Node.js

Часть II

Сегодня

Работа с файлами

Потоки (Stream)

Процессы (Child Processes)

Потоки (Worker Threads)

Отладка

Работа с файлами

Buffer

Массив байт

Класс для работы с бинарными данными

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

Каждое число представляет байт

Buffer

Инициализация

```
// Выделяем буфер заданного размера
const buffer = Buffer.alloc(5);

console.log(buffer);

// <Buffer 00 00 00 00 00>
```

Buffer

Инициализация

```
// Быстрое выделение буфера
const buffer = Buffer.allocUnsafe(5);

console.log(buffer);

// <Buffer 00 00 00 00 00>
```

Buffer

Инициализация

```
// Выделяем буфер заданного размера, заполняем данными
const buffer = Buffer.alloc(5);

buffer.fill('ab')

console.log(buffer);

// <Buffer 61 62 61 62 61>
```

Buffer

Инициализация

```
// Выделяем буфер заданного размера, заполняем данными
const buffer = Buffer.alloc(5, 'ab');



console.log(buffer);

// <Buffer 61 62 61 62 61>
```

Buffer

Инициализация

```
// Создаем буфер из переданного объекта
const buffer = Buffer.from('Hello, world!');

console.log(buffer);

// <Buffer 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21>
```

Buffer

Кодировки

```
const msg = Buffer.from([
    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('ucs-2');
// Яндекс
```

Работа с файлами

Чтение

```
import fs from 'fs';

fs.readFile(__filename, (error: Error, data: Buffer) => {
    console.log(data);
});
```

Работа с файлами

Чтение

```
import fs from 'fs';

fs.readFile(__filename, (error: Error, data: Buffer) => {
    console.log(data);
});
```

fs – модуль для работы с файловой системой

Работа с файлами

Чтение

```
import fs from 'fs';

fs.readFile(__filename, (error: Error, data: Buffer) => {
    console.log(data);
});
```

__filename – полный путь до текущего файла

Работа с файлами

Чтение

```
import fs from 'fs';

fs.readFile(__filename, (error: Error, data: Buffer) => {
    console.log(data);
});
```

Работа с файлами

Чтение

```
import fs from 'fs';

fs.readFile(__filename, (error: Error, data: Buffer) => {
    console.log(data);
});
```

Работа с файлами

Чтение

```
import fs from 'fs';

fs.readFile(__filename, (error: Error, data: Buffer) => {
    console.log(data);
});

// <Buffer 69 6d 70 6f 72 74 20 2a ... 110 more bytes>
```

Работа с файлами

Из буфера в текст

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

fs.readFile(__filename, (error: Error, data: Buffer) => {
     console.log(data.toString());
});

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

Работа с файлами

Запись

```
import fs from 'fs';

const data = 'Hello, world!';

fs.writeFile('file.txt', Buffer.from(data), (error: Error) => {
    if (err) {
        process.exit(1);
    }
});
```

Работа с файлами

Запись

```
import fs from 'fs';

const data = 'Hello, world!';

fs.writeFile('file.txt', data, 'utf-8', (error: Error) => {
    if (err) {
        process.exit(1);
    }
});
```

Работа с файлами

Запись

```
import fs from 'fs';

const data = 'Hello, world!';

fs.writeFile('file.txt', data, (error: Error) => {
    if (err) {
        process.exit(1);
    }
});

fs.appendFile('file.txt', 'And also hello to UrFU', 'utf-8');
```

Работа с файлами

Синхронные аналоги

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

fs.writeFileSync(...);

fs.appendFileSync(...);
```

Блокируют поток выполнения программы

Работа с файлами

Promise

```
fs.promises.readFile(...);

fs.promises.writeFile(...);

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

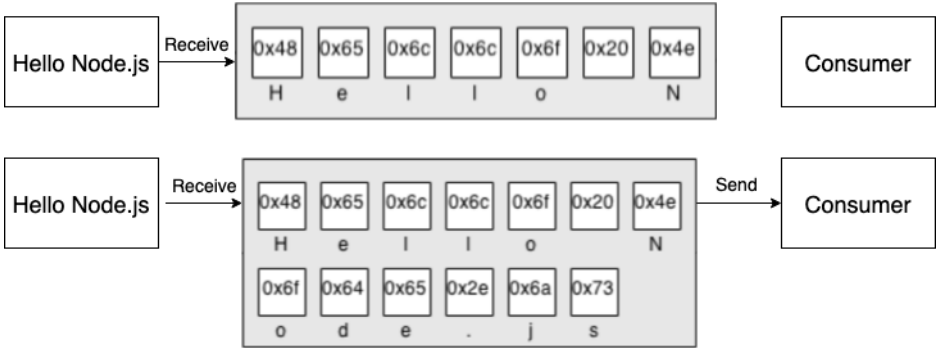
Stability: 1 - Experimental

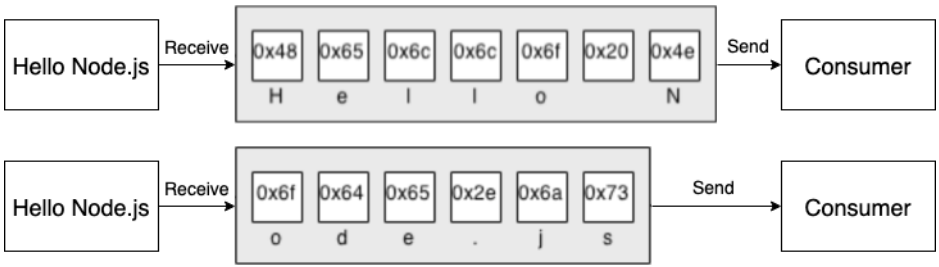
Работа с файлами

```
fs.readFile(__filename, (error: Error, data: Buffer) => {
    console.log(data);
});
```

Данные записываются в Buffer

Только когда весь файл прочитан, данные передаются в обработчик


Hello Node.js
Receive
0x48
0x65
0x6c
0x6c
0x6f
0x20
0x4e
H
e
l
l
o
N
Consumer
Hello Node.js
Receive
0x48
0x65
0x6c
0x6c
0x6f
0x20
0x4e
H
e
l
l
o
N
0x6f
0x64
0x65
0x2e
0x6a
0x73
o
d
e
.
j
s
Send
Consumer


Hello Node.js
Receive
0x48
0x65
0x6c
0x6c
0x6f
0x20
0x4e
H
e
l
l
o
N
Send
Consumer
Hello Node.js
Receive
0x6f
0x64
0x65
0x2e
0x6a
0x73
o
d
e
.
j
s
Send
Consumer

Потоки

Stream

Потоки

Данные готовы для обработки, как только будет прочитан первый chunk

✓ Экономия ресурсов

✓ Экономия времени

Архиватор

```
import fs from 'fs';
import zlib from 'zlib';

fs.readFile(__filename, (_, fileBuffer: Buffer) => {
    zlib.gzip(fileBuffer, (_, gzipBuffer: Buffer) => {
        fs.writeFile(__filename + '.gz', gzipBuffer, () => {
            console.log('Success');
        });
    });
});
```

Еще одна проблема

Buffer в V8 не может быть больше 0x3FFFFFFF bytes ~1 Gb

File size is greater than possible Buffer: 0x3FFFFFFF bytes

Архиватор

Используем потоки

```
import fs from 'fs';
import zlib from 'zlib';

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

Архиватор

Используем потоки

```
import fs from 'fs';
import zlib from 'zlib';

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

Архиватор

Используем потоки

```
import fs from 'fs';
import zlib from 'zlib';

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

Архиватор

Используем потоки

```
import fs from 'fs';
import zlib from 'zlib';

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

Архиватор

Используем потоки

```
import fs from 'fs';
import zlib from 'zlib';

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

Архиватор

Используем потоки

```
import fs from 'fs';
import zlib from 'zlib';

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

Архиватор

Используем потоки

```
import fs from 'fs';
import zlib from 'zlib';

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

Stream – реализует методы EventEmitter

Виды потоков

Readable - для чтения

Writable - для записи

Duplex - для чтения и записи

Transform - Duplex, но с преобразованием

Виды потоков

Readable

```
const s: fs.ReadStream = fs.createReadStream(__filename);

http.request({ hostname: 'yandex.ru' })
    .on('response', (res: http.IncomingMessage) => {
        res.on('data', (chunk: Buffer) => {});
        res.on('end', () => {});
    });
```

- data – при получении чанка данных
- end – при завершении данных в потоке
- close – при закрытии потока
- error – в случае ошибки

Виды потоков

Writable

```
const s: fs.WriteStream = fs.createWriteStream(__filename);

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

error – в случае ошибки

Виды потоков

Duplex, Transform

Сокеты

Криптография

Сжатие и кодирование

Перерыв

Процессы

Child Process

Запускаем процесс

child_process.exec

```
import { exec } from 'child_process';

const ls = exec('ls -l /home',
    (error: Error, stdout: string, stderr: string) => {
        if (error) {
            console.error(`exec error: ${error}`);
        }

        console.log(`stdout: ${stdout}`);
        console.log(`stderr: ${stderr}`);
    });
```

Запускаем процесс

child_process.exec

```
import { exec } from 'child_process';

const ls = exec('ls -l /home',
    (error: Error, stdout: string, stderr: string) => {
        if (error) {
            console.error(`exec error: ${error}`);
        }

        console.log(`stdout: ${stdout}`);
        console.log(`stderr: ${stderr}`);
    });
```

Запускаем процесс

child_process.exec

```
import { exec } from 'child_process';

const ls = exec('ls -l /home',
    (error: Error, stdout: string, stderr: string) => {
        if (error) {
            console.error(`exec error: ${error}`);
        }

        console.log(`stdout: ${stdout}`);
        console.log(`stderr: ${stderr}`);
    });
```

Запускаем процесс

child_process.exec

```
import { exec } from 'child_process';

const ls = exec('ls -l /home',
    (error: Error, stdout: string, stderr: string) => {
        if (error) {
            console.error(`exec error: ${error}`);
        }

        console.log(`stdout: ${stdout}`);
        console.log(`stderr: ${stderr}`);
    });
```

Запускаем процесс

child_process.exec

```
import { exec } from 'child_process';

const ls = exec('ls -l /home',
    (error: Error, stdout: string, stderr: string) => {
        if (error) {
            console.error(`exec error: ${error}`);
        }

        console.log(`stdout: ${stdout}`);
        console.log(`stderr: ${stderr}`);
    });
```

Запускаем процесс

child_process.spawn

```
import { spawn } from 'child_process';

const ls = spawn('ls', ['-l', '/home']);

ls.stdout.on('data', (data: string) => {
  console.log(`stdout: ${data}`);
});

ls.stderr.on('data', (data: string) => {
  console.log(`stderr: ${data}`);
});

ls.on('close', (code: number) => {
```

Запускаем процесс

child_process.spawn

```
import { spawn } from 'child_process';

const ls = spawn('ls', ['-l', '/home']);

ls.stdout.on('data', (data: string) => {
  console.log(`stdout: ${data}`);
});

ls.stderr.on('data', (data: string) => {
  console.log(`stderr: ${data}`);
});

ls.on('close', (code: number) => {
```

Запускаем процесс

child_process.spawn

```
import { spawn } from 'child_process';

const ls = spawn('ls', ['-l', '/home']);

ls.stdout.on('data', (data: string) => {
  console.log(`stdout: ${data}`);
});

ls.stderr.on('data', (data: string) => {
  console.log(`stderr: ${data}`);
});

ls.on('close', (code: number) => {
```

Запускаем процесс

child_process.spawn

```
import { spawn } from 'child_process';

const ls = spawn('ls', ['-l', '/home']);

ls.stdout.on('data', (data: string) => {
  console.log(`stdout: ${data}`);
});

ls.stderr.on('data', (data: string) => {
  console.log(`stderr: ${data}`);
});

ls.on('close', (code: number) => {
```

ChildProcess

События

close – при закрытии ввода/вывода

error – при ошибке работы с процессом

exit – при завершении процесса

message – новое сообщение

Потоки

Worker Threads

JavaScript

Живем в одном потоке

```
import fs from 'fs';

fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
    const lines = data.split('\n');

    for (const line of lines) {
        someComputation(line);
    }
});
```

JavaScript

Живем в одном потоке

```
import fs from 'fs';

fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
    const lines = data.split('\n');

    for (const line of lines) {
        someComputation(line);
    }
});
```

Worker Threads

Hello, world!

```
import { Worker, isMainThread, parentPort } from 'worker_threads';

if (isMainThread) {
    const worker = new Worker(__filename);

    worker.on('message', (message: string) => {
        console.log(message);
    });
} else {
    parentPort.postMessage('Hello, world!');
}
```

Worker Threads

Hello, world!

```
import { Worker, isMainThread, parentPort } from 'worker_threads';

if (isMainThread) {
    const worker = new Worker(__filename);

    worker.on('message', (message: string) => {
        console.log(message);
    });
} else {
    parentPort.postMessage('Hello, world!');
}
```

Worker Threads

Hello, world!

```
import { Worker, isMainThread, parentPort } from 'worker_threads';

if (isMainThread) {
    const worker = new Worker(__filename);

    worker.on('message', (message: string) => {
        console.log(message);
    });
} else {
    parentPort.postMessage('Hello, world!');
}
```

Worker Threads

Hello, world!

```
import { Worker, isMainThread, parentPort } from 'worker_threads';

if (isMainThread) {
    const worker = new Worker(__filename);

    worker.on('message', (message: string) => {
        console.log(message);
    });
} else {
    parentPort.postMessage('Hello, world!');
}
```

Worker Threads

Hello, world!

```
import { Worker, isMainThread, parentPort } from 'worker_threads';

if (isMainThread) {
    const worker = new Worker(__filename);

    worker.on('message', (message: string) => {
        console.log(message);
    });
} else {
    parentPort.postMessage('Hello, world!');
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';

if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Worker Threads

Передача аргументов

```
import fs from 'fs';
import { Worker, isMainThread, parentPort, workerData } from 'worker_threads';


if (isMainThread) {
    fs.readFile(__filename, 'utf-8', (error: Error, data: string) => {
        const lines = data.split('\n');

        for (const line of lines) {
            const worker = new Worker(__filename, { workerData: line });

            worker.on('message', console.log)
        }
    });
} else {
    parentPort.postMessage(someComputation(workerData));
}
```

Отладка

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

Отладка в VS Code

Отладка в VS Code

Для того чтобы открыть отладчик нужно:

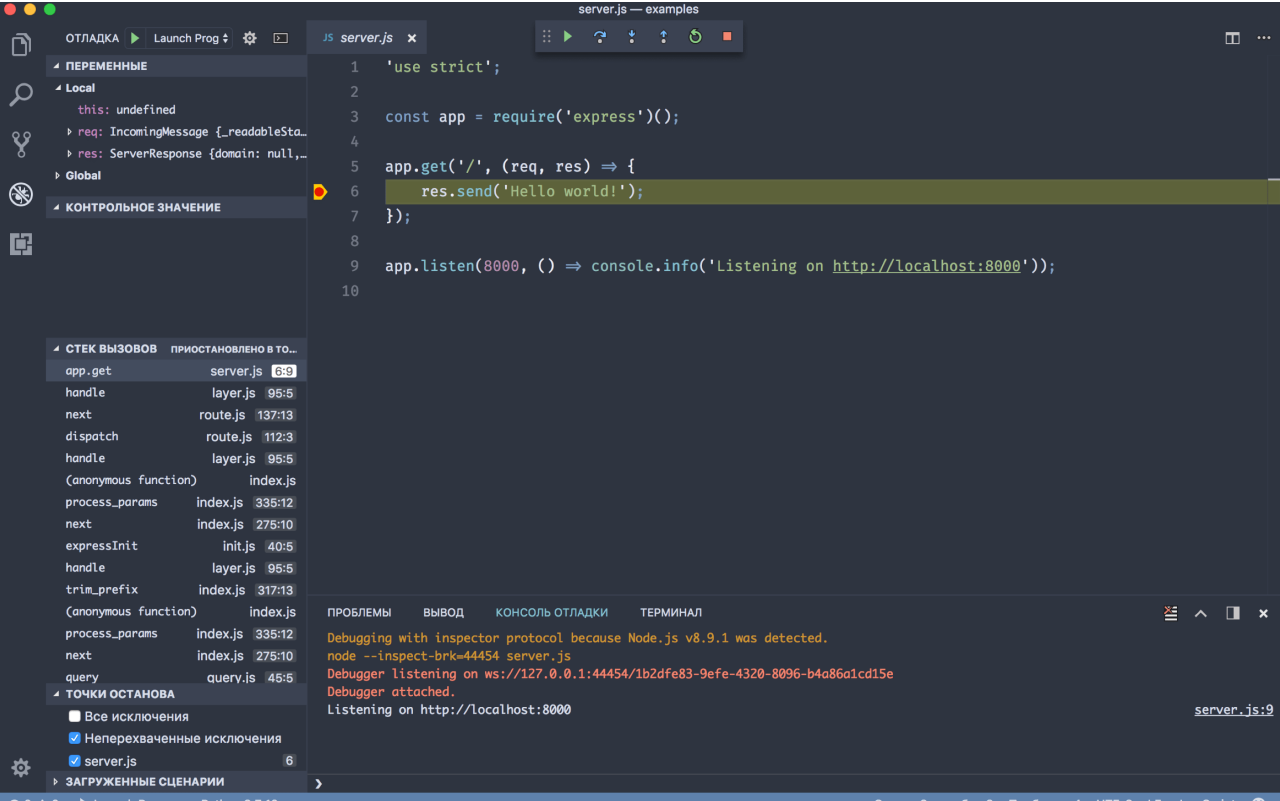
1. Перейти на вкладку Отладка (Ctrl+Shift+D)

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"
        }
    ]
}
```


server.js — examples
ОТЛАДКА
Launch Prog
server.js
ПЕРЕМЕННЫЕ
Local
this: undefined
req: IncomingMessage {_readableSta…
res: ServerResponse {domain: null,…
Global
КОНТРОЛЬНОЕ ЗНАЧЕНИЕ
СТЕК ВЫЗОВОВ
ПРИОСТАНОВЛЕНО В ТО…
app.get server.js 6:9
handle layer.js 95:5
next route.js 137:13
dispatch route.js 112:3
handle layer.js 95:5
(anonymous function) index.js
process_params index.js 335:12
next index.js 275:10
expressInit init.js 40:5
handle layer.js 95:5
trim_prefix index.js 317:13
(anonymous function) index.js
process_params index.js 335:12
next index.js 275:10
query querv.js 45:5
ТОЧКИ ОСТАНОВА
Все исключения
Неперехваченные исключения
server.js 6
ЗАГРУЖЕННЫЕ СЦЕНАРИИ
1 'use strict';
2
3 const app = require('express')();
4
5 app.get('/', (req, res) ⇒ {
6 res.send('Hello world!');
7 });
8
9 app.listen(8000, () ⇒ console.info('Listening on http://localhost:8000'));
10
ПРОБЛЕМЫ
ВЫВОД
КОНСОЛЬ ОТЛАДКИ
ТЕРМИНАЛ
Debugging with inspector protocol because Node.js v8.9.1 was detected.
node --inspect-brk=44454 server.js
Debugger listening on ws://127.0.0.1:44454/1b2dfe83-9efe-4320-8096-b4a86a1cd15e
Debugger attached.
Listening on http://localhost:8000
server.js:9
0 0
Launch Program
Python 2.7.10
Строка 6, столбец 9
Пробелов: 4
UTF-8
LF
JavaScript

Ссылки

fs и Buffer

Stream и как их писать

Child Process и Worker Threads

Про отладку

Вопросы?

Спасибо!