fs 模块

fs 全称为 `file system`，称之为 `文件系统`，是 Node.js 中的 `内置模块`，可以对计算机中的磁盘进行操作。

本章节会介绍如下几个操作：

1. 文件写入
2. 文件读取
3. 文件移动与重命名
4. 文件删除
5. 文件夹操作
6. 查看资源状态

一、文件写入

文件写入就是将 `数据` 保存到 `文件` 中，我们可以使用如下几个方法来实现该效果

方法	说明
writeFile	异步写入
writeFileSync	同步写入
appendFile / appendFileSync	追加写入
createWriteStream	流式写入

1-1. writeFile 异步写入

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

参数说明：

- file 文件名
- data 待写入的数据
- options 选项设置 `（可选）`
- callback 写入回调

返回值：`undefined`

代码示例：

```
// require 是 Node.js 环境中的'全局'变量，用来导入模块
const fs = require('fs');

//将 『三人行，必有我师焉。』 写入到当前文件夹下的『座右铭.txt』文件中
fs.writeFile('./座右铭.txt', '三人行，必有我师焉。', err => {
  //如果写入失败，则回调函数调用时，会传入错误对象，如写入成功，会传入 null
  if(err){
    console.log(err);
    return;
  }
  console.log('写入成功');
});
```

1-2. writeFileSync 同步写入

语法: `fs.writeFileSync(file, data[, options])`

参数与 fs.writeFile 大体一致，只是没有 callback 参数

返回值: `undefined`

代码示例:

```
try{
  fs.writeFileSync('./座右铭.txt', '三人行，必有我师焉。');
}catch(e){
  console.log(e);
}
```

> Node.js 中的磁盘操作是由其他 `线程` 完成的，结果的处理有两种模式:
>
> - 同步处理 JavaScript 主线程 `会等待` 其他线程的执行结果，然后再继续执行主线程的代码，`效率较低`
> - 异步处理 JavaScript 主线程 `不会等待` 其他线程的执行结果，直接执行后续的主线程代码，`效率较好`

1-3. appendFile / appendFileSync 追加写入

appendFile 作用是在文件尾部追加内容，appendFile 语法与 writeFile 语法完全相同

语法:

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

`fs.appendFileSync(file, data[, options])`

返回值: 二者都为 `undefined`

实例代码:

```
fs.appendFile('./座右铭.txt','择其善者而从之，其不善者而改之。', err => {
  if(err) throw err;
  console.log('追加成功')
});

fs.appendFileSync('./座右铭.txt','\r\n温故而知新，可以为师矣');
```

1-4. createWriteStream 流式写入

语法： `fs.createWriteStream(path[, options])`

参数说明：

- path 文件路径
- options 选项配置（`可选`）

返回值： `Object`

代码示例：

```
let ws = fs.createWriteStream('./观书有感.txt');

ws.write('半亩方塘一鉴开\r\n');
ws.write('天光云影共徘徊\r\n');
ws.write('问渠那得清如许\r\n');
ws.write('为有源头活水来\r\n');

ws.end();
```

> `程序打开一个文件是需要消耗资源的`，流式写入可以减少打开关闭文件的次数。
>
> 流式写入方式适用于 `大文件写入或者频繁写入` 的场景, writeFile 适合于 `写入频率较低的场景`

1-5 写入文件的场景

`文件写入` 在计算机中是一个非常常见的操作，下面的场景都用到了文件写入

- 下载文件
- 安装软件
- 保存程序日志，如 Git
- 编辑器保存文件
- 视频录制

> 当 `需要持久化保存数据` 的时候，应该想到 `文件写入`

二、文件读取

文件读取顾名思义，就是通过程序从文件中取出其中的数据，我们可以使用如下几种方式：

方法	说明
readFile	异步读取
readFileSync	同步读取
createReadStream	流式读取

2-1 readFile 异步读取

语法：`fs.readFile(path[, options], callback)`

参数说明：

- path 文件路径
- options 选项配置
- callback 回调函数

返回值：`undefined`

代码示例：

```
//导入 fs 模块
const fs = require('fs');

fs.readFile('./座右铭.txt', (err, data) => {
  if(err) throw err;
  console.log(data);
});

fs.readFile('./座右铭.txt', 'utf-8',(err, data) => {
  if(err) throw err;
  console.log(data);
});
```

2-2 readFileSync 同步读取

语法：`fs.readFileSync(path[, options])`

参数说明：

- path 文件路径
- options 选项配置

返回值：`string | Buffer`

代码示例：

```
let data = fs.readFileSync('./座右铭.txt');
let data2 = fs.readFileSync('./座右铭.txt', 'utf-8');
```

2-3 createReadStream 流式读取

语法：`fs.createReadStream(path[, options])`

参数说明：

- path 文件路径

- options 选项配置（`可选`）

返回值：`Object`

代码示例：

```
//创建读取流对象
let rs = fs.createReadStream('./观书有感.txt');
//每次取出 64k 数据后执行一次 data 回调
rs.on('data', data => {
  console.log(data);
  console.log(data.length);
});
//读取完毕后, 执行 end 回调
rs.on('end', () => {
  console.log('读取完成')
})
```

2-4 读取文件应用场景

- 电脑开机
- 程序运行
- 编辑器打开文件
- 查看图片
- 播放视频
- 播放音乐
- Git 查看日志
- 上传文件
- 查看聊天记录

三、文件移动与重命名

在 Node.js 中，我们可以使用 `rename` 或 `renameSync` 来移动或重命名`文件或文件夹`

语法：

`fs.rename(oldPath, newPath, callback)`

`fs.renameSync(oldPath, newPath)`

参数说明：

- oldPath 文件当前的路径
- newPath 文件新的路径
- callback 操作后的回调

代码示例：

```
fs.rename('./观书有感.txt', './论语/观书有感.txt', (err) =>{
  if(err) throw err;
  console.log('移动完成')
});

fs.renameSync('./座右铭.txt', './论语/我的座右铭.txt');
```

四、文件删除

在 Node.js 中，我们可以使用 `unlink` 或 `unlinkSync` 来删除文件

语法：

`fs.unlink(path, callback)`

`fs.unlinkSync(path)`

参数说明：

- path　文件路径
- callback 操作后的回调

代码示例：

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

fs.unlink('./test.txt', err => {
  if(err) throw err;
  console.log('删除成功');
});

fs.unlinkSync('./test2.txt');
```

五、文件夹操作

借助 Node.js 的能力，我们可以对文件夹进行 `创建` 、 `读取` 、 `删除` 等操作

方法	说明
mkdir / mkdirSync	创建文件夹
readdir / readdirSync	读取文件夹
rmdir / rmdirSync	删除文件夹

5-1 mkdir 创建文件夹

在 Node.js 中，我们可以使用 `mkdir` 或 `mkdirSync` 来创建文件夹

语法：

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

`fs.mkdirSync(path[, options])`

参数说明：

- path　文件夹路径
- options 选项配置（ `可选` ）
- callback 操作后的回调

示例代码：

```
//异步创建文件夹
```

```
fs.mkdir('./page', err => {
  if(err) throw err;
  console.log('创建成功');
});

//递归异步创建
fs.mkdir('./1/2/3', {recursive: true}, err => {
  if(err) throw err;
  console.log('递归创建成功');
});

//递归同步创建文件夹
fs.mkdirSync('./x/y/z', {recursive: true});
```

5-2 readdir 读取文件夹

在 Node.js 中，我们可以使用 `readdir` 或 `readdirSync` 来读取文件夹

语法：

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

`fs.readdirSync(path[, options])`

参数说明：

- path　文件夹路径
- options 选项配置（`可选`）
- callback 操作后的回调

示例代码：

```
//异步读取
fs.readdir('./论语', (err, data) => {
  if(err) throw err;
  console.log(data);
});
//同步读取
let data = fs.readdirSync('./论语');
console.log(data);
```

5-3 rmdir 删除文件夹

在 Node.js 中，我们可以使用 `rmdir` 或 `rmdirSync` 来删除文件夹

语法：

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

`fs.rmdirSync(path[, options])`

参数说明：

- path　文件夹路径
- options 选项配置（`可选`）
- callback 操作后的回调

示例代码：

```

```

```
//异步删除文件夹
fs.rmdir('./page', err => {
  if(err) throw err;
  console.log('删除成功');
});
//异步递归删除文件夹
fs.rmdir('./1', {recursive: true}, err => {
  if(err) {
    console.log(err);
  }
  console.log('递归删除')
});
//同步递归删除文件夹
fs.rmdirSync('./x', {recursive: true})
```

六、查看资源状态

在 Node.js 中，我们可以使用 `stat` 或 `statSync` 来查看资源的详细信息

语法：

```
fs.stat(path[, options], callback)
```

```
fs.statSync(path[, options])
```

参数说明：

- path　文件夹路径
- options 选项配置（`可选`）
- callback 操作后的回调

示例代码：

```
//异步获取状态
fs.stat('./data.txt', (err, data) => {
  if(err) throw err;
  console.log(data);
});
//同步获取状态
let data = fs.statSync('./data.txt');
```

结果值对象结构：

- size　　文件体积
- birthtime　创建时间
- mtime　　最后修改时间
- isFile　　检测是否为文件
- isDirectory 检测是否为文件夹
- ....

七、相对路径问题

fs 模块对资源进行操作时，路径的写法有两种：

- 相对路径
  - `./座右铭.txt` 当前目录下的座右铭.txt
  - `座右铭.txt` 等效于上面的写法
  - `../座右铭.txt` 当前目录的上一级目录中的座右铭.txt
- 绝对路径
  - `D:/Program Files` windows 系统下的绝对路径
  - `/usr/bin` Linux 系统下的绝对路径

> 相对路径中所谓的 `当前目录`，指的是 `命令行的工作目录`，而并非是文件的所在目录
>
> 所以当命令行的工作目录与文件所在目录不一致时，会出现一些 BUG

八、__dirname

`__dirname` 与 `require` 类似，都是 Node.js 环境中的'全局'变量

`__dirname` 保存着 `当前文件所在目录的绝对路径`，可以使用 __dirname 与文件名拼接成绝对路径

代码示例：

```
let data = fs.readFileSync(__dirname + '/data.txt');
console.log(data);
```

> 使用 fs 模块的时候，尽量使用 __dirname 将路径转化为绝对路径，这样可以避免相对路径产生的 Bug

九、练习

1. 编写一个 JS 文件，实现复制文件的功能
2. 文件重命名