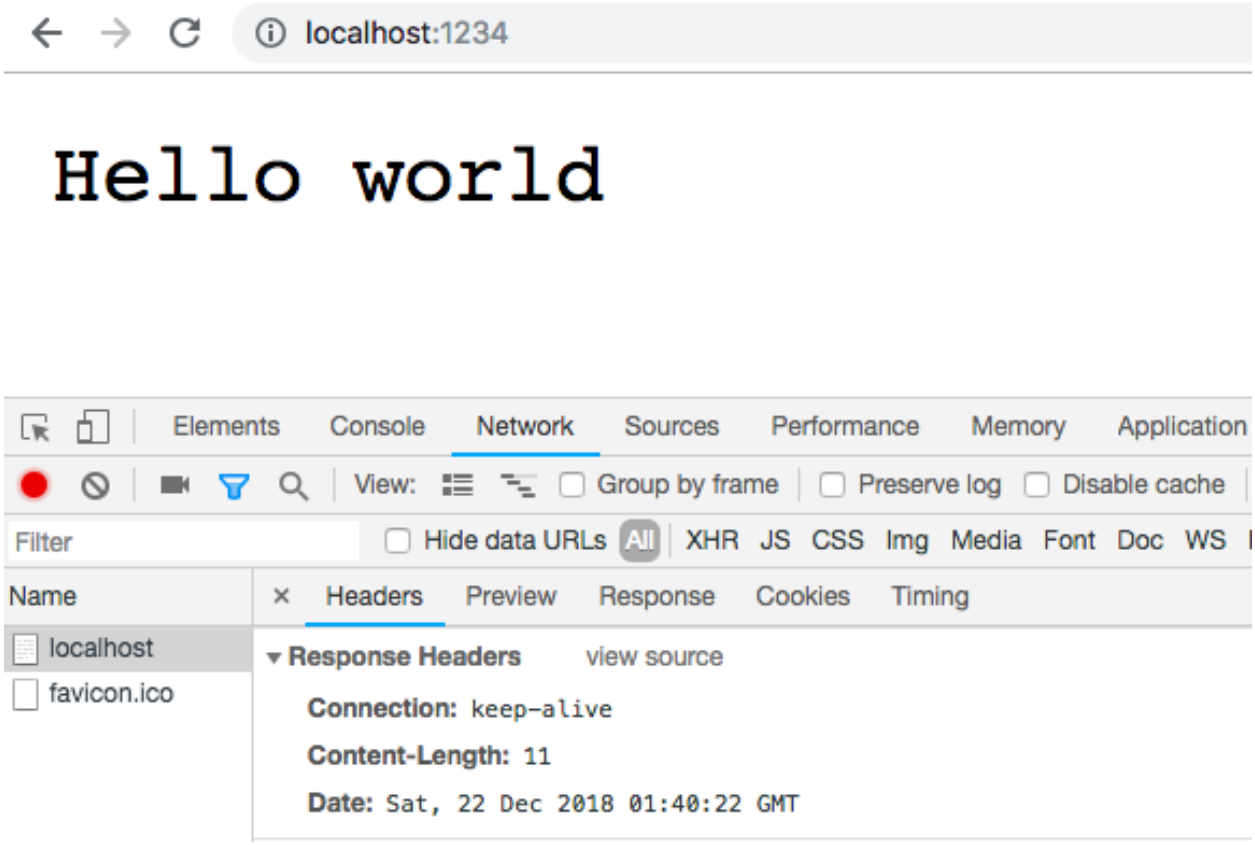
Итак, если файл с кодом ниже назвать s.js и запустить его с помощью node s,
то мы запустим жизненный цикл обработки HTTP-запросов, приходящих на localhost:1234,
и на каждый запрос сервер отвечает одно и то же: Hello world

```
const http = require('http');
function cb (req, res) {
  res.end('Hello world');
}
http.createServer(cb)
.listen(1234);
```

В req.url содержится собственно URL, например / для корня (http://localhost), или /add?a=4&b=10 для http://localhost/add?a=4&b=10 – можно использовать модуль querystring для удобного разбора того, что находится после знака вопроса.



Эти запросы можно подавать как угодно: браузером, с помощью curl, с помощью telnet, nc, приложения Postman или рукописного сценария, использующего клиентскую функциональность net или http.




```
 $ curl -i localhost:1234
HTTP/1.1 200 OK
Date: Sat, 22 Dec 2018 01:43:10 GMT
Connection: keep-alive
Content-Length: 11

Hello world-▶ eliasgoss@ ~/Yandex.Disk
```

Мы видим, что сервер вернул дату, режим соединения (keep-alive) и длину отданного контента в байтах. Это заголовки, которые высылаются благодаря работе модуля http.

```
const tp = {
    'Content-Type': 'text/plain; charset=utf-8'
};

const hu = {
    'Content-Type': 'text/html; charset=utf-8'
};

const cors = {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Methods': 'GET,POST,PUT,DELETE,OPTIONS',
    'Access-Control-Allow-Headers': 'Content-Type,Accept,Access-Control-Allow-Headers'
};
```

Чтобы отправить собственные заголовки, можно использовать метод объекта ответа writeHead. В сочетании с оператором rest/spread мы можем удобно соединять вместе несколько определённых ранее заголовков (представленны как объекты выше):

res.writeHead(200, { ...hu, ...cors });

В значении поля Content-Type мы указываем MIME-тип и кодировку (стандартом де-факто является utf-8).

> MIME (Multipurpose Internet Mail Extension, Многоцелевые расширения почты Интернета) — спецификация для передачи по сети файлов различного типа: изображений, музыки, текстов, видео, архивов и др. До слэша указывается класс типа (например text или image), а после слэша конкретный вид (plain или jpeg).

Расширение файла	Тип данных
css	application/x-pointplus
html	text/html
jpg	image/jpeg
js	text/javascript
png	image/png
txt	text/plain
zip	application/zip

Cross-origin resource sharing (CORS; с англ. — «совместное использование ресурсов между разными источниками») — технология современных браузеров, которая позволяет предоставить веб-странице доступ к ресурсам другого домена. Например, Есть три домена, желающие загрузить ресурсы с сервера Z. Для того чтобы это стало возможным, веб-серверу Z, который отдает контент, достаточно указать в заголовке ответа Access-Control-Allow-Origin белый список доверенных доменов: A, B, C. Тогда для страниц этих доменов, ограничения принципа одинакового источника на запрашиваемые страницы, не будут действовать:

Access-Control-Allow-Origin: A, B, C

После этого страницы серверов A, B, C смогут загружать контент с сервера Z.

В примере выше указана звёздочка (наименее безопасный путь) – подключение разрешаются клиентскому сценарию, находящемуся на любом хосте.