

ПРОГРАММИРОВАНИЕ В INTERNET

FILE SYSTEM

Файл

=

это специальный
именованный сегмент диска
для хранения информации.

Файловая система

=

это набор спецификаций и соответствующее им программное обеспечение, которые отвечают за создание, уничтожение, организацию, чтение, запись, модификацию и перемещение файлов, а также за управление доступом к файлам и за управление ресурсами, которые используются файлами.

Популярные файловые системы:

- FAT12, FAT16, FAT32
- exFAT (или FAT64)
- NTFS
- EXT2/3/4
- APFS
- HFS+

Модуль FS

- позволяет взаимодействовать с файловой системой;
- предоставляет синхронные и асинхронные методы для работы с файловой системой;
- все операции с файловой системой имеют синхронные формы, а также асинхронные формы на основе callback и на основе promise.

```
const fs = require('fs/promises'); //promise-based
```

```
const fs = require('fs'); //synchronous, callback-based
```

Создание файла и запись в файл

```
const fs = require('fs'); // пакет для работы с файлами

// -- создание пустого файла
fs.open('./Files/File00.txt', 'w', (e, file)=>{
  if (e) throw e;
  console.log('Файл создан ');
});
```

fs.open (path, flags[, mode], callback)

flags – режим работы с файлом.

mode – режим файла (права доступа),

но только если файл был создан

В callback вторым параметром можно
получить дескриптор

```
// -- асинхронная перезапись/создание и запись в файл
let str01 = 'Строка 01\nСтрока 02\nСтрока 03\n';
fs.writeFile('./Files/File01.txt', str01, (e)=>{
  if (e) throw e;
  console.log('Запись в файл выполнена успешно');
});

// -- асинхронная запись в хвост/создание и запись в файл
let str01x = 'Строка 04\nСтрока 05\nСтрока 06\n';
fs.appendFile('./Files/File01.txt', str01x, (e)=>{
  if (e) throw e;
  console.log('Запись в файл выполнена успешно');
});
```

fs.writeFile(file, data[, options], callback)

fs.appendFile(file, data [, options], callback)

Чтение из файла (асинхронное)

```
const fs = require('fs'); // пакет для работы с файлами

fs.readFile('./Files/File01.txt', (e, data)=>{
  if (e) console.log('Ошибка: ', e);
  else console.log('data: ', data.toString('utf8'));
});
```

fs.readFile(file [, options], callback)

В callback вторым параметром можно получить считанные данные

Удаление файла

```
fs.unlink('./Files/File04.txt', (e)=>{  
  if (e) console.log('Ошибка: ', e);  
  else console.log('Файл удален');  
});
```

fs.unlink(path, callback)

Переименование файла

```
const fs = require('fs');

fs.rename('./Files/File04.txt', './Files/File04x.txt', (e) => {
  if (e) console.log('Ошибка: ', e);
  else console.log('Файл переименован');
});
```

fs.rename(oldPath, newPath, callback)

В случае, если newPath уже существует, он будет перезаписан. Если в newPath указан каталог, будет выдана ошибка.

Копирование файла

```
// новый файл будет создан или перекроет существующий
fs.copyFile('./Files/File04x.txt', './Files/File04.txt', (e) => {
  if(e) console.log('Ошибка: ', e);
  else console.log('Файл скопирован');
})
```

fs.copyFile(src, dest[, mode], callback)

По умолчанию, dest перезаписывается, если он уже существует.

Проверка наличия файла

```
const fs = require('fs');

fs.access('./Files/File04.txt', fs.constants.F_OK, (err) => {
  if (err) console.log('Файл не существует');
  else console.log('Файл существует');
});
```

fs.access(path[, mode], callback)

Проверяет права пользователя для файла или каталога, указанного в path. Аргумент mode – необязательное целое число, указывающее, какие проверки доступности необходимо выполнить.

Работа с директориями (создание, переименование)

```
const fs = require('fs');

// создание директории
fs.mkdir('./Files/NewDir', { recursive: true }, (err) => {
  if (err) console.log(err);
  else console.log('Директория создана');
});

// переименование директории
fs.rename('./Files/NewDir', './Files/MyDir', (err) => {
  if (err) console.log(err);
  else console.log('Директория переименована');
});
```

fs.mkdir(path[, options], callback)

Необязательный аргумент options может содержать свойство recursive указывающее, следует ли создавать родительские каталоги.

fs.rename(oldPath, newPath, callback)

В случае, если newPath уже существует, она будет перезаписана.

Работа с директориями (удаление, чтение содержимого)

```
// удаление директории
fs.rmdir('./Files/MyDir', { recursive: true }, (err) => {
  if (err) console.log(err);
  else console.log('Директория удалена');
});
```

fs.rmdir(path[, options], callback)

Аргумент options может содержать свойство recursive. В рекурсивном режиме операции повторяются при сбое.

Можно также указать maxRetries и retryDelay, если recursive = true.

Работа с директориями (чтение содержимого)

```
// чтение содержимого директории
fs.readdir('./Files', { withFileTypes: true }, (err, files) => {
  if (err) console.log(err);
  else console.log('Содержимое директории: ', files);
});
```

fs.readdir(path[, options], callback)

Вторым параметром можно получить массив имен файлов в каталоге.

Необязательный аргумент options может быть строкой или объектом, определяющим кодировку, используемую для имен файлов.

```
PS D:\NodeJS\samples\сwp_09> node 09-02
Содержимое директории: [
  Dirent { name: 'File04.txt', [Symbol(type)]: 1 },
  Dirent { name: 'File21.txt', [Symbol(type)]: 1 },
  Dirent { name: 'inFile.txt', [Symbol(type)]: 1 },
  Dirent { name: 'MyDir', [Symbol(type)]: 2 },
  Dirent { name: 'outFile.txt', [Symbol(type)]: 1 },
  Dirent { name: 'тест.txt', [Symbol(type)]: 1 }
]
PS D:\NodeJS\samples\сwp_09> node 09-02
Содержимое директории: [
  'File04.txt',
  'File21.txt',
  'inFile.txt',
  'MyDir',
  'outFile.txt',
  'тест.txt'
]
```

без withFileTypes

Если для параметра **options.withFileTypes** установлено значение true, массив файлов будет содержать объекты fs.Dirent

Служка за файлом/директорией

```
const fs = require('fs');

const filename='./Files/File01.txt';

try{
  fs.watch(filename, (event, f) => {           // следить за файлом
    if (f) console.log(`file: ${f}, event = ${event}`);
  });
}
catch(e){
  console.log('catch e = ', e.code);
}
```

fs.watch (filename[, options], listener)

В обработчик будет приходить первым аргументом название события: изменен этот файл или переименован (создан, удален). Таким же образом мы можем следить не только за файлами, но и за директориями.

Аргумент options можно опустить. Объект options может содержать логическое значение со свойством *recursive* (отслеживать ли все подкаталоги, для каталогов), *encoding* (кодировка символов для имени файла, передаваемого слушателю)

```
const fs = require('fs');
const dirname='./Files';

try{
  fs.watch(dirname, (event, f) => {           // следить папкой
    if (f) console.log(`folder: ${f}, event = ${event}`);
  });
}
catch(e){
  console.log('catch e = ', e.code);
}
```

Поток данных =
Stream

это абстракция над данными,
используемая для
чтения/записи файлов, сокетов
и др.

Виды потоков в Node.js

- **Readable** – поток, который предоставляет данные на чтение.

Примеры: request на сервере, response на клиенте, поток чтения fs, process.stdin

- **Writable** – поток, в который данные можно записывать.

Примеры: response на сервере, request на клиенте, поток записи fs, process.stdout/stderr

- **Duplex** – поток, из которого можно как читать данные (Readable), так и записывать в него (Writable), при этом процесс чтения и записи происходит независимо друг от друга.

Примеры: socket, websocket stream

- **Transform** – разновидность Duplex потоков, которые могут изменять данные при их записи и чтении в/из потока.

Примеры: zlib, crypto

Режимы работы Readable потока

1. Автоматический (.pipe(), события data и end)

```
const fs = require('fs');

let i = 0;
fs.createReadStream('./Files/File14.txt')
  .on('data', (chunk) => {
    console.log(`--${i++}-- ${chunk.length}`);
  })
  .on('end', () => {
    console.log('--end--');
  })
  .on('error', (err) => {
    console.log(`error: ${err}`);
  })
```

```
PS D:\NodeJS\samples\cwp_09> node .\09-10.js
--0-- 65536
--1-- 488
--end--
```

```
const fs = require('fs');
let readStream = fs.createReadStream('./Files/File14.txt');
let writeStream = fs.createWriteStream('./Files/File22.txt');
readStream.pipe(writeStream);
```

Режимы работы Readable потока

2. Пошаговый или приостановленный (событие readable)

```
const fs = require('fs');

let rstream = fs.createReadStream('./Files/File14.txt'); // поток для чтения

rstream.on('readable', ()=>{                                // можно читать

    let chunk = null;
    while ((chunk = rstream.read()) != null) {
        console.log(chunk.toString());
    }
})

rstream.on('error', (e)=>{                                    // обработка ошибок
    console.log('error = ', e);
})
```

```
rstream.on('readable', ()=>{

    let chunk = null;
    while ((chunk = rstream.read(5)) != null) {
        console.log(chunk.toString());
    }
})
```

Writable поток

1.

```
const fs = require('fs');  
  
let ws = fs.createWriteStream('./Files/File21.txt');  
  
for (let i = 0; i < 10; i++) {  
  ws.write(`---${i}---`);  
}  
  
ws.end(); // сигнализирует, что данных больше не будет, закрытие потока  
  
// ws.write(`last chunk`); // ERR_STREAM_WRITE_AFTER_END: запись больше невозможна
```

```
Files > ≡ File21.txt  
1  ---0-----1-----2-----3-----4-----5-----6-----7-----8-----9---
```

2.

```
const fs = require('fs');  
let readStream = fs.createReadStream('./Files/File14.txt');  
let writeStream = fs.createWriteStream('./Files/File22.txt');  
readStream.pipe(writeStream);
```

Duplex поток (Readable + Writable)

```
const net = require('net');

const server = net.createServer(socket => {
  socket.on('data', data => {
    socket.write('ECHO: ' + data.toString());
  });

  socket.on('error', error => {
    console.log(error);
  });
});

server.listen(3000, () => console.log('Server is running'));
```

```
PS D:\NodeJS\samples\cwp_09> node 09-11s
Server is running
□
```

```
const net = require('net');

const client = net.createConnection({ port: 3000, host: '127.0.0.1' }, () => {
  client.write('Hello, server!');
});

client.on('data', data => {
  console.log(`Received data: ${data.toString()}`);
});

client.on('error', error => {
  console.log(error);
});
```

```
PS D:\NodeJS\samples\cwp_09> node 09-11c
Received data: ECHO: Hello, server!
□
```

Transform поток

```
const fs = require('fs');
const zlib = require('zlib');

const input = fs.createReadStream('./Files/input.txt');

const gzip = zlib.createGzip(); // transform stream

const output = fs.createWriteStream('./Files/input.txt.gz');
input.pipe(gzip).pipe(output);
```