# Intro

function User(name){

this.name = name;

this.sayHello = () => {

console.log('Hello ', this.name);

};

}

var Jhon = new User('Jhon');

Jhon.sayHello();

PS C:\Users\24Scroll\Desktop\Nodejs\1\Samples\001\_Simple> node main.js

## Module

require('./mod');

console.log('Hello from MAIN module!');

## Global

//Переменные обьявленные внутри модуля, являются локальными для модуля

var x = 10;

function test(){

console.log('Test function, number = ', x);

};

// что бы переменная была доступна в другом модуле, можно сделать ее свойством глобального обьекта

// для доступа к глобальному объекту используется переменная global

global.number = x;

global.func = test;

require('./mod');

console.log(number);

func();

## Exports

//На практике обьект global не используется

//Переменные обьявленные внутри модуля, являются локальными для модуля

var x = 10;

function test(){

console.log('Test function, number = ', x);

};

// для того чтобы сделать переменные доступны в другом модуле, необходимо добавить их в обьект exports

// exports - объект, который возвращается функцией require

exports.number = x;

exports.func = test;

// exports = x; // Error

var obj = require('./mod');

console.log(obj);

## Module

console.log('test module');

module.exports.x = 10;

module.exports.y = 15;

var mod = require('./mod.js');

// Объект module хранит информацию о текущем модуле

console.log(module);

//Module {

// id: '.', - путь к запускаемому файлу

// exports: {}, - обьект который возвращается функцией require

// parent: null, - родительский модуль

// filename: 'D:\\Node\\006\_Module\\main.js', - Абсолютный путь к файлу

// loaded: false, - статус обработки модуля

// children: - дочерние модули

// [ Module {

// id: 'D:\\Node\\006\_Module\\mod.js',

// exports: {},

// parent: [Circular],

// filename: 'D:\\Node\\006\_Module\\mod.js',

// loaded: true,

// children: [],

// paths: [Object] } ],

// paths: - пути по которым происходит поиск модуля

// [ 'D:\\Node\\006\_Module\\node\_modules',

// 'D:\\Node\\node\_modules',

// 'D:\\node\_modules' ] }

## Connect module

function CreateConnection(){

this.connect = () =>{

console.log('Connection established!');

}

}

function testConnection(){

console.log('Test connection...');

new CreateConnection().connect();

}

// проверяем является ли модуль подключаемым или запускаемым

// если модуль подключается, мы возвращаем функцию, если модуль запускается, устанавливаем тестовое соединение

if(module.parent){

module.exports = CreateConnection;

}

else{

testConnection();

}

var db = require('./dbConnection');

var cn = new db();

cn.connect();

## Load modules

/\* Алгоритм поиска модулей при вызыове функции require('имя\_модуля')

1. Если модуль является системным, происходит его загрузка

2. Если 'имя\_модуля' начинается со знака '/' - поиск модуля будет производится в корне файловой системы

3. Если 'имя\_модуля' начинается со знака '/', './', '../' - поиск осуществляется несколькими способами:

3.1 Происходит поиск файла 'имя\_модуля' по указанному пути (см. Загрузка файла)

3.2 Происходит поиск директории имя которой 'имя\_модуля' по указанному пути (см. Загрузка директории

4. Поиск папки node\_modules

4.1 Поиск модуля происходит в директории node\_modules

4.2 Если в текущем каталоге нет каталога node\_modules, происходит переход в каталог уровнем выше, и производится поиск в нее

5. Ошибка, модуль не найден

\*/

// Загрузка файла

// 1. Если 'имя\_модуля' или 'имя\_модуля.js' происходит загрузка JavaScript файла

// 2. Если 'имя\_модуля.json', происходит парсинг файла и загружается JavaScript объект

// 3. Если 'имя\_модуля.node', происходит загрузка бинарного файла

// Загрузка директории

// 1. Если 'имя\_модуля' соответствует имени директории и в ней есть файл package.json, происходит парсинг сожержимого в объект

// и поиск свойства main (в свойстве находится путь к загружаемым модулям).

// Если все прошло успешно, со свойства извлекается значение и происходит загрузка файла (см. Загрузка файла)

// 2. Если предыдущий пункт не выполнен, происходит загрузка файла с именем index (см. Зашрузка индекс файла)

// Загрузка index файла

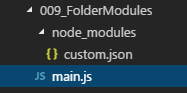
// 1. Если 'имя\_модуля/index' или 'имя\_модуля/index.js' происходит загрузка JavaScript файла

// 2. Если 'имя\_модуля/index.json', происходит парсинг файла и загружается JavaScript объект

// 3. Если 'имя\_модуля/index.node', происходит загрузка бинарного файла

var obj = require('custom'); // Нужно ввести имя подключаемого файла

console.log(obj.name);



Caching

// В node.js при первом подключении модуля происходит выполнение кода в подключаемом модуле, и модуль помещается в кеш

// при повторном подключени того же модуля, среда проверяет путь к подключаемому файлу, если в кеше уже есть такой путь, то возвраoается уже существующий объект

var user = require('./cache')

user.sayHello();

require('./sample');

user.sayHello();

module.exports = {

fname : "Ivan",

lname : "Ivanov",

age : 25,

sayHello : function () {

console.log('Hello! ', 'My name is ', this.fname, ' I\'m ', this.age, ' years old!')

}

}

var test = require('./cache');

test.fname = 'Sergey';

test.age = 30;

# Events

## Emitter

// Для работы с событиями, необходимо подключить модуль 'events'

var evt = require('events');

// Все обьекты которые генерируют события в Node.js должны быть экземплярами класса EventEmitter, или его наследников

var emitter = new evt.EventEmitter();

// Что бы установить слушателя на событие, необходимо воспользоваться методом on(), который доступен на обьекте события

emitter.on('create', function(){

console.log('Folder was created!');

});

// Метод emit предназначен для генерации события

emitter.emit('create');

## Custom events

var evt = require('events');

// Создаем функцию конструктор, которая будет унаследована от функции EventEmitter

// Объект созданый функцией позволит подключиться к файлу и считать из него данные

function ReadFile(){

this.\_file = "";

}

// В прототип ридера записываем объект EventEmitter что бы была возможность генерировать события

ReadFile.prototype = new evt.EventEmitter();

// fileName - имя файла

// callback - функция, которая вызовется после того как данные будут прочитаны

ReadFile.prototype.readDataFromFile = function(path, callback){

this.file = path;

if(typeof callback == 'function'){

this.on('readData', callback)

}

this.\_read();

};

ReadFile.prototype.\_read = function(){

console.log('Loading...');

var someDataFromFile = 'Text text text'; // данные считанные из файла

this.emit('readData', someDataFromFile);

console.log('Data was read.');

}

module.exports.Reader = ReadFile;

// Для работы с событиями, необходимо подключить модуль 'events'

var fileReader = require('./readFile.js');

// Создаем объект ридера

var reader = new fileReader.Reader();

// функция readDataFromFile подключается к файлу, считывает данные и передает в нашу callback функцию

reader.readDataFromFile('file.js', function(responce){

console.log(responce);

});

## Add Listener

'use strict'

var count = 0;

var print = () => {

count++;

console.log('Click - ', count);

}

// Для работы с событиями, необходимо подключить модуль 'events'

var evt = require('events');

var emt = new evt.EventEmitter;

// метод on и addListener добавляют обработчик на событие

emt.on('click', print);

emt.addListener('click', print);

// добавив обработчик с помощью метода once, обраюотчик сработает только один раз и будет удален

emt.once('click', print);

// Вызов обработчиков

emt.emit('click');

emt.emit('click');

## Synchronous

// Обработчики, которые установленны на определенное событие, вызываются синхронно

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

emt.on('event', function(){

console.log("Listener № 1");

});

emt.on('event', function(){

console.log("Listener № 2");

});

emt.on('event', function(){

console.log("Listener № 3");

});

// Генерируем событие myEvent и в функцию обработчик передаем 2 параметра

emt.emit('event');

## Asynchronous

// Обработчики, которые установленны на определенное событие, вызываются синхронно

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

emt.on('event', function(){

setImmediate(() => {

console.log("Listener № 1");

});

});

emt.on('event', function(){

console.log("Listener № 2");

});

emt.on('event', function(){

setImmediate(() => {

console.log("Listener № 3");

});

});

// Генерируем событие myEvent и в функцию обработчик передаем 2 параметра

emt.emit('event');

## Remove Listener

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

// Добавляем обработчик на событие myEvent

emt.on('myEvent', test);

console.log('Listener added!');

// Генерируем событие myEvent

emt.emit('myEvent');

// Удаляем обработчик с события myEvent

emt.removeListener('myEvent', test);

console.log('Listener removed!');

// Повторно генерируем событие myEvent

emt.emit('myEvent');

function test(){

console.log('test function!');

}

## Prepend Listener

// Для работы с событиями, необходимо подключить модуль 'events'

var evt = require('events');

var emt = new evt.EventEmitter;

emt.on('myEvent', function(){

console.log('test string 1');

});

// Метод prependListener добавляет подписчика в начало цепочки

emt.prependListener('myEvent', function(){

console.log('test string 2');

});

emt.once('once', function(){

console.log('First once listener');

});

// Метод prependListener добавляет подписчика в начало цепочки

emt.prependOnceListener('once', function(){

console.log('Seconds once listener');

});

emt.emit('myEvent');

emt.emit('once');

## Loggin event add and remove

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

var count = 0;

// придобавлении нового подписчика, генерируется событие newListener

emt.on('newListener', function(event, listener){

if(event == 'myEvent'){

console.log('myEvent was be added to emt.')

}

});

// когда подписчик удаляется, срабатывает событие removeListener

emt.on('removeListener', function(event, listener){

if(event == 'myEvent'){

console.log('myEvent was be remowed from emt.')

}

});

emt.on('myEvent', test);

console.log(emt.listenerCount('myEvent'));

emt.removeListener('myEvent', test);

console.log(emt.listenerCount('myEvent'));

function test(){

console.log('test event!');

}

## Parametrs

var evt = require('events');

var emitter = new evt.EventEmitter();

emitter.on('click', function(a, b){

console.log('function 1');

console.log(a, b);

});

emitter.on('click', function(){

console.log('function 2');

console.log(arguments);

});

emitter.emit('click', 1, 2);

## Arrow function

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

// Когда вызывается функция обработчик, ключевое слово this внутри нее указывает на объект EventEmitter,

// от этого можно избавиться используя стрелочную функцию

emt.on('myEvent', function(){

console.log('Ordinary function: ');

console.log(this);

});

emt.on('myEvent', () => {

console.log('Arrow function: ')

console.log(this)

});

// Генерируем событие myEvent и в функцию обработчик передаем 2 параметра

emt.emit('myEvent');

## Max Listener

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

// emt.setMaxListeners(20);

for (var index = 0; index < 10; index++) { // попробовать 11

(function(){

var current = index;

emt.on('myEvent', function(){

console.log(current);

});

})();

}

// функция возвразает количество зарегистрированных обработчиков указанного события

console.log('EMT has', emt.listenerCount('myEvent'), 'listeners.');

// Генерируем событие myEvent

emt.emit('myEvent');

## Listeners

var evt = require('events').EventEmitter;

// создаем обьект события

var emt = new evt();

emt.on('myEvent', function(){

console.log('First listener.');

});

emt.on('myEvent', function(){

console.log('Second listener.');

});

// метод listeners возвращает массив функций обработчиков для указаного события

var listeners = emt.listeners('myEvent');

for (var index = 0; index < listeners.length; index++) {

listeners[index]();

}

# Console

## Enviroment

### Cache

require('./test.js');

require('./simple.js');

console.log('Main file is loaded!');

console.log('');

console.log(require.cache); // свойство cache объекта require, хранит закешированные модули

### FileInfo

// \_\_dirname - глобальная переменная, хранит имя текущей директории

console.log("Directory name:", \_\_dirname);

// \_\_filename - глобальная переменная, хранит имя текущего файла

console.log("File name:", \_\_filename);

### Path1

// path модуль ядра, обеспечивающий обработку и преобразование путей к файлам

var path = require('path');

// С помощью метода basename можно получить имя файла с расширением

var filename = path.basename(\_\_filename);

console.log('Current file: ', filename);

// path.resolve - метод преобразующий относительный путь в абсолютный

console.log(path.resolve('./test'));

// метод возвразает расширение файла

var extension = path.extname(\_\_filename);

console.log('Current file, has', extension, 'extensions!');

// path.isAbsolute проверяет, является ли путь абсолютным

var abs = path.isAbsolute(\_\_dirname);

console.log('path.isAbsolute(', \_\_dirname, ') - ', abs);

console.log('path.isAbsolute(','public/myProject/test', ') - ', path.isAbsolute('public/myProject/test'));

### Path2

// path модуль ядра, обеспечивающий работу с путями директорий и файлов

var path = require('path');

var file = '004\_Path\_2.js';

// метод join предназначен для генерации путей, на основе принятых параметров

var filePath = path.join(\_\_dirname, file);

console.log(filePath);

// Парсит заданый путь и возвращает объект

var pathParts = path.parse(filePath);

console.log(pathParts);

console.log('Systems separator:', path.sep);

var objFormated = {

root: 'D:\\',

dir: 'D:\\Node\\003\_ConosleProgram\\001\_Environment',

base: '004\_Path\_2.js',

ext: '.js',

name: '004\_Path\_2'

}

// Метод format генерирует новый путь на основе свойств объекта

var obj = path.format(objFormated);

console.log(obj);

### Process

// Объект process является глобальным и всегда доступен, без подключения модулей.

// Этот объект предоставляет информацию и позволяет управлять текущим процессом

// Свойство хранит информацию о архитектуре процессора

console.log(process.arch);

// Свойство хранит путь по которому запускается Node js, путь по которому находится исполняемый файл, и аргументы командной строки

console.log(process.argv);

// возвразает путь к серверу

console.log(process.execPath);

// метод возвращает версию Node js

console.log(process.version);

// метод возвращает название платформы, на которой выполняется приложение

console.log(process.platform);

// возвращает информацию об использовании памяти

console.log(process.memoryUsage());

// {

// heapTotal: 1826816, - используемая движком V8, память

// heapUsed: 650472, - используемая движком V8, память

// external: 49879 - память используемая внешними библиотеками

// }

## Utils

### Inherits

var utils = require('util');

function Base(){

this.name = "Base function"

}

Base.prototype.say = function(){

console.log('Hello, this is a %s function', this.name);

}

function Derived(){

this.name = "Derived";

}

// Метод utils.inherits позволяет производить наследование

utils.inherits(Derived, Base);

Derived.prototype.getData = function(){

console.log('Some data from Derived function')

}

var derived = new Derived();

derived.getData();

derived.say();

### Format

var utils = require('util');

// %s - Строка

// %d - Число (целое или число с плавающей запятой)

// %j - JSON

// %% - символ '%'

var user = {

name : "Ivan",

age : 25,

salary : 10000,

bonus: 15

}

var str = utils.format('Hello, my name is %s. I\'m %d years old! My bonuses from the salary are %d%%', user.name, user.age, user.bonus);

console.log(str);

console.log();

console.log('%j', user);

### Inspect

var utils = require('util');

var user = {

name : "Ivan",

age : 25,

salary : 10000,

bonus: 15

}

var objInfo = utils.inspect(user);

console.log(objInfo);

var point = {

name : "A",

x : 10,

y : 20,

inspect : function(){

return utils.format('Point %s[%d:%d]', this.name, this.x, this.y);

}

}

console.log(point);

### Check type

var utils = require('util');

var obj = { name: "Some object" };

var numb = 10;

var str = "Some string";

var und = null;

var arr = [1,2,3,4,5];

console.log('arr is array:', utils.isArray(arr));

console.log('obj is object:', utils.isObject(obj));

console.log('numb is number:', utils.isNumber(numb));

console.log('str is string:', utils.isString(str));

console.log('und is NULL:', utils.isNull(und));

console.log('und is function:', utils.isFunction(und));

## Console

### Console

// Объект Console, является глобальным и обеспечивает простую отладку приложений в Node.js

// По умолчинию, объект console сконфигурирован на передачу информации в\из process.stdout и process.stdin

console.log('Simple message.');

console.info('Some data'); // алиас функции console.log()

console.error(new Error('Somthig wrong!'));

var name = "---";

console.warn('Danger ${name}!'); // алиас функции console.error()

var obj = {name : "Ivan", age : 25};

console.dir(obj);

### Console time

var arr = [];

// функция console.time(label) позволяет запустить таймер, для замера продолжительности выполнения операции.

console.time('1-st');

for (var index = 0; index < 1000000; index++) {

arr[index] = (index + 1) \* 2;

}

// функция console.timeEnd(label) останавливает таймер который был вызван функцией console.time(label)

console.timeEnd('1-st'); // Время в миллисекундах

### Buffer

// Класс Buffer позволяет работать с потоками двоичных данных.

// Он является глобальным объектом, поэтому не нужно использовать функцию require

// Метод Buffer.alloc - создает буфер, проинициализированный нулями на указанное количество байт

var buff1 = Buffer.alloc(10);

console.log('empty buffer:', buff1);

// Метод Buffer.allocUnsafe создает непроинициализированный буфер

var nibuff = Buffer.allocUnsafe(10);

console.log('uninitialized buffer:', nibuff);

console.log('Lenght of buffer:', buff1.length);

console.log(Buffer.byteLength('hello world', 'utf-8'));

var buff1 = Buffer.from([10, 20, 30, 40, 50, 60, 70, 80, 90]);

var buff2 = Buffer.from('Hello world!!!', 'utf8');

// Возможные значения кодировок

// 'ascii' - только для 7-битных ASCII-cтpoк. Этот метод кодирования очень быстрый, он сбрасывает старший бит символа;

// 'utf' - Uniсоdе-символы UTF-8;

// 'binary' - хранит двоичные данные в строке, используя младшие 8 бит каждого символа.

// 'base64' - строка, закодированная в системе Base64;

// 'hex' - кодирует каждый байт как два шестнадцатеричных символа

var buff3 = Buffer.from([50, 60, 70, 80, 90, 10, 20, 30, 40]);

console.log(buff1[2]);

// buf.compare(target[, targetStart[, targetEnd[, sourceStart[, sourceEnd]]]])

// метод сравнивает два буфера, и возвращает 0 - если буферы равны, 1 если buf > target, -1 если buf < target

console.log('Compare buffer:', buff1.compare(buff2));

console.log('index compare:', buff1.compare(buff3, 5, 9, 0, 4));

// Buffer.concat метод объединяющий несколько буферов в один

var newBuff = Buffer.concat([buff1, buff2]);

console.log(newBuff);

// метод проверяет является ли параметр типом Buffer

console.log(Buffer.isBuffer(newBuff));

var symb = buff2.indexOf('l');

console.log('Position of symbol \'l\' from start', symb);

symb = buff2.lastIndexOf('l');

console.log('Position of symbol \'l\' from end', symb);

// Типизированный массив, который хранит 16 разрядные значения

var arr1 = new Uint16Array(11);

// свойство buffer возвращает буфер на который указывает массив

const buf1 = Buffer.from(arr1.buffer);

var arr2 = new Uint16Array(4);

const buf2 = Buffer.from(arr2.buffer);

var arr4 = new Uint16Array(10);

const buf4 = Buffer.from(arr4.buffer);

// Метод buf.write(string[, offset][, length][, encoding]) - запись данных в буфер, аргументы:

// string - данные в строковом виде

// offset - индекс буфера, с которого начнется запись

// length - количество байтов для записи

// encoding - кодировка; по умолчанию utf-8

buf1.write('123', 2, 2);

buf2.write('1234', 2, 3);

// buf.toString([encoding][, start][, end]) - чтение данных из буфера, аргументы:

// encoding - кодировка; по умолчанию utf-8

// start - индекс, с которого начать чтение данных буфера

// end - индекс, до которого читать данные буфера

var bufData = buf1.toString('utf-8', 2, 3);

console.log(bufData); // 1

// конкатенация буферов: Buffer.concat(list[, totalLength]), аргументы:

// list - список буферов для конкатенации

// totalLength - длина буфера, получаемого в результате конкатенации

var buf3 = Buffer.concat([buf1, buf2], 28);

console.log(buf3.toString()); //12

// копирование буферов: buf.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd]), аргументы:

// targetBuffer - буфер, в который копировать данные

buf2.copy(buf4, 0, 0, 3);

console.log(buf4.toString());

## File Stream

### Fs

var fs = require('fs');

console.log("Going to open file!");

// открыть файл

// open(filename, flag, callback)

// filename - имя файла

// flag - флаг, который указывает каким образом открывать файлы

// r - открыть для чтения. Генерирует исключение при отсутствии файла;

//r+ - открыть для чтения и записи. Генерирует исключение при отсутствии файла;

//rs - открыть для чтения в синхронном режиме;

//rs+ - открыть для чтения и записи в синхронном режиме;

//w - открыть для записи. Если файл не существует, он будет создан. Если файл существует, его содержимое будет очищено;

//w+ - открыть для чтения и записи. Если файл не существует, он будет создан. Если файл существует, его содержимое будет очищено;

//а - открыть для записи в конец файла. Если файл не существует, он будет создан;

//а+ - открыть для чтения и записи в конец файла. Если файл не существует, он будет создан

// callback - функция, которая вызывается после завершения чтения

fs.open('demofile.txt', 'w+', function (err, fd) {

console.log('opening file!');

if (err) {

console.log(err);

}

else {

// Метод write - альтернатива методу writeFile; позволяет записать данные в файл и принимает такие аргументы:

// fd - дескриптор файла

// buffer - данные в виде буфера или строки,

// offset, length - определяют часть буфера, которую следует записать в файл

// position - отступ от начала файла, куда будут записаны данные

// callback - функция, принимающая аргументы err, written, string

fs.write(fd, 'This is the file content!', { encoding: 'utf-8' }, function (err, written, string) {

console.log('writing to file!');

if (err) throw err;

console.log(written); // written - количество байтов, которое потребовалось для записи данных

console.log(string); // строка, записанная в файл

});

var arr = new Uint16Array(1024);

var buf = Buffer.from(arr.buffer);

// Метод read позволяет читать данные из файла, принимает аргументы:

// fd -дескриптор файла

// buffer - буфер, в который будут помещены прочитанные данные

// offset, length - определяют часть буфера, которую следует записать в файл

// position - отступ от начала файла, данные которого считываются

// callback - функция, принимающая аргументы err, bytesRead, buffer

fs.read(fd, buf, 0, buf.length, 0, function (err, bytes) {

console.log('reading from file!');

if (err) throw err;

console.log(bytes);

console.log(buf.slice(0, bytes).toString());

})

// закрыть файл

fs.close(fd, function (err) {

if (err) throw err;

console.log('file closed!');

})

}

});

### writeFile

// Модуль fs предназначен для работы с файлами. методы содержащиеся в модуле имеют синхронную и асинхронную формы

var fs = require('fs');

var utils = require('util');

user = {

fname : "Ivan",

lname : "Ivanov",

age : 30,

position : "developer"

}

console.log('File writing...');

fs.writeFile('text.txt', utils.format('%j', user), function(err){

if(err){

console.log(err);

return;

}

console.log('File was wrote!');

});

### readFile

var fs = require('fs');

fs.exists('text.txt', function(){

fs.readFile('text.txt', {encoding : 'utf-8'}, function(err, data){

if(err){

console.log(err);

return;

}

console.log("it's- "+ data);

var obj = JSON.parse(data);

console.log(obj.fname, obj.lname, obj.age, obj.position);

});

});

### readDir

var fs = require('fs');

fs.readdir('testdir', function(err, filenames){

console.log(filenames);

});

### Watcher

var fs = require('fs');

fs.watch('text.txt', function(event, filename){

console.log('File %s is %s.', filename, event);

});

fs.writeFile('text.txt', 'test string', function(err){

if(err) throw err;

console.log('data was wrote');

});

## Stream

### Read and write stream

// Потоки(Streams) - объекты, которые позволяют считывать данные или записывать данные в источник в непрерывном режиме.

// В NodeJS есть 4 типа потоков:

// Readable - поток, который используется для операций чтения

// Writable - поток, который используется для операци записи

// Duplex - поток, который может быть использован и для операций чтения, и для операций записи

// Transform - тип duplex-потока, в котором выходные данные рассчитываеются на основе входных данных

var fs = require('fs');

var writeData = 'This is the file content!';

// поток для записи данных

// генерирует события error(при ошибке), finish(при завершении текущей операции записи данных)

var writerStream = fs.createWriteStream('output.txt');

writerStream.write(writeData, 'utf-8');

writerStream.end();

writerStream.on('finish', function() {

console.log('Write completed');

});

writerStream.on('error', function(err) {

console.log(err);

});

// поток для чтения данных

// генерирует события error(при ошибке), data(когда данные доступны для чтения),

// end(когда больше нет доступных данных для чтения)

var readData = '';

var readerStream = fs.createReadStream('output.txt');

readerStream.setEncoding('UTF8');

readerStream.on('data', function (chunk) {

readData += chunk;

});

readerStream.on('end', function () {

console.log(readData);

});

readerStream.on('error', function (err) {

console.log(err);

});

### Pipe stream

// Piping streams - перенаправление данных одного потока к другому потоку

var fs = require("fs");

// создать поток для чтения данных

var readerStream = fs.createReadStream('input.txt');

// создать поток для записи данных

var writerStream = fs.createWriteStream('output.txt');

// передача данных потока readerStream потоку writerStream

readerStream.pipe(writerStream);

writerStream.on('finish', function () {

console.log('data written to file output.txt')

});

## Mock url

var url = require('url');

// Строка URL - структурированная строка, состоящая из многочисленных значимых компонентов.

// При применении к ней метода parse возвращается объект URL, содержащий все эти компоненты в качестве свойств

var mockUrl = 'http://user:pass@host.com:8080/p/a/t/h?query=string#hash';

// при вызове url.parse со вторым параметром true

// функция будет парсить параметры запроса(queryString)

var mockUrlObj = url.parse(mockUrl);

console.log(mockUrlObj);

// метод url.format преобразует объект URL в строку URL

var formattedUrl = url.format(mockUrlObj);

console.log(formattedUrl);

// метод url.resolve(from, to) преобразует второй аргумент(целевой URL) по образцу первого аргумента(базовый URL)

var resolvedUrl = url.resolve('http://example.com/one/two', '/two/three');

console.log(resolvedUrl); // 'http://example.com/two/three'

## Timers

// setTimeout(callback, delay[, ...arg])

// callback - функция, которая будет вызвана через указанный интервал времени

// delay - задержка (в миллисекундах)

// ...arg - аргументы, которые будут использованы при вызове callback

var timeout = setTimeout(function () {

console.log('timeout example');

}, 2000)

// clearTimeout(timeout); // отмена setTimeout

// setInterval(callback, delay[, ...arg])

// callback - функция, которая будет вызвана через указанный интервал времени

// delay - задержка (в миллисекундах)

// ...arg - аргументы, которые будут использованы при вызове callback

var counter = 0;

var interval = setInterval(function () {

console.log('interval ' + counter);

counter++;

if (counter > 10) {

clearInterval(interval); // отмена setInterval

}

}, 100)

# Http

## createServer

var http = require('http');

var server = http.createServer();

var server = http.Server();

server.on('request', function(request, response){

console.log('Connection established!');

console.log(request.method);

// Завершает конфигурацию ответ

response.end();

// response.end("Responce is configured!", 'utf8', () => { console.log('finish'); });

});

server.listen(8080)

server.on('listening', function(){

console.log('Server running on port 8080');

});

## write and writeHead

var http = require("http");

var server = http.createServer(function(request, response) {

// writeHead - метод позволяет записать в ответ заголовки и статус код

response.writeHead(200, {"Content-Type": "text/plain"});

// write - метод позволяет создать тело ответа в виде потока writeable Stream

response.write("Hello World");

// end - завершает конфигурациюответа и отправляет его

response.end();

}).listen(8080, function(){

console.log('Server running on port 8080');

});

## request

var http = require('http');

var utils = require('util');

http.createServer(function(req, res){

var requestInfo = utils.format('HTTPVersion: %s \nMethod: %s \nStatus code: %s \nMessage: %s \nURL: %s',

// верисия http протокола

req.httpVersion,

// http глагол

req.method,

// статус код

req.statusCode,

// текстовое описание статус кода

req.statusMessage,

// запрашиваемый ресурс

req.url);

console.log(requestInfo);

console.log();

// headers - свойство содержит объект с заголовками

for (var key in req.headers) {

console.log(key, ":", req.headers[key]);

}

// отправляем ответ клиенту

res.end();

}).listen(8080, 'localhost');

## headers

var http = require('http');

http.createServer(function(req, res){

// возвращает массив c заголовками, четные элементы массива - имена заголовков, нечетные - значения заголовков

console.log(req.rawHeaders);

// setHeader - метод записывает в ответ указаный заголовок и его значение

res.setHeader('Content-Type', 'text/plain');

//

var ct = res.getHeader('Content-Type');

console.log(ct);

// свойство определяет будет ли сервр отправлять заголовок Date. Если значение свойства false - дата не отправляется

res.sendDate = false;

//

console.log(res.headersSent);

res.end('<h1>Test page</h1>');

console.log(res.headersSent);

}).listen(8080);

## returnPage

var http = require('http');

var url = require('url');

var fs = require('fs');

http.createServer(function(req, res){

var path = url.parse(req.url).pathname;

path = '005\_returnPage/' + path.substr(1);

console.log(path);

fs.readFile(path, function(err, data){

if(err){

console.log(err);

res.writeHead(404, { 'Content-Type' : 'text/plain'});

res.end('Not Found!');

}

else{

res.writeHead(200, { 'Content-Type' : 'text/html'});

res.write(data.toString());

console.log(data.toString());

console.log('data was sent');

res.end();

}

});

}).listen(8080, function(){

console.log('Server starting!');

});

## queryParam

const http = require('http');

const url = require('url');

const server = http.createServer();

var port = 8080;

server.on('request', function(req, res) {

var method = req.method;

var \_url = req.url;

    console.log('Method:', method, '; URL:', \_url);

    var parsed = url.parse(req.url, true);

    console.log(parsed);

    if (parsed.pathname == '/test' && parsed.query.message) {

        res.statusCode = 200; // OK

        res.end(parsed.query.message);

    }

    else {

        res.statusCode = 404; // Not Found

        res.end('Page not found on server!');

    }

});

server.listen(port);

server.on('listening', function() {

    console.log('Server running on port ' + port);

})

## get

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" />

<title>Just a Page</title>

<script>

window.onload = function() {

    var btn = document.getElementById('btn');

    var container = document.getElementById('output');

        // функция для отправки GET запроса при нажатии на кнопку

    function makeRequest() {

        var xhr = new XMLHttpRequest();

        xhr.open('GET', 'request');

        xhr.onload = function() {

                container.innerHTML += this.responseText;

        };

        xhr.onerror = function() {

                console.log('error!');

        };

        xhr.send();

    };

    btn.addEventListener('click', makeRequest);

};

</script>

</head>

<body>

<button class="btn-lg btn btn-success" id="btn">Get Data!!!</button>

<div id="output"></div>

</body>

</html>

var http = require('http');

var fs = require('fs');

var url = require('url');

var path = require('path');

var port = 8080;

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

    // обработка ошибок запросов

    req.on('error', function(err) {

       console.log(err);

    });

console.log(req);

if (req.url == '/') {

// чтение файла index.html

var file\_path = path.join(\_\_dirname, '009\_index.html');

fs.readFile(file\_path, function (err, data) {

// обработка ошибок

if (err) {

console.log(err);

res.writeHead(404, { 'Content-Type': 'text/plain' });

res.write('Not Found!');

} else {

res.writeHead(200, { 'Content-Type': 'text/html' });

// записать в овет содержимое читаемого файла

res.write(data.toString());

}

res.end();

});

}

else if (req.url == '/request') {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.write('<h2>Data from server!</h2>');

res.end();

console.log('data sent!');

}

else {

res.writeHead(404, { 'Content-Type': 'text/html' });

res.end('Resource not found');

}

}).listen(port);

console.log('server running on port ' + port);

## post

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" />

<title>Just a Page</title>

<script>

window.onload = function() {

    var btn = document.getElementById('btn');

    var container = document.getElementById('output');

    function makeRequest(data) {

        var xhr = new XMLHttpRequest();

        xhr.open('POST', 'data');

        xhr.onload = function() {

var data = JSON.parse(this.responseText);

            for (var i = 0; i < data.length; i++) {

container.innerHTML += `Product: ${data[i].name} - ${data[i].price}$ <br/>`;

}

        };

        xhr.onerror = function() {

            console.log('error!!!');

        };

        xhr.send(data);

    };

    btn.addEventListener('click', function() {

var data = '<h3>This is some data!</h3>';

makeRequest(data);

});

};

</script>

</head>

<body>

<button class="btn-lg btn btn-success" id="btn">Get Data!!!</button>

<div id="output"></div>

</body>

</html>

var http = require('http');

var fs = require('fs');

var url = require('url');

var path = require('path');

var port = 8080;

var data = [

{name : 'Book', price : 10},

{name : 'Pen', price : 20},

{name : 'Lamp', price : 30},

{name : 'Pencil', price : 40},

{name : 'Desk', price : 50}

];

var server = http.createServer(function(req, res) {

// обработка ошибок запросв

req.on('error', function (err) {

console.log(err);

});

if (req.url == "/") {

// чтение файла index.html

var file\_path = path.join(\_\_dirname, '010\_index.html');

fs.readFile(file\_path, function (err, data) {

// обработка ошибок

if (err) {

console.log(err);

res.writeHead(404, { 'Content-Type': 'text/plain' });

res.write('Not Found!');

} else {

res.writeHead(200, { 'Content-Type': 'text/html' });

// записать в овет содержимое читаемого файла

res.write(data.toString());

}

res.end();

})

} else if (req.url == "/data") {

var body = '';

// получение данных POST запроса

req.on('data', function () {

body = data.toString();

// создание тела ответа

res.writeHead(200, { 'Content-Type': 'application/json' });

res.write(JSON.stringify(data));

res.end();

console.log('data sent!');

});

} else {

res.writeHead(404, { 'Content-Type': 'text/html' });

res.end('Resource not found');

}

}).listen(port);

console.log('server running on port ' + port);

## getHandle

// сервер для обработки запроса

var server = http.createServer(function (req, res) {

console.log('request')

res.end('GET request path: ' + req.url);

}).listen(port);

## postHandle

var http = require('http');

var fs = require('fs');

var url = require('url');

var port = 8080;

var server = http.createServer(function(req, res) {

console.log(res);

     var body = '';

    // обработка ошибок запросв

    req.on('error', function(err) {

       console.log(err);

    });

    // получение данных POST запроса

    req.on('data', function(data) {

       body = data.toString();

       // создание тела ответа

       res.writeHead(200, {'Content-Type': 'text/html'});

        res.write(body);

        res.end();

        console.log(`data from request: ${body}`);

    });

}).listen(port);

## request method

var http = require('http');

http.createServer(function (req, res) {

// формирование разных ответов сервера для запросов с различными HTTP методами

switch (req.method) {

case 'GET': {

var response\_text = 'GET request to path ' + req.url

console.log(response\_text);

res.end(response\_text);

break;

}

case 'POST': {

var response\_text = 'POST request to path ' + req.url

console.log(response\_text);

res.end(response\_text)

break;

}

}

}).listen(8080);