

Node.js

Часть 2

Работа с файлами

Работа с файлами

```
const fs = require('fs');
```

```
fs.readFile(__filename, (error, content) => {  
  console.log(content);  
});
```

`__filename` – строка, которая хранит абсолютный путь до текущего файла

```
Buffer 63 6f 6e 73 74 20 66 73 20 3d 20 72 65...
```

Buffer

Класс для работы с бинарными данными

Буфер можно рассматривать как массив чисел, ограниченных диапазоном 0-255

Каждое число представляет байт

Buffer

```
const letterB = new Buffer([98]);
```

```
letterB.toString(); // b
```

```
letterB.toString('utf-8'); // b
```

Buffer

```
const msg = new Buffer([0x2f, 0x04, 0x3d, 0x04,  
    0x34, 0x04, 0x35, 0x04, 0x3a, 0x04, 0x41, 0x04]);
```

```
msg.toString(); // По умолчанию: utf-8
```

```
// \u0004=\u00044\u00045\u0004:\u0004A\u0004
```

```
msg.toString('usc-2');
```

```
// Яндекс
```

Работа с файлами

```
fs.readFile(__filename, (error, buffer) => {  
    console.log(buffer.toString('utf-8'));  
});
```

```
fs.readFile(__filename, 'utf-8', (error, content) => {  
    console.log(content);  
});
```

Работа с файлами

```
fs.appendFile();
```

```
fs.writeFile();
```

```
fs.unlink();
```

```
fs.mkdir();
```


Работа с файлами

```
fs.readFileSync(__filename);
```

```
fs.writeFileSync(__filename, data);
```

```
fs.mkdirSync('/games/diablo3');
```

Блокируют поток выполнения программы

```
const path = require('path');
```

```
// Windows
```

```
path.join('\a', 'b', '..', 'c'); // \a\c
```

```
// UNIX
```

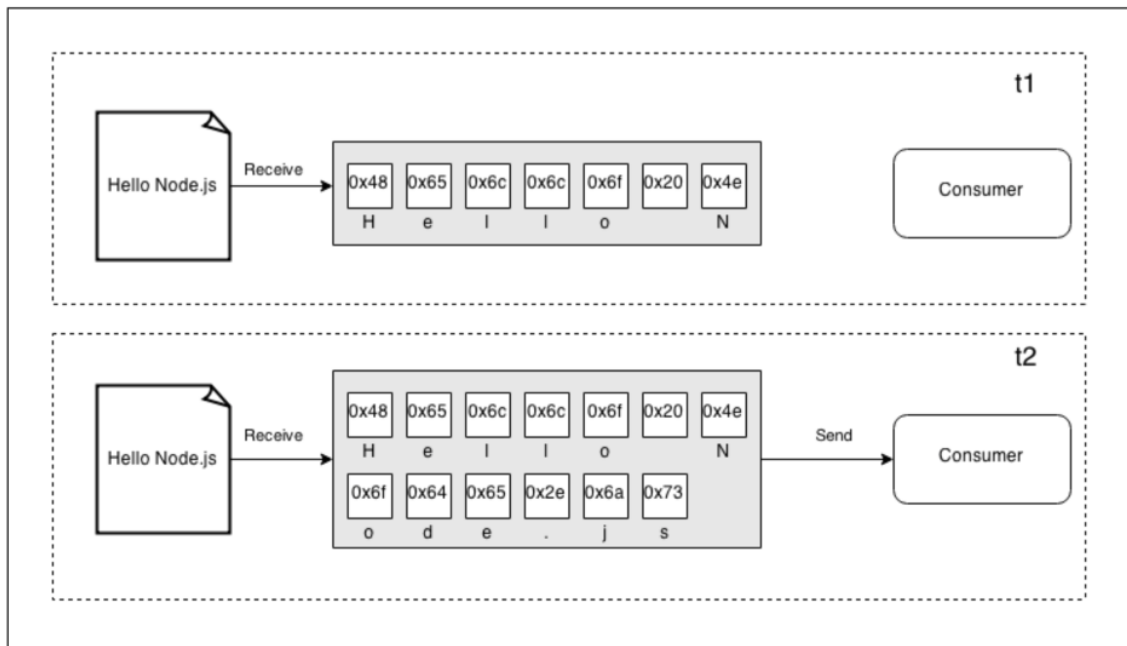
```
path.join('/a', 'b', '..', 'c'); // /a/c
```

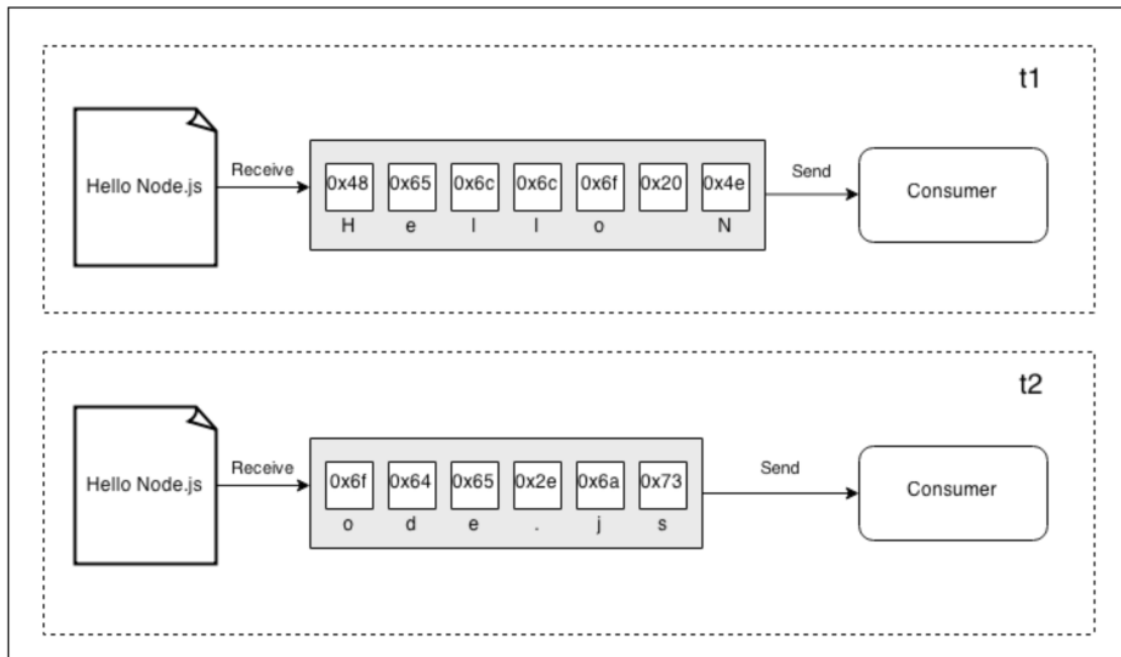
Работа с файлами

```
fs.readFile(__filename, (error, buffer) => {  
    console.log(buffer);  
});
```

Данные предварительно сохраняются в Buffer

Только когда **весь** файл прочитан, данные
передаются в обработчик





Потоки

Потоки

Данные готовы для обработки, как только
будет прочитан **первый** chunk

- ✓ Экономия ресурсов
- ✓ Экономия времени

```
const fs = require('fs');
const zlib = require('zlib');

fs.readFile(__filename, (error, buffer) => {
  zlib.gzip(buffer, (error, buffer) => {
    fs.writeFile(__filename + '.gz', buffer, error => {
      console.log('Success');
    });
  });
});
```


Экономия ресурсов

Buffer в V8 не может быть больше
0x3FFFFFFFF bytes ~ 1 Gb

File size is greater than possible Buffer:
0x3FFFFFFFF bytes

```
const fs = require('fs');  
const zlib = require('zlib');
```

```
fs  
  .createReadStream(__filename)  
  .pipe(zlib.createGzip())  
  .pipe(fs.createWriteStream(__filename + '.gz'))  
  .on('finish', () => console.log('Success'))  
  .on('error', () => console.error('Error!'));
```

```
stream instanceof EventEmitter === true
```

Виды потоков

Readable - для чтения

Writable - для записи

Duplex - для чтения и записи

Transform - Duplex, но с преобразованием

Примеры Readable потоков

```
fs.createReadStream(filename); // fs.ReadStream

require('http')
  .request(options)
  .on('response', res => { // http.IncomingMessage
    res.on('data', chunk => {});
    res.on('end', () => {});
  });
```

События Readable потоков

data - при получении чанка данных

end - при завершении данных в потоке

close - при закрытии потока

error - в случае ошибки

Примеры Writable потоков

```
fs.createWriteStream(filename); // fs.WriteStream
```

```
server.on('request', (req, res) => { // http.ServerResponse  
  res.write('Hello, ');  
  res.write('World!');  
  res.end();  
});
```

Методы Writable потоков

`.write()` - отправляет порцию данных в поток

`.end()` - завершает запись в поток

События Writable потоков

`error` - в случае ошибки передачи данных

Передача данных из одного потока в другой

```
$ cat index.js | grep "function" | wc -l
```

```
readable.on('data', chunk => {  
    writable.write(chunk);  
});
```

```
readable.pipe(writable);
```

```
readable.pipe(transform).pipe(writable);
```

```
const fs = require('fs');
const zlib = require('zlib');

fs.readFile(__filename, (error, buffer) => {
  zlib.gzip(buffer, (error, buffer) => {
    fs.writeFile(__filename + '.gz', buffer, error => {
      console.log('Success');
    });
  });
});
```

```
const fs = require('fs');  
const zlib = require('zlib');
```

```
fs  
  .createReadStream(__filename)  
  .pipe(zlib.createGzip())  
  .pipe(fs.createWriteStream(__filename + '.gz'))  
  .on('finish', () => console.log('Success'))  
  .on('error', () => console.error('Error!'));
```

Почитать про работу с файлами

File System

nodejs.org

Stream

nodejs.org

Node.js Streams: Everything you need to know

Samer Buna

Node.js Design Patterns

Mario Casciaro, Luciano Mammino

Отладка

```
console.log();
```

Консоль

```
console.log('Some info'); // Эквивалентно console.info
```

```
console.error('Some error'); // Эквивалентно console.warn
```

```
$ node index.js 1>stdout.log 2>stderr.log
```

```
$ cat stdout.log  
Some info
```

```
$ cat stderr.log  
Some error
```

Консоль

```
console.trace();
```

```
$ node index.js 2>stderr.log
```

```
$ cat stderr.log
```

```
Trace
```

```
  at Object.<anonymous> (/Users/username/examples/console.js:
  at Module._compile (module.js:624:30)
  at Object.Module._extensions..js (module.js:635:10)
  at Module.load (module.js:545:32)
  at tryModuleLoad (module.js:508:12)
  at Function.Module._load (module.js:500:3)
  at Function.Module.runMain (module.js:665:10)
```


Консоль

```
console.time('long-operation');  
executeSomeLongOperation();  
  
console.timeEnd('long-operation');  
// long-operation: 8589.798ms
```

```
require('debug');
```

The screenshot shows the npm package page for 'debug'. The browser address bar shows 'https://www.npmjs.com/package/debug'. The page features the npm logo, the package name 'debug' with a 'public' tag, and statistics: 'build passing', 'coverage 74%', 'slack 10', and 'backers 3'. A terminal window is displayed with a log of debug output. On the right, there are links to 'npm install debug', 'how? learn more', the publisher 'tootallnate', the latest version '3.1.0', the GitHub repository, and a list of collaborators. At the bottom, download statistics and open issues are listed.

debug public

build passing coverage 74% slack 10 backers 3

```
$ DEBUG=* node examples/node/app.js
http booting 'My App' +0ms
worker:a doing lots of uninteresting work +0ms
worker:b doing some work +0ms
http listening +23ms
worker:a doing lots of uninteresting work +424ms
worker:a doing lots of uninteresting work +307ms
worker:b doing some work +814ms
worker:b doing some work +58ms
worker:a doing lots of uninteresting work +312ms
worker:a doing lots of uninteresting work +647ms
worker:a doing lots of uninteresting work +469ms
worker:b doing some work +1s
worker:a doing lots of uninteresting work +797ms
worker:a doing lots of uninteresting work +153ms
worker:b doing some work +1s
worker:a doing lots of uninteresting work +491ms
worker:a doing lots of uninteresting work +323ms
worker:b doing some work +602ms
```

npm install debug
how? learn more

tootallnate published 2 months ago

3.1.0 is the latest of 53 releases

github.com/visionmedia/debug

MIT

Collaborators list

Stats

946 985 downloads in the last day

15 873 186 downloads in the last week

65 829 709 downloads in the last mo...

28 open issues on GitHub

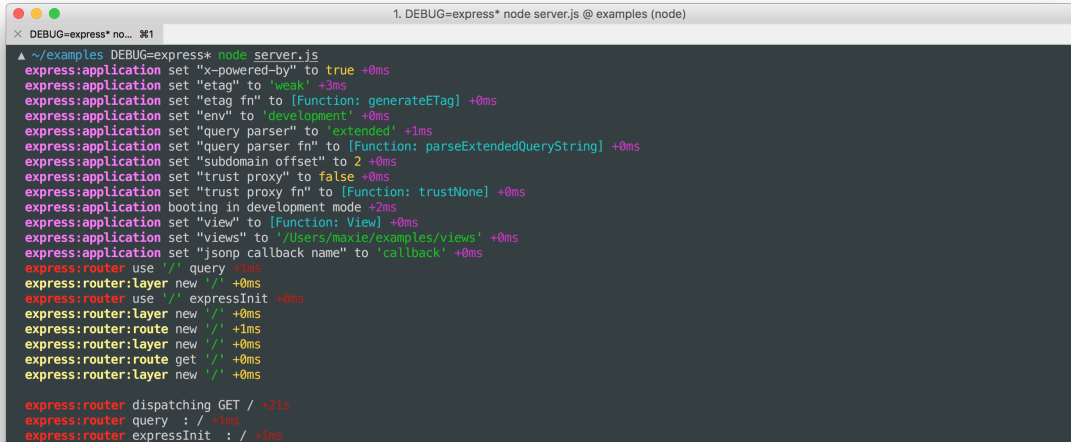
6 open pull requests on GitHub

A tiny JavaScript debugging utility modelled after Node.js core's debugging technique. Works in Node.js and web browsers.

Installation

```
require('debug');
```

```
$ DEBUG=express* node server.js
```

A terminal window titled "1. DEBUG=express* node server.js @ examples (node)" displays the output of running a Node.js application with the DEBUG environment variable set to "express*". The logs show the initialization of the Express application, including setting various options like "x-powered-by", "etag", "env", "query parser", and "trust proxy". It also shows the booting process in development mode, setting the "view" engine and "views" directory, and configuring the JSONP callback name. Finally, it shows the router layer being initialized and the first GET request being dispatched to the root path. The logs are color-coded: green for application-level settings, red for router-related events, and blue for the final request dispatch.

```
1. DEBUG=express* node server.js @ examples (node)
x DEBUG=express* no...  81
^ ~/examples DEBUG=express* node server.js
express:application set "x-powered-by" to true +0ms
express:application set "etag" to 'weak' +3ms
express:application set "etag fn" to [Function: generateETag] +0ms
express:application set "env" to 'development' +0ms
express:application set "query parser" to 'extended' +1ms
express:application set "query parser fn" to [Function: parseExtendedQueryString] +0ms
express:application set "subdomain offset" to 2 +0ms
express:application set "trust proxy" to false +0ms
express:application set "trust proxy fn" to [Function: trustNone] +0ms
express:application booting in development mode +2ms
express:application set "view" to [Function: View] +0ms
express:application set "views" to '/Users/maxie/examples/views' +0ms
express:application set "jsonp callback name" to 'callback' +0ms
express:router use '/' query +1ms
express:router:layer new '/' +0ms
express:router use '/' expressInit +0ms
express:router:layer new '/' +0ms
express:router:route new '/' +1ms
express:router:layer new '/' +0ms
express:router:route get '/' +0ms
express:router:layer new '/' +0ms
express:router dispatching GET / +21s
express:router query : / +1ms
express:router expressInit : / +1ms
```

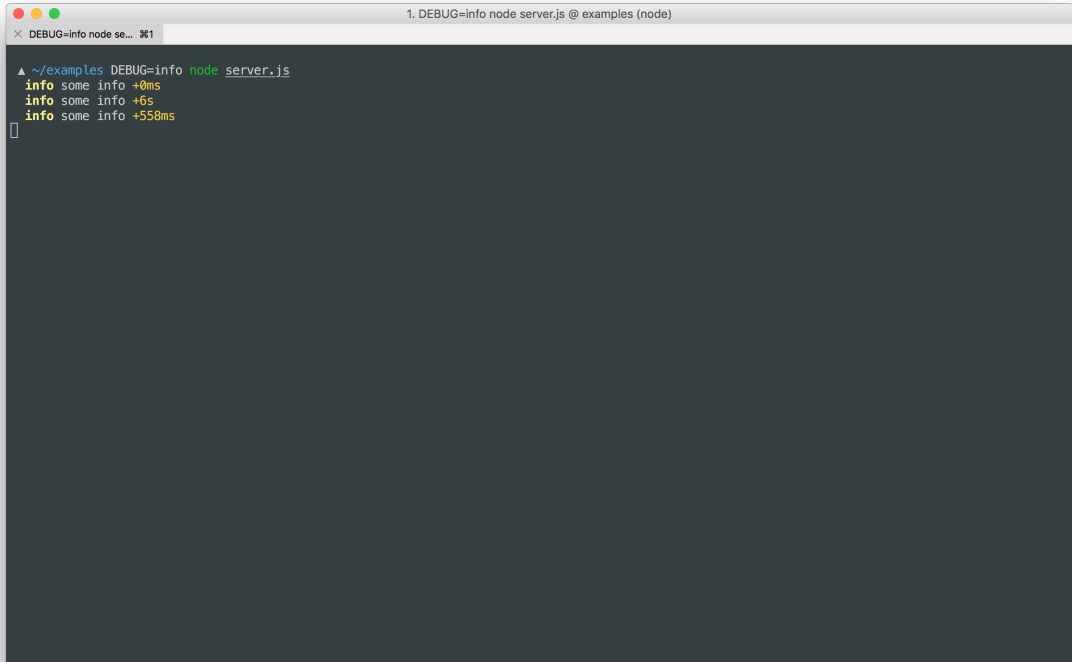
```
require('debug');
```

```
$ npm i debug --save
```

```
const debug = require('debug');  
const app = require('express')();  
  
const infoDebug = debug('info');  
  
app.get('/', (req, res) => {  
  infoDebug('some info');  
  res.send('Hello, world!');  
});  
  
app.listen(8000);
```

```
require('debug');
```

```
$ DEBUG=info node server.js
```

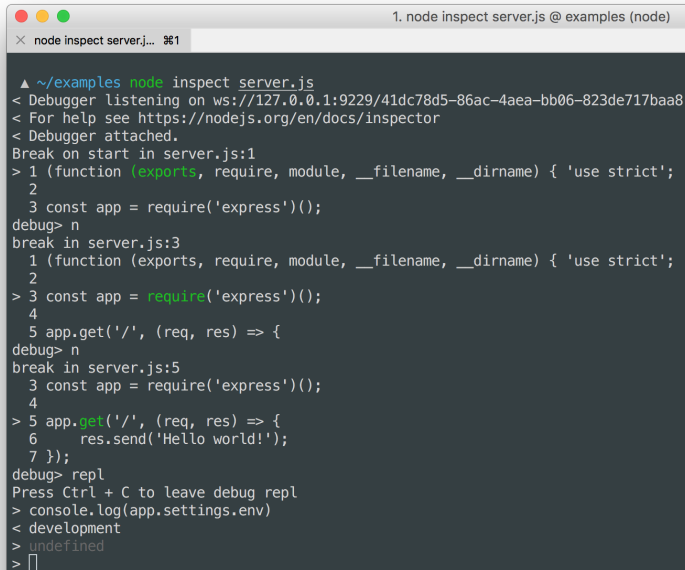
A terminal window with a dark background and light text. The title bar shows '1. DEBUG=info node server.js @ examples (node)'. The terminal content shows a command prompt with a green cursor, followed by the command 'node server.js'. Below this, three lines of output are shown, each starting with 'info' in yellow and followed by 'some info' and a timestamp. The timestamps are '+0ms', '+6s', and '+558ms'. A cursor is visible on the line following the last output.

```
1. DEBUG=info node server.js @ examples (node)  
x DEBUG=info node se... 261  
^ ~/examples DEBUG=info node server.js  
info some info +0ms  
info some info +6s  
info some info +558ms  
^
```

Debugger

Debugger

```
$ node inspect server.js
```



```
1. node inspect server.js @ examples (node)
x node inspect server.js... 1
▲ ~/examples node inspect server.js
< Debugger listening on ws://127.0.0.1:9229/41dc78d5-86ac-4aea-bb06-823de717baa8
< For help see https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in server.js:1
> 1 (function (exports, require, module, __filename, __dirname) { 'use strict';
  2
  3 const app = require('express')();
debug> n
break in server.js:3
  1 (function (exports, require, module, __filename, __dirname) { 'use strict';
  2
  3 const app = require('express')();
> 3 const app = require('express')();
  4
  5 app.get('/', (req, res) => {
debug> n
break in server.js:5
  3 const app = require('express')();
  4
  5 app.get('/', (req, res) => {
  6   res.send('Hello world!');
  7 });
debug> repl
Press Ctrl + C to leave debug repl
> console.log(app.settings.env)
< development
> undefined
> |
```

Debugger

Команды:

`help` – Посмотреть полный список доступных команд

`repl` – Открыть REPL в текущем контексте исполнения

`cont, c` – Продолжить до точки останова

`next, n` – Продолжить до следующей строки

`step, s` – Продолжить с заходом в функцию

`out, o` – Продолжить выйдя из текущей функции

Debugger

Чтобы указать в коде точку останова достаточно добавить в него ключевое слово `debugger`

```
const app = require('express')();
```

```
app.get('/', (req, res) => {  
  debugger;  
  res.send('Hello, world!');  
});
```

```
app.listen(8000);
```

Debugger

```
1. node inspect server.js @ examples (node)

▲ ~/examples node inspect server.js
< Debugger listening on ws://127.0.0.1:9229/a1c90046-ec91-471e-97aa-65693b20a894
< For help see https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in server.js:1
> 1 (function (exports, require, module, __filename, __dirname) { 'use strict';
  2
  3 const app = require('express')();
debug> c
< Listening on http://localhost:8000
break in server.js:6
  4
  5 app.get('/', (req, res) => {
> 6     debugger;
  7     res.send('Hello, world!');
  8 });
debug> repl
Press Ctrl + C to leave debug repl
> res.send('Message from debugger repl!');
{ domain: null,
  _events: Object,
  _eventsCount: 1,
  _maxListeners: 'undefined',
  output: Array(0),
  ... }
> []
```

Отладка в Chrome DevTools

Отладка в Chrome DevTools

```
$ node --inspect server.js
```

Для того чтобы сразу остановить
исполнение кода на первой строке:

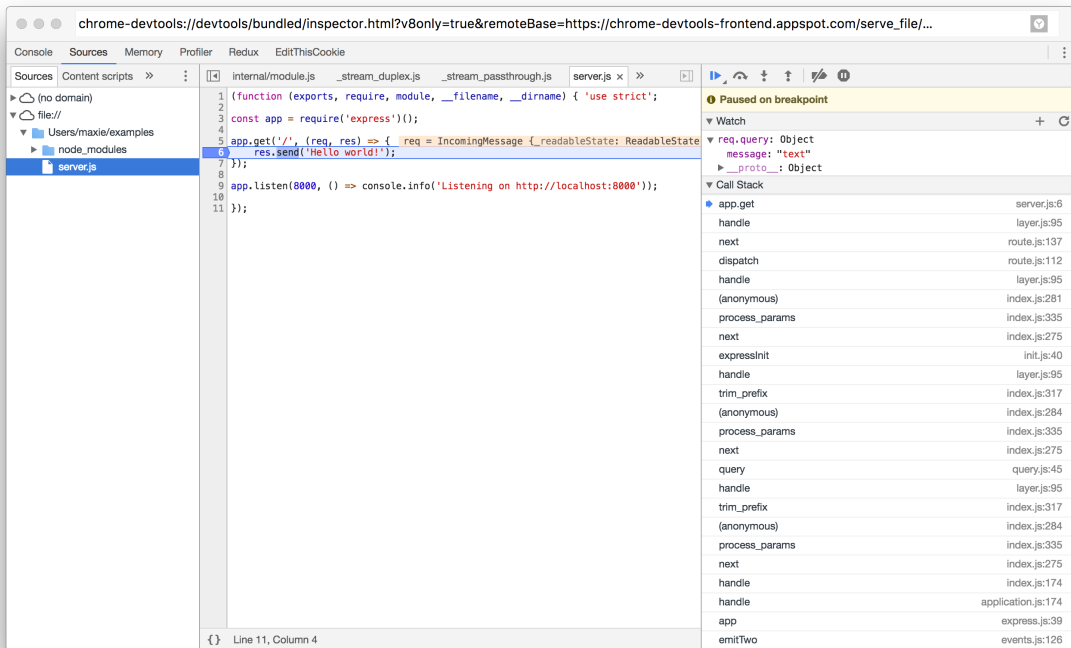
```
$ node --inspect-brk server.js  
Debugger listening on ws://127.0.0.1:9229/21bd68bf-b326-49b2-b4  
For help see https://nodejs.org/en/docs/inspector  
Listening on http://localhost:8000
```

Отладка в Chrome DevTools

Для того чтобы открыть отладчик нужно:

1. Перейти на страницу `about:inspect`
2. В разделе `Remote targets` выбрать или добавить новую цель для отладки и нажать `inspect`

Отладка в Chrome DevTools



Отладка в VS Code

Отладка в VS Code

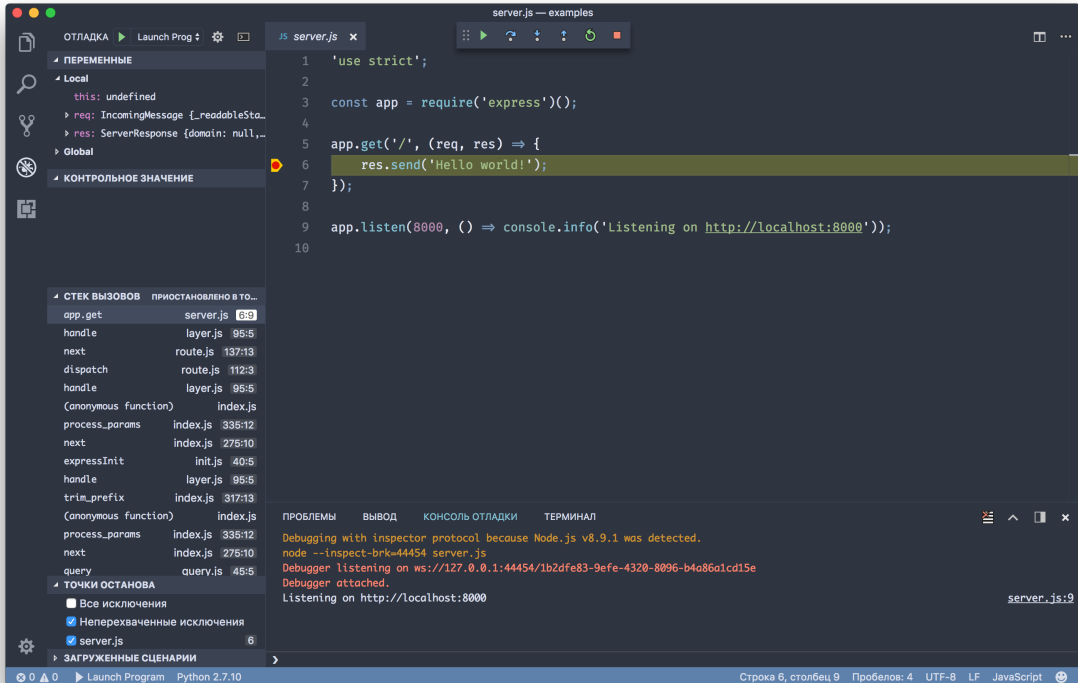
Для того чтобы открыть отладчик нужно:

1. Перейти на вкладку **Отладка** (Ctrl+Shift+D)
2. Создать или выбрать конфигурацию для отладки
3. Запустить отладчик нажав **Начать отладку** (F5)

Базовая конфигурация для отладки в VS Code

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "type": "node",  
      "request": "launch",  
      "name": "Launch Program",  
      "program": "${workspaceFolder}/server.js"  
    }  
  ]  
}
```

Отладка в VS Code



Почитать про отладку

[debug](#)

[npmjs.com](#)

[Debugger](#)

[nodejs.org](#)

[Debugging Node.js with Chrome DevTools](#)

Paul Irish

[How to Debug Node.js with the Best Tools Available](#)

Gergely Nemeth

Профилирование

Профилирование - это сбор характеристик
работы программы для дальнейшего
анализа

«Преждевременная оптимизация — корень
всех зол»

Дональд Кнут

Сначала сделайте так, чтобы ваш код работал

И только потом, чтобы он работал быстро
(если это нужно)

Инструменты

1. `node --prof + node --prof-process`

2. `v8-profiler`

3. `0x + cpuprofilify`

4. `node-report`

5. `Chrome DevTools + node --inspect`

...


```
const crypto = require('crypto');
const app = require('express')();

function hash(password) {
  return crypto.pbkdf2Sync(
    password, 'salt', 100000, 512, 'sha512'
  ).toString();
}

app.get('/', (req, res) => res.send(hash('p@ssw0rd')));

app.listen(8000);
```

Профилирование в Chrome DevTools

В режиме отладки в Chrome DevTools нужно перейти на вкладку **profiler** и нажать **Start**

После этого профилировщик начнет собирать информацию по ходу работы нашего кода

Чтобы закончить профилирование и перейти к просмотру результатов достаточно нажать **Stop**

Профилирование в Chrome DevTools

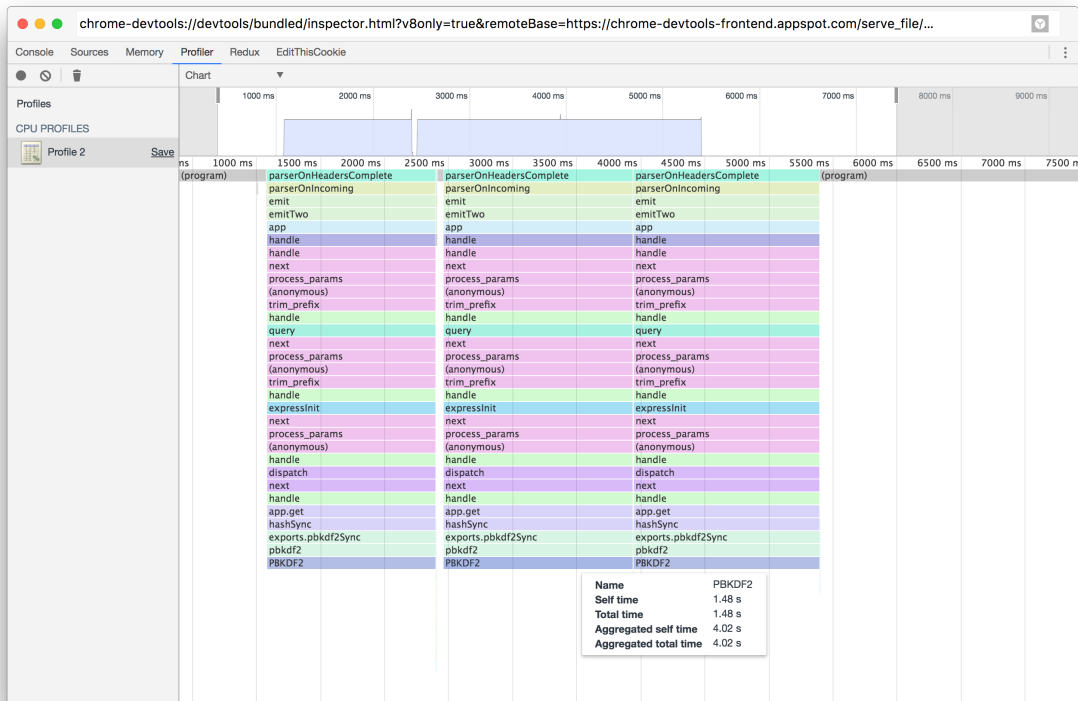
Результаты доступны в трех видах:

Tree (Top Down) - Стек вызовов

Heavy (Bottom Up) - Список всех вызванных функций, вместе с временем их выполнения

Chart - Графическое представление хода исполнения кода (Flame Chart)

Профилирование в Chrome DevTools



Почитать про профилирование

Easy profiling for Node.js Applications

nodejs.org

Node.js Post-Mortem Diagnostics & Debugging

Gergely Nemeth