Лабораторная работа №2  
«Работа с файловой системой и потоками в Node.js»

Цель: изучить основные возможности по работе с файловой системой и потоками в Node.js

# Работа с файловой системой

Для работы с файловой системой используется File System module. Для использования этого модуля необходимо выполнить require('fs'). Все методы имеют асинхронные и синхронные формы.

Асинхронная форма всегда принимает завершающий обратный вызов в качестве последнего аргумента. Аргументы, передаваемые на завершающий обратный вызов, зависят от метода, но первый аргумент всегда зарезервирован для исключения. Если операция была завершена успешно, то первый аргумент будет null или undefined.

При использовании синхронной формы любые исключения сразу срабатывают. Поэтому необходимо использовать try/catch для обработки исключений.

Основные приемы при работе с файлами приведены ниже:

const fs = require('fs');

//create file

fs.writeFile('test.txt', 'simple text', (err) => {

    if (err)

        console.log(err)

    else

        console.log('file was created succesfully!!!');

});

//чтение файла

fs.readFile('test.txt', (err, data) => {

    if (err)

        console.log(err);

    else

        console.log(data);

});

fs.readFile('test.txt', 'utf8', (err, data) => {

    if (err)

        console.log(err);

    else

        console.log(data);

});

//rename file

fs.rename('test.txt', 'test2.txt', (err) => {

    if (err)

        console.log(err);

    else

        console.log('file was renamed!!!');

});

//append data

fs.appendFile('test2.txt', '\nnew line', (err) => {

    if (err)

        console.log(err);

    else

        console.log('file was changed!!!');

});

//delete file

fs.unlink('test2,txt', (err) => {

    if (err)

        console.log(err);

    else

        console.log('file was deleted!!!');

});

В процессе работы приложения файлы подвергаются модификации. Node.js позволяет программисту отслеживать изменения с использованием класса FSWatcher:

const fs = require('fs');

const filePath = './test2.txt';

//file statistic

fs.stat(filePath, (err, stats) => {

    if (err)

        console.log(err);

    else

        console.log(stats);

});

const watcher = fs.watch(filePath);

const appendData = (data) => {

    console.log('append - ', new Date(), `data - ${ data }`);

    fs.appendFileSync(filePath, data);

}

const stopWatch = () => {

    console.log('stop - ', new Date());

    watcher.close();

}

console.log('start - ', new Date());

appendData('text1');

watcher.on('change', (eventType, fileName) => {

    console.log('change - ', new Date(), `, Event type - ${ eventType }, File name - ${ fileName }`);

});

setTimeout(() => {

    appendData('text2')

}, 5000);

setTimeout(() => {

    appendData('text3');

    stopWatch()

}, 10000);

# Работа с директориями

Модуль FS позволяет программисту осуществлять стандартные операции не только с файлами, но и с директориями:

const fs = require('fs');

// create dir

fs.mkdir('data', (err) => {

    if (err)

        console.log(err);

    else {

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

        fs.writeFile('./data/test3.txt', 'working with directories', (err) => {

            if (err)

                console.log(err);

        });

    }

});

//delete dir

fs.rmdir('data', (err) => {

    if (err)

        console.log(err);

    else

        console.log('directoria was deleted!!!');

});

fs.unlink('./data/test3.txt', (err) => {

    if (err)

        console.log(err);

    else {

        fs.rmdir('data', (err) => {

            if (err)

                console.log(err);

            else

                console.log('directoria was deleted!!!');

        });

    }

});

//read files in dir

const files = fs.readdirSync('./');

console.log(files);

Следует обратить внимание на то, что если в директории находятся вложенные каталоги или файлы метод rmdir выдает ошибку. Следует сперва удалить все из директории, а затем только удалять ее. Можно также воспользоваться методом remove() из пакета fs-extra, который умеет удалять директории, в которых уже что-то есть.

# Работа с потоками

Потоки бывают различных типов, среди которых можно выделить потоки для чтения и потоки для записи. Для создания потока для записи применяется метод fs.createWriteStream(). Запись данных производится с помощью метода write(), в который передаются данные. Для окончания записи вызывается метод end(). Для создания потока для чтения используется метод fs.createReadStream(). Сам поток разбивается на ряд кусков или чанков (chunk). И при считывании каждого такого куска, возникает событие data. С помощью метода on() мы можем подписаться на это событие и вывести каждый кусок данных на консоль:

const fs = require('fs');

// stream for reading

const readStream = fs.createReadStream('./example.html');

// stream for writing

const writeStream = fs.createWriteStream('./example2.html');

readStream.on('data', (chunk) => {

    console.log('\nnew chank\n', chunk);

    writeStream.write(chunk);

});

Следует отметить, что при чтении файлов больших объемов потоки позволяют считать информацию, а вот стандартный метод readFile приведет к переполнению буфера.

Только работой с файлами функциональность потоков не ограничивается, также имеются сетевые потоки, потоки шифрования, архивации и т.д., но общие принципы работы с ними будут те же, что и у файловых потоков.

# Использование PIPE

Pipe - это канал, который связывает поток для чтения и поток для записи и позволяет сразу считать из потока чтения в поток записи.

Перепишем предыдущий пример с использованием pipe:

const fs = require('fs');

// stream for reading

const readStream = fs.createReadStream('./example.html');

// stream for writing

const writeStream = fs.createWriteStream('./example3.html');

readStream.pipe(writeStream);

Рассмотрим использование канала для архивации и разархивации файла. Здесь нам надо сначала считать файл, затем сжать данные и в конце записать сжатые данные в файл-архив, а затем прочитать заархивированные данные распаковать и записать в новый файл:

const zlib = require("zlib");

let readStream = fs.createReadStream("./example.html", "utf8");

let writeStream = fs.createWriteStream("./example.html.gz");

//compressed

const gzip = zlib.createGzip();

readStream.pipe(gzip).pipe(writeStream);

// uncompressd

const gunzip = zlib.createGunzip();

readStream = fs.createReadStream("./example.html.gz");

writeStream = fs.createWriteStream("./example4.txt");

readStream.pipe(gunzip).pipe(writeStream);

**Задание**

Задание 1. Написать приложение, которое логирует ошибки (вводятся пользователем) в текстовом файле. В файл добавляется дата ошибки и ее описание.

Задание 2. Создать массив объектов на основе класса в соответствии со своим вариантом. Записать этот массив в формате JSON в файл.

|  |  |  |
| --- | --- | --- |
| Вариант | Описание | Свойство |
| 01 | Процессор | Тактовая частота  Количество ядер  Разрядность процессора  Техпроцесс |
| 02 | Память | Тип памяти  Объем  Скорость чтения  Скорость записи |
| 03 | Корпус | Мощность блока питания  Тип корпуса  Форм-фактор материнской платы  Охлаждение |
| 04 | Жесткий диск | Тип накопителя  Объем  Интерфейс  Скорость вращения шпинделя |
| 05 | SSD диск | Объем  Интерфейс  Тип микросхем Flash  Контроллер |
| 06 | Оптический носитель | Тип установки  Тип  Интерфейс подключения  Максимальная скорость чтения  Максимальная скорость чтения |
| 07 | Клавиатура | Тип  Технология переключения  Назначение  Интерфейс подключения |
| 08 | Мышь | Интерфейс подключения  Назначение  Тип сенсора  Модель сенсора  Максимальное разрешение сенсора |
| 09 | Звуковая карта | Аналоговые выходы  Аналоговые входы  Микрофонные входы  Тип  Количество каналов  Интерфейс подключения |
| 10 | Видеокарта | Интерфес  Производитель графического процессора  Графический процессор  Частота графического процессора  Видеопамять (объем) |
| 11 | Блок бесперебойного питания | Тип  AVR  Мощность  Время автономного питания |
| 12 | Планшет | Диагональ экрана  Процессор  Разрешение экрана  Операционная система  Оперативная память  Внутренняя память |
| 13 | Электронная книга | Размер экрана  Тип экрана  Разрешение экрана  Флэш-память |
| 14 | Акустика | Тип  Усилитель  Максимальная мощность  Материал корпуса |
| 15 | Фитнес-браслет | Объем оперативной памяти  Поддерживаемые платформы  Датчики |
| 16 | Аккумулятор | Тип аккумулятора  Емкость  Напряжение  Полярность |
| 17 | Мобильный телефон | Тип  Операционная система  Размер экрана  Разрешение экрана  Оперативная память |
| 18 | Радиоприемник | Питание  Тип  Выходная мощность звука  Тип тюнера  Модуляция |
| 19 | Телевизор | Тип  Диагональ  Частота  Разрешение |
| 20 | Фотоаппарат | Тип камеры  Тип матрицы  Размер экрана  Количество точекматрицы  Максимальное разрешение видео |
| 21 | Видеокамера | Разрешение видеосъемки  Формат сжатия видео  Размер экрана  Количество точек матрицы |
| 22 | Принтер | Тип  Формат  Количество цветов  Технология печати  Скорость печати |
| 23 | Игровая приставка | Тип  Объем накопителя  Максимальное разрешение в играх  Оптический привод |
| 24 | IP камера | Тип матрицы  Конструкция  Стандарты беспроводной связи  Сетевой интерфейс |
| 25 | Портативная радиостанция | Стандарт  Количество каналов  Дальность радиосвязи  Количество радиостанций в комплекте |
| 26 | Модем | Тип  Интерфейс подключения  Поддержка сотовой связи  Поддержка карт памяти |
| 27 | Монитор | Диагональ  Соотношение сторон  Разрешение  Частота обновления экрана |
| 28 | Велосипед | Класс  Материал рамы  Тип трансмиссии  Диаметр колес |
| 29 | Микроволновая печь | Тип  Объем  Потребляемая мощность  Выходная мощность микроволн |
| 30 | Стиральная машина | Загрузка белья  Максимальная загрузка  Класс энергопотребления  Класс стирки  Максимальная скорость отжима |

Задание 3. Произвести считывание данных из созданного файла в новый массив. Вывести массив в консоль.

Задание 4. Создать в папке проекта две папки src и dist. В папку src скопировать исходные файлы приложений (\*.js). В папку dist скопировать созданные файлы.