ОНЛАЙН-ОБРАЗОВАНИЕ

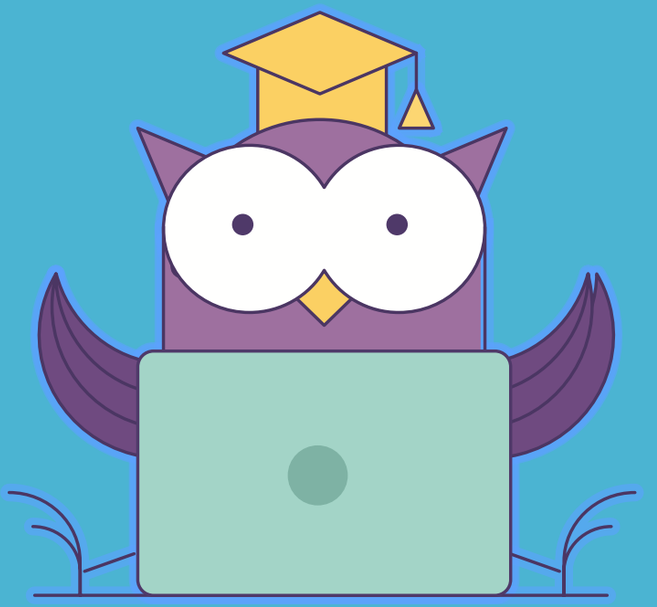
OTUS

Modern JavaScript Frameworks

Node in Production

Александр Коржиков

Как меня слышно и видно?



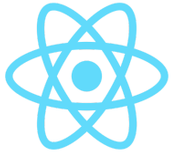
> Напишите в чат

+ если все хорошо

– если есть проблемы со звуком или с видео


JS


node
JS

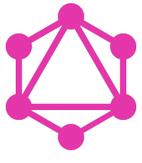



A


npm


TS


express

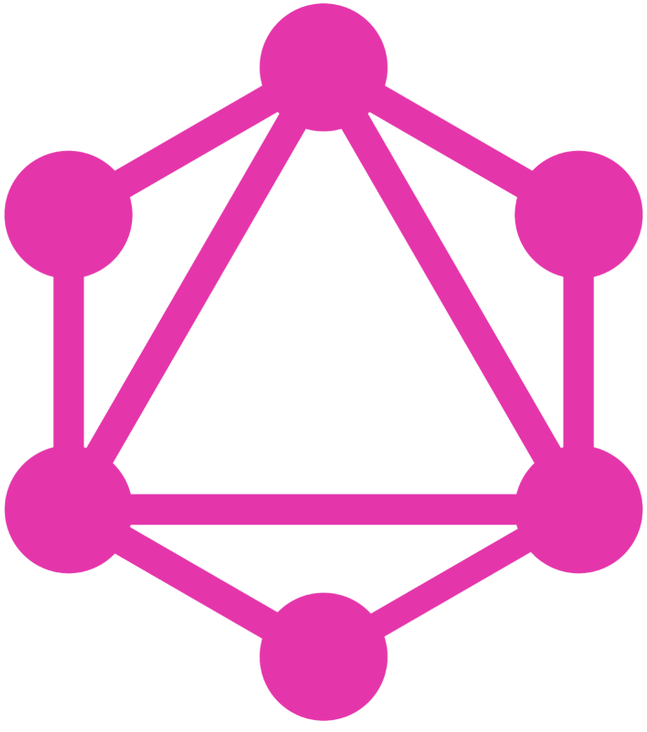


Предложение

- Среда **15.05** - JavaScript - Работа с браузером (было **16.05**)
- Пятница **17.05** - встреча в Отусе "Что должен знать разработчик JavaScript"
  - Рассмотрим основные требования, актуальные для трудоустройства
  - Разберем популярные вопросы по технологиям для интервью"
- Понедельник **21.05** "Custom Elements" (было **20.05** "Обзор Web Components")

Темы предыдущего занятия

- GraphQL
- Apollo



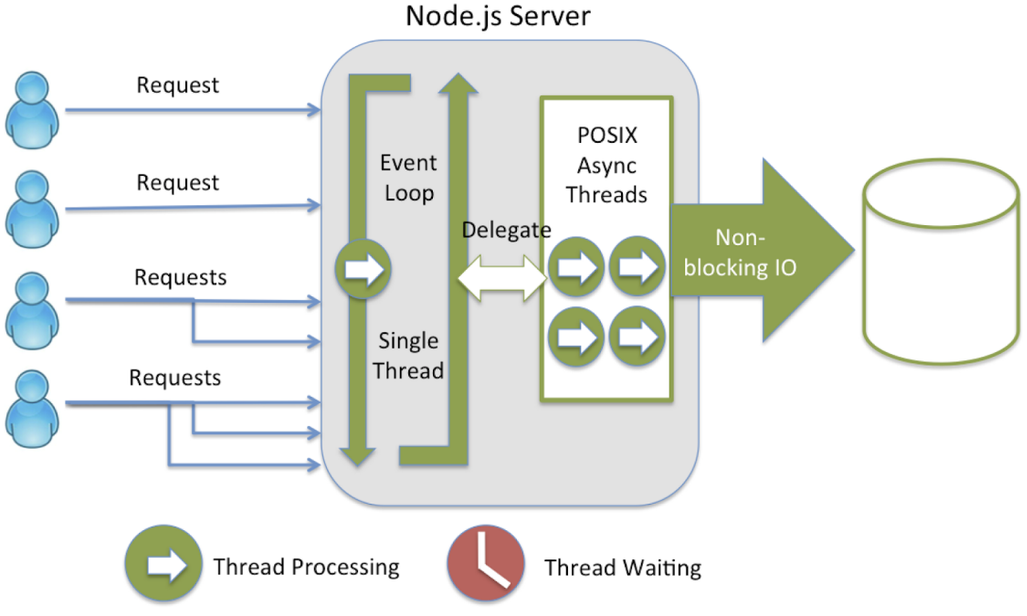
- Web Assembly
- Processes
  - Process
  - child_process
  - fork, exec, spawn
  - PM2
- Node Summary

- WebAssembly.org
- Node.js Official Documentation
- An Introduction to libuv

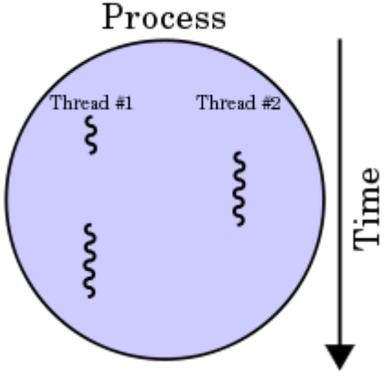
Павел Асташкин - выпускник первого потока

Web Assembly Q&A

При старте **Node** инициализируется **Event Loop**, исполняется переданный скрипт вместе с синхронными вызовами (**API, setTimeout, process.nextTick**), после чего происходит работа **Event Loop**



- **Process** - программа, которая выполняется в текущий момент
- **System Call** - запросы в процесс ядра для получения сервиса
  - Process Creation & Management
  - File Access
  - Networking
  - Memory Management
- **Thread** - частичная копия исходного процесса с доступом к его ресурсам



Properties

- **pid** - процесса
- **ppid** - родительского процесса
- **gid, uid**
- environment variables
- **cwd**
- terminal, priority
- **state**

Tools

- kill - отправка сигнала процессу

Дерево процессов:

- htop
- ps
- pstree

- popen() - создание нового процесса
- fork() - создание копии родительского процесса
- exec() - замещение текущего процесса
- clone() - создание потока (thread)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
  int status
  printf("before \n")
  int id = fork()
  printf("parent id %d\n", getppid())
  printf("my id %d\n", getpid())
  printf("child id %d\n", id)
  printf("--- \n")                    // что будет выведено на экран?
  return 0
}
```

Модуль **child_process** экспортирует

- **spawn()**,
- **fork()**,
- **exec()**,
- **execFile()** и синхронные альтернативные функции

Методы возвращают **ChildProcess** объект, являющийся **EventEmitter**

- **command**
- **[arguments]** - command-line arguments
- **[options]** - **spawn()** settings

```
const { spawn }
  = require('child_process')

spawn('ps', ['ax'])
```

Options:

- **cwd** - (current) working directory
- **env** - переменные окружения
- **detached** - отцепить от родительского процесса
- **stdio** - отношение потоков ввода вывода между процессами

Вопрос

```
const spawn = require('child_process').spawn
let ls = spawn('ls', ['-lh', '.'])
ls.stdout.on('readable', function() {
    let d = this.read()
    d && console.log(d.toString())
})
ls.on('close', code => {
    console.log(`child process exited with code: ${code}`)
})
```

Что здесь происходит?

- **stdio** - потоки **stdin, stdout, stderr** - **'pipe', 'inherit', 'ignore'**
- **unref(), ref()** - отношение с родительским Event Loop
- **send(), on()** - отправить, подписаться на сообщения
- **kill()** - **'SIGTERM'**

Events

- **'exit'** - процесс заканчивается
- **'close'** - закрываются потоки
- **'message'** - сообщение от **process.send()**

Demo Signal

```
setInterval(() => {
  console.log('hello')
}, 500);

process.addListener('SIGINT', () => {
  console.log('got it')
  process.exit(0)
})
```

- **spawn[Sync]()** - **pipe + fork + shell**
- **fork()**
  - **IPC** communication channel
- **exec[Sync]()** - полная команда с аргументами
  - **[callback]**
  - **timeout** - ограничить время исполнения
- **execFile[Sync]()** - без **shell**
  - **[callback]**

```
console.log('before')

const { fork } = require('child_process')

const pid = process.pid

if (process.argv[process.argv.length - 1] === 'true') return

const childId = fork('./fork', [true])

console.log('parent id %d', process.ppid)
console.log('my id', pid)
console.log('child id', childId.pid)

console.log('after')
```

Что делать с **promisify()**?

```
const util = require('util')
const exec = util.promisify(require('child_process').exec)

async function lsExample() {
  const { stdout, stderr } = await exec('ls')
  console.log('stdout:', stdout)
  console.log('stderr:', stderr)
}
lsExample()
```

Встроенная возможность расшаривать ресурсы

```
const cluster = require('cluster')
const http = require('http')
const numCPUs = require('os').cpus().length

if(cluster.isMaster) {
   for(let i = 0; i < numCPUs; i++) {
      cluster.fork()
   }
}

if(cluster.isWorker) {
   http.createServer((req, res) => {
      res.writeHead(200)
      res.end(`Hello from ${cluster.worker.id}`)
   }).listen(8080)
}
```

```
const { Worker, isMainThread } = require('worker_threads')

if (isMainThread) {
  new Worker(__filename)
  console.log(process.pid)
} else {
  console.log(isMainThread)
  setTimeout(() => {
    console.log('Inside Worker!')
    console.log(process.pid)
  }, 1e4)
}
```

- **workerData** - копия данных, переданных в **constructor**
- **ArrayBuffer, SharedArrayBuffer** для общих данных

Advanced, production process manager for Node.js

```
pm2 start http.js -i max
```

```
var http = require('http')

var server = http.createServer(function(req, res) {
  res.writeHead(200)
  res.end('hey')
}).listen(process.env.PORT || 8000, function() {
  console.log('App listening on port %d', server.address().port)
})
```

Q&A

- Какие недостатки у **Node**?

- Design Mistakes in Node


NOT THIS TIME!

Спасибо за внимание!

Пожалуйста, пройдите опрос

в личном кабинете

- Сколько времени Вы тратите на самостоятельную работу?