NODE JS

Версия NodeJS: **node-v**

Помощь: **.help**

Запуск NodeJS: **node**

После запуска **node** открывается консоль, в которой можно писать JS и он будет выполняться. Выйти из консоли можно с помощью команды **.exit** или сочетанием клавиш **CTRL+C.**

Пользоваться только консолью неудобно, поэтому лучше в ноде запускать файлы.

После создания файла index.js, мы можем запустить его в ноде, для этого в командной строке пишем: **node index** (расширение js можно не писать)

NPM

npm – менеджер пакетов

версия: **npm -v**

**Инициализация проекта**

Вызывается командой **npm init**

На этой стадии инициализируется проект, создается файл **package.json:**

{  
 "name": "nodejscourse",  
 "version": "1.0.0",  
 "description": "Node JS course for beginners",  
 "main": "index.js",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "repository": {  
 "type": "git",  
 "url": "git+https://github.com/Smovg-by/nodejs\_minin.git"  
 },  
 "keywords": [  
 "node",  
 "nodejs"  
 ],  
 "author": "Uladzimir Smouzh <smovg1986@gmail.com>",  
 "license": "ISC",  
 "bugs": {  
 "url": "https://github.com/Smovg-by/nodejs\_minin/issues"  
 },  
 "homepage": "https://github.com/Smovg-by/nodejs\_minin#readme"  
}

В этом файле мы будем хранить список зависимостей для приложения.

**Установка пакетов (зависимостей)**

Вызывается командой **npm i <имя пакета>** (можно npm install <имя пакета>).

При установке первого пакета создается папка node modules

В этой папке ставятся папки зависимостей, в каждой есть тоже файл package.json. *Т.е. каждый пакет (зависимость) тянет за собой и другие зависимости.*

***Поле devDependencies***

Можно поставить пакеты, которые будут нужны только на стадии разработки. Установка таких пакетов происходит с помощью команды:

**npm i** <имя пакета> **-D**

**Импорт пакетов(модулей)**

Импорт пакетов делается с помощью слова **require()**. Свойства будут доступны по как методы переменной, например:

const chalk = require('chalk');

Можно создать свой модуль. Создадим файл data.js и сделаем его импортируемым модулем с помощью **module.exports()**:

const text = 'Hello Node JS from data.js';  
  
module.exports = text;

**Встроенные модули Node JS**

Их можно посмотреть на оф сайте в доках для текущей версии, например:

<https://nodejs.org/dist/latest-v14.x/docs/api/>

**- Модуль PATH**

Метод **basename** – выводим имя файла

*console*.log("Filename: ", path.basename(\_\_filename));

Метод **dirname** – выводит абсолютный путь к директории и ее имя

*console*.log("Directory name: ", path.dirname(\_\_filename));

Метод **extname** – выводит расширение файла

*console*.log("File Extension", path.extname(\_\_filename));

Метод **parse** – универсальный метод, выводит объект с данными

*console*.log ('Parse: ', path.parse(\_\_filename));

Parse: {

root: 'C:\\',

dir: 'C:\\Users\\user\\Desktop\\nodeJSCourse\\demo',

base: 'path.js',

ext: '.js',

name: 'path'

}

Метод **join** – создает путь к файлу

*console*.log(path.join(\_\_dirname, 'server', 'index.html'));

**вывод: C:\Users\user\Desktop\nodeJSCourse\demo\server\index.html**

**- Модуль FS (file system)**

Метод **mkdir** – создает директорию.

**!!!** Node JS является однопоточным, поэтому предлагает создание папок/файлов СИНХРОННЫМ методом и АСИНХРОННЫМ методом. В данном случае **mkdir** является *асинхронным методом*, поэтому мы должны обязательно использовать колл-бек. Также можно воспользоваться СИНХРОННЫМ методом **mkdirSync, НО ЭТО БУДЕТ БЛОКИРОВАТЬ ПОТОК.**

Чтобы воспользоваться методом, нам понадобится библиотека path, т.к. **первым параметром передается путь**, **вторым имя папки**, **третьим идет колл-бек (если используется асинхронный метод).**

const fs = require('fs'); // fs - filesystem  
const path = require('path');  
  
fs.mkdir(path.join(\_\_dirname, 'test'), (*err*) => {  
 if (*err*) {  
 throw *err*;  
 }  
 *console*.log('Directory created')  
});

Метод **writeFile** – создает файл

Может перетирать существующие файлы!!!

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.writeFile(filePath, 'Hello NODE JS again', (*err*)=>{  
 if (*err*) {  
 throw *err*;  
 }  
 *console*.log('File was created')  
});

Метод **appendFile** – добавляет содержимое к существующему файлу

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.appendFile(filePath, '\n Hello NODE JS again and again', (*err*)=>{  
 if (*err*) {  
 throw *err*;  
 }  
 *console*.log('File was updated')  
});

Метод **readFile** – чтение содержимого файла. Возвращает ОБЪЕКТ БУФЕРА

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.readFile(filePath, (*err*, *content*) => {  
 if (*err*) {  
 throw *err* }  
 *console*.log(*content*)  
});

Есть несколько способов преобразовать буферные данные в строку.

1. **Buffer.from(content)**

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.readFile(filePath, (*err*, *content*) => {  
 if (*err*) {  
 throw *err* }  
 const data = Buffer.from(*content*);  
 *console*.log('Content', data.toString())  
});

1. **Использование кодировки (предпочтительнее)**

const filePath = path.join(\_\_dirname, 'test', 'text.txt');  
  
fs.*readFile*(filePath, 'utf-8',(*err*, *content*) => {  
 if (*err*) {  
 throw *err* }  
 console.log('Content', *content*)  
});

**- Модуль OS**

Метод **platform()** – название ОС

*const* os = require('os')

console.log('Operation system: ', os.platform());

Метод **arch()** – архитектура процессора

console.log('CPU architechture: ', os.arch());

Метод **cpus()** – инфо по ядрам (процессорам). Выводится массив объектов.

console.log('all CPUs info: ', os.cpus());

**Другие методы:**

console.log('free memory: ', os.freemem()); // свободная оперативная память

console.log('total memory: ', os.totalmem()); // всего оперативная память

console.log('home directory: ', os.homedir()); // домашняя директория

console.log('how log system is working: ', os.uptime()); // сколько времени включена система

**- Модуль EVENTS**

С помощью этого модуля можно **создавать объекты от данного класса** и с помощью них **диспатчить события**

*const* EventsEmitter = require('events')

*const* emitter = new EventsEmitter() // создаем объект инстанс класса

// создаем событие

emitter.on('anything', *data* *=>* {

  console.log('ON: anything', *data*)

})

// выпускаем событие

emitter.emit('anything', { a: 1 })

emitter.emit('anything', { b: 2 })

// пока не завершится settimeout, nodeJS не завершит работу

setTimeout(() *=>* {

  emitter.emit('anything', { c: 3 })

}, 2000)

**Расширение классов:**

*class* Dispatcher extends EventsEmitter {

  subscribe (*eventName*, *cb*) {

    console.log('subscribe...')

    this.on(*eventName*, *cb*)

  }

  dispatch (*eventName*, *data*) {

    console.log('dispatching...')

    this.emit(*eventName*, *data*)

  }

}

*const* dis = new Dispatcher()

// сначала ставим прослушку события

dis.subscribe('aa', *data* *=>* {

  console.log('ON: aa', *data*)

})

// потом запускаем диспатчер

dis.dispatch('aa', { aa: 22 })

**Создание серверов**

Для создания серверов будем использовать встроенный модуль http

**- Модуль HTTP**

Для создания серверов будем использовать встроенный модуль http

Метод **createServer()** – создание сервера

*const* http = require('http')

*const* server = http.createServer((*req*, *res*) *=>* {

*res*.writeHead(200, {

    'Content-type': 'text/html' // хром парсит html

    // 'Content-type': 'text/plain' // хром парсит обычный текст

  })

*res*.end('<h1>Hello NodeJS!!!</h1>') // здесь то, чем завершится ответ от сервера

})

// запускаем прослушивание сервера на порт 3000

// после запуска файла, можно открыть в браузере порт 3000 и просмотреть а нетворке preview

server.listen(3000, () *=>* {

  console.log('Server has been started...')

})

Внесем изменения в файл package.json

*"scripts"*: {

*"start"*: "node index.js", // создаем скрипт для запуска index.js

*"dev"*: "nodemon index.js" // либа nodemon будет перезапускать сервер при обновлении кода автоматически. Ранее надо было стопать сервер и запускать заново

  },

**Скрипты запускаются с помощью команды:**

**npm run <скрипт>**

**Далее создадим несколько веб страниц, которые будем возвращать с сервера.**

Чтобы возвращать файлы сервера на клиента будем использовать библиотеку fs и path

1. Определяем URL, по которому будем делать запрос

Он лежит в

console.log(*req*.url)

1. Пропишем логику if else и с помощью библиотек отдадим файлы согласно url:

const server = http.*createServer*((*req*, *res*) => {  
  
 if (*req*.url === '/') {  
 fs.*readFile*(path.join(\_\_dirname, 'public', 'index.html'), (*err*, *data*) => {  
 if (*err*) {  
 throw *err* }  
 *res*.writeHead(200, {  
 'Content-type': 'text/html' *// хром парсит html  
 // 'Content-type': 'text/plain' // хром парсит обычный текст* })  
 *res*.end(*data*) *// здесь то, чем завершится ответ от сервера* })  
 } else if (*req*.url === '/contact') {  
 fs.*readFile*(path.join(\_\_dirname, 'public', 'contact.html'), (*err*, *data*) => {  
 if (*err*) {  
 throw *err* }  
 *res*.writeHead(200, {  
 'Content-type': 'text/html' *// хром парсит html  
 // 'Content-type': 'text/plain' // хром парсит обычный текст* })  
 *res*.end(*data*) *// здесь то, чем завершится ответ от сервера* })  
 }  
})

**Оптимизируем двухстраничный сервер и добавим страницу 404 при переходе на несуществующий url .**

const http = require('http')  
const fs = require('fs')  
const path = require('path')  
  
const server = http.*createServer*((*req*, *res*) => {  
  
 let filePath = path.join(\_\_dirname, 'public', *req*.url === '/' ? 'index.html' : *req*.url)  
 const ext = path.extname(filePath)  
  
 if (!ext) {  
 filePath += '.html' // если нужно, добавляем расширение html  
 }  
  
 fs.*readFile*(filePath, (*err*, *content*) => {  
 if (*err*) {  
 fs.*readFile*(path.join(\_\_dirname, 'public', 'error.html'), (*err*, *data*) => {  
 if (*err*) {  
 *res*.writeHead(500)  
 *res*.end('Error')  
 } else {  
 *res*.writeHead(200, {  
 'Content-type': 'text-html'  
 })  
 *res*.end(*data*)  
 }  
 })  
 } else {  
 *res*.writeHead(200, {  
 'Content-type': 'text-html'  
 })  
 *res*.end(*content*)  
 }  
 })  
})  
  
*// запускаем прослушивание сервера на порт 3000  
// после запуска файла, можно открыть в браузере порт 3000 и просмотреть а нетворке preview*server.listen(3000, () => {  
 console.log('Server has been started...')  
})

**Добавим файл стилей**

У стилей .css MIME type будет **text/css**, поэтому когда мы делаем запрос, нам надо проверить расширение файла и в зависимости от этого прописать MIME content type

const http = require('http')  
const fs = require('fs')  
const path = require('path')  
  
const server = http.*createServer*((*req*, *res*) => {  
  
 let filePath = path.join(\_\_dirname, 'public', *req*.url === '/' ? 'index.html' : *req*.url)  
 const ext = path.extname(filePath)  
 *// зададим дефолтный content type* let contentType = "text.html"  
 *// переключаем тип content type В зависимости от переменной  
 // и прокинем переменную contentType ниже в* switch (ext) {  
 case '.css':  
 contentType = 'text/css'  
 break  
 case ".js":  
 contentType = "text/javascript"  
 break  
 default:  
 contentType = 'text/html'  
 }  
  
 if (!ext) {  
 filePath += '.html'  
 }  
  
 fs.*readFile*(filePath, (*err*, *content*) => {  
 if (*err*) {  
 fs.*readFile*(path.join(\_\_dirname, 'public', 'error.html'), (*err*, *data*) => {  
 if (*err*) {  
 *res*.writeHead(500)  
 *res*.end('Error')  
 } else {  
 *res*.writeHead(200, {  
 'Content-type': 'text-html'  
 })  
 *res*.end(*data*)  
 }  
 })  
 } else {  
 *res*.writeHead(200, {  
 *// передаем сюда переменную, тобы в хедер передавался нужный MIME content type* 'Content-type': contentType  
 })  
 *res*.end(*content*)  
 }  
 })  
})  
  
*// запускаем прослушивание сервера на порт 3000  
// после запуска файла, можно открыть в браузере порт 3000 и просмотреть а нетворке preview*server.listen(3000, () => {  
 console.log('Server has been started...')  
})

**Задаем нужный порт**

Сейчас мы передаем порт как фиксированную константу, но если считать параметры консоли, можем динамически задать номер порта.

*// прочитаем порт, если нет задаем по умолчанию 3000*const PORT = process.env.PORT || 3000  
  
*// запускаем прослушивание сервера на порт 3000  
// после запуска файла, можно открыть в браузере порт 3000 и просмотреть а нетворке preview*server.listen(PORT, () => {  
 console.log(`Server has been started on ${PORT}...`)  
})