# 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

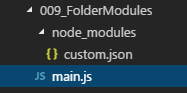
// 1. Если 'имя\_модуля/index' или 'имя\_модуля/index.js' происходит загрузка JavaScript файла

// 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]();

}