Сервер

Создание сервера

Последнее обновление: 23.11.2023

Для работы с сервером и протоколом http в Node.js используется модуль http.

Чтобы создать сервер, следует вызвать метод `http.createServer()`:

```
const http = require("http");

const server = http.createServer();
```

Метод `createServer()` возвращает объект `http.Server`. Для обработки подключений в метод createServer передается функция-обработчик:

```
const http = require("http");

const server =  http.createServer(function(request, response){
    response.end("Hello METANIT.COM!");
});
```

Эта функция принимает два параметра:

- **request**: хранит информацию о запросе
- **response**: управляет отправкой ответа

В примере выше с помощью метода `response.end()` в ответ клиенту посылается строка "Hello METANIT.COM!".

Но чтобы сервер мог прослушивать и обрабатывать входящие подключения, у объекта сервера необходимо вызвать метод **listen()**. Данный метод может принимать различный набор параметров. Но

обычно в качестве первого параметра передается номер порта, по которому запускается сервер.

```
const http = require("http");

const server =  http.createServer(function(request, response){
    response.end("Hello METANIT.COM!");
});
server.listen(3000);
```

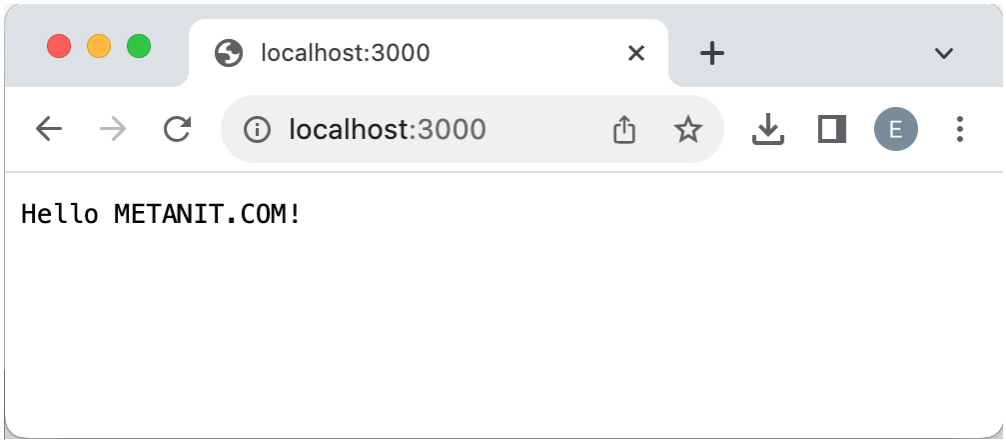
В данном случае сервер запускается по адресу 3000. Также дополнительно можно передать в метод listen функцию, которая будет срабатывать при запуске сервера:

```
const http = require("http");

const server =  http.createServer(function(request, response){
    response.end("Hello METANIT.COM!");
});
server.listen(3000, function(){ console.log("Сервер запущен по адресу http://localhost:3000")});
```

Например, запустим приложение, и после успешного запуска мы увидим на консоли соответствующее сообщение:

```
c:\app> node app.js
Сервер запущен по адресу http://localhost:3000
```

Поскольку сервер запущен на порту 3000, то мы можем обратиться к нашему приложению в браузере по адресу `http://localhost:`**`3000`**



Request

Параметр request позволяет получить информацию о запросе и представляет объект **http.IncomingMessage**. Отметим некоторые основные свойства этого объекта:

- **headers**: возвращает заголовки запроса

- **method**: тип запроса (GET, POST, DELETE, PUT)
- **url**: представляет запрошенный адрес

Например, определим следующий файл app.js:

```
const http = require("http");

http.createServer(function(request, response){

    console.log("Url:", request.url);
    console.log("Тип запроса:", request.method);
    console.log("User-Agent:", request.headers["user-agent"]);
    console.log("Все заголовки");
    console.log(request.headers);

    response.end();
}).listen(3000, function(){ console.log("Сервер запущен по адресу http://localhost:3000")});
```

Запустим его и обратимся в браузере по адресу *http://localhost:3000/*, и консоль выведет нам информацию о запросе:

```
c:\app> Сервер запущен по адресу http://localhost:3000
Url: /
Тип запроса: GET
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, li
Все заголовки
{
  host: 'localhost:3000',
  connection: 'keep-alive',
  'sec-ch-ua': '"Google Chrome";v="119", "Chromium";v="119", "Not?A_Brand";v="24"',
  'sec-ch-ua-mobile': '?0',
  'sec-ch-ua-platform': '"macOS"',
  'upgrade-insecure-requests': '1',
  'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTM
  'sec-purpose': 'prefetch;prerender',
  purpose: 'prefetch',
  accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,im
  'sec-fetch-site': 'none',
  'sec-fetch-mode': 'navigate',
  'sec-fetch-user': '?1',
  'sec-fetch-dest': 'document',
  'accept-encoding': 'gzip, deflate, br',
```

```
  'accept-language': 'ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7,fr;q=0.6,de;q=0.5,tr;q=0.4,zh-C
}
```

Response

Параметр response управляет отправкой ответа и представляет объект **http.ServerResponse**. Среди его функциональности можно выделить следующие методы:

- **statusCode**: устанавливает статусный код ответа
- **statusMessage**: устанавливает сообщение, отправляемое вместе со статусным кодом
- **setHeader(name, value)**: добавляет в ответ один заголовок
- **write**: пишет в поток ответа некоторое содержимое
- **writeHead**: добавляет в ответ статусный код и набор заголовков
- **end**: сигнализирует серверу, что заголовки и тело ответа установлены, в итоге ответ отсылается клиента. Данный метод должен вызываться в каждом запросе.

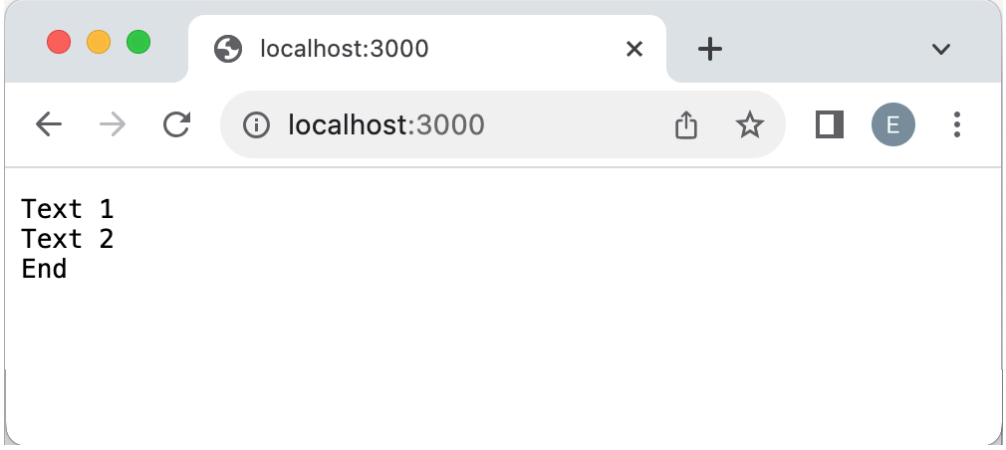
В общем случае для отправки ответа достаточно вызвать метод `end()`, в который передаются отправляемые данные:

```
response.end("Hello METANIT.COM!");
```

С помощью метода `write()` можно кусками добавить данные в ответ. Например, изменим файл app.js следующим образом:

```
const http = require("http");

http.createServer(function(request, response){

    response.write("Text 1\n");
    response.write("Text 2\n");
    response.end("End");
}).listen(3000, function(){ console.log("Сервер запущен по адресу http://localhost:3000")});
```

Запустим файл и обратимся в браузере к приложению:



Можно через end() ничего не добавлять в ответ, но в любом случае этот метод следует вызывать при отправке ответа:

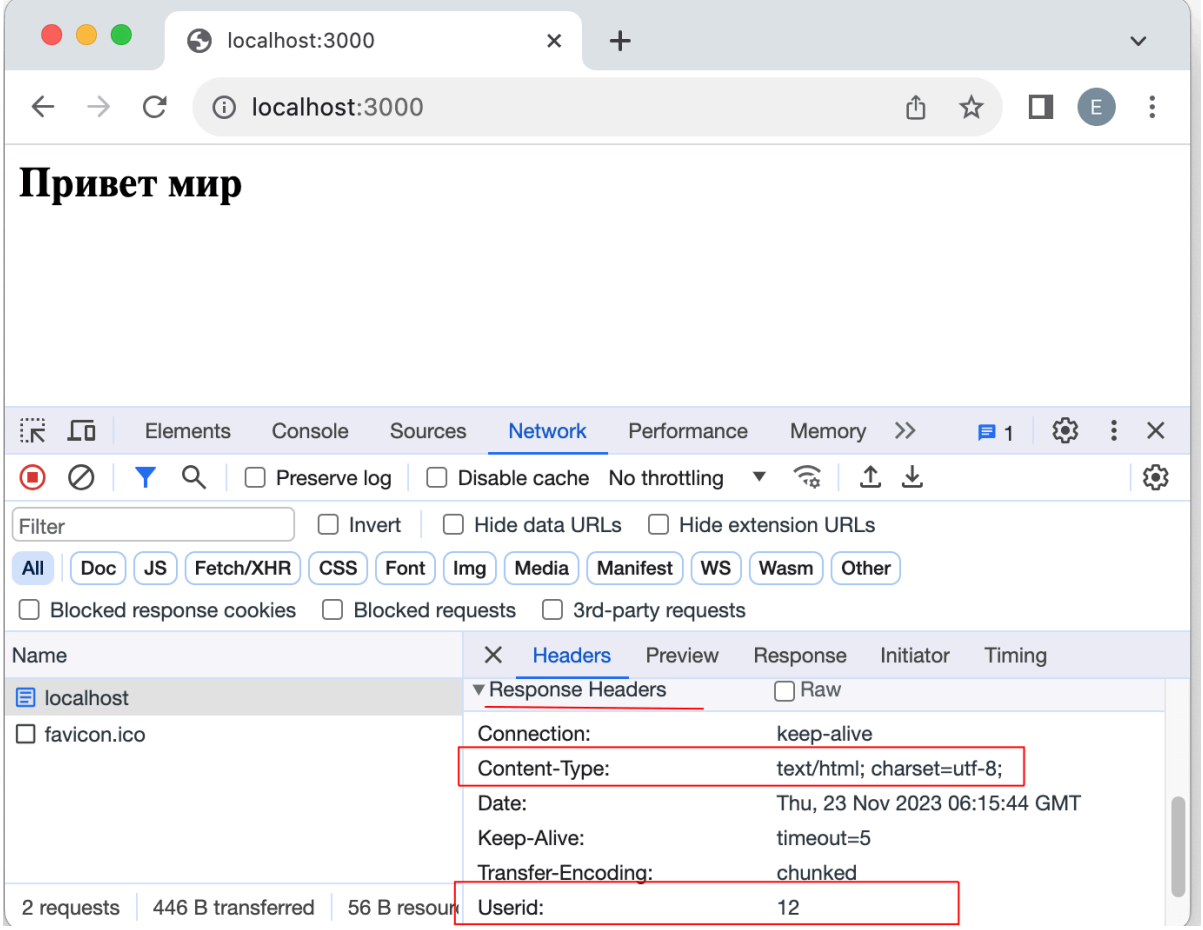
```
const http = require("http");

http.createServer(function(_, response){

    response.write("Text 1\n");
    response.write("Text 2\n");
    response.end();
}).listen(3000, function(){ console.log("Сервер запущен по адресу http://localhost:3000")});
```

Отправка заголовков

Метод setHeader() позволяет установить заголовки ответа:

```
const http = require("http");

http.createServer(function(_, response){

    response.setHeader("UserId", 12);   // установка кастомного заголовка
    response.setHeader("Content-Type", "text/html; charset=utf-8;");
    response.write("<h2>Привет мир</h2>");
    response.end();
}).listen(3000, function(){ console.log("Сервер запущен по адресу http://localhost:3000")});
```

В данном случае для теста устанавливаем кастомный заголовок "UserId", пусть он равен 12. А чтобы отправляемый ответ интерпретировался браузером как код html, для заголовка "Content-Type" устанавливаем значение "text/html; charset=utf-8;". Результат работы:



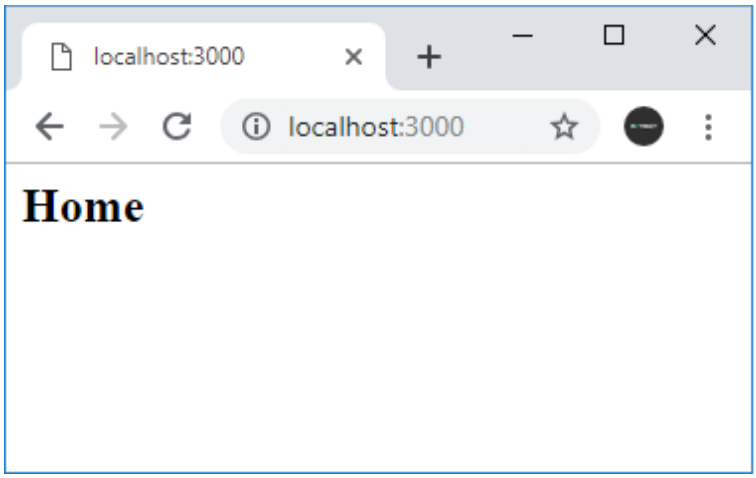
Маршрутизация

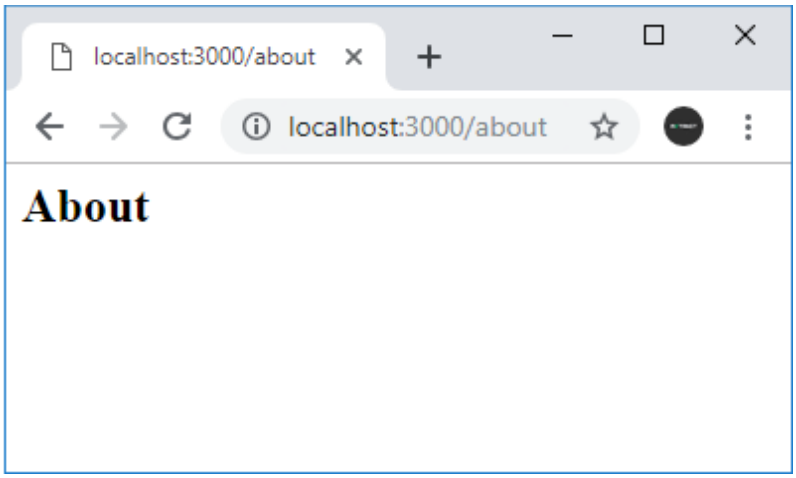
По умолчанию Node.js не имеет встроенной системы маршрутизации. Обычно она реализуется с помощью специальных фреймворков типа Express, о котором речь пойдет в следующей главе. Однако если необходимо разграничить простейшую обработку пары-тройки маршрутов, то вполне можно использовать для этого свойство **url** объекта Request. Например:

```
const http = require("http");

http.createServer(function(request, response){

    response.setHeader("Content-Type", "text/html; charset=utf-8;");

    if(request.url === "/home" || request.url === "/"){
        response.write("<h2>Home</h2>");
    }
    else if(request.url == "/about"){
        response.write("<h2>About</h2>");
    }
    else if(request.url == "/contact"){
        response.write("<h2>Contacts</h2>");
    }
    else{
        response.write("<h2>Not found</h2>");
    }
```

```
    response.end();
}).listen(3000);
```

В данном случае обрабатываются три маршрута. Если идет обращение к корню сайта или по адресу *localhost:3000/home*, то пользователю выводится строка "Home". Ели обращение идет по адресу *localhost:3000/about*, то пользователю в браузере отображается строка About и так далее. Если запрошенный адрес не соответствует ни одному маршруту, то выводится заголовок "Not Found".





Однако опять же отмечу, что рамках специальных фреймворков, которые работают поверх Node.js, например, Express, есть более удобные способы для обработки маршрутов, которые нередко и используются.

Переадресация

Переадресация предполагает отправку статусного кода 301 (постоянная переадресация) или 302 (временная переадресация) и заголовка **Location**, который указывает на новый адрес. Например, выполним переадресацию с адреса *localhost:3000/* на адрес *localhost:3000/newpage*

```
const http = require("http");

http.createServer(function(request, response){

    response.setHeader("Content-Type", "text/html; charset=utf-8;");

    if(request.url === "/"){
        response.statusCode = 302; // временная переадресация
        // на адрес localhost:3000/newpage
        response.setHeader("Location", "/newpage");
    }
    else if(request.url == "/newpage"){
        response.write("New address");
    }
    else{
        response.statusCode = 404; // адрес не найден
        response.write("Not Found");
    }
```

```
    response.end();
}).listen(3000);
```

Назад **Содержание** **Вперед**

Перевод на карту
Номер карты:
4048415020898850