一、 文件系统fs

fs 全称为file system,是NodeJS中的内置模块，可以对计算机中的文件进行增删改查等操作。

1. 使用fs写入文件

1. 普通（fs.writeFile()）写入

1. fs.writeFile（）语法介绍

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

`file`:要将文件写到哪儿去（可以是相对路径，也可以是绝对路径）
`data`:写入的内容
`[,options]`:可选参数；配置参数 {flag:'a'}
`callback`:回调函数

2. 具体流程

1. 引入 fs模块

```
const fs = require('fs'); //固定写法，建议同一写成这句话
```

2. 调用fs中的方法完成写入
   fs 的路径是参照于执行命令的，所以为了在我们期望的位置实现效果，通常建议将路径设置为绝对路径。
   `__dirname` 是一个特殊的变量，保存的值时当前文件所在目录的绝对路径，不受到执行命令位置的影响
   绝对路径版本

```
fs.writeFile(__dirname + '/index.html','面子都是互相给的','function(err){
    if(err){
        console.log('写入失败')
        return
    }else{
        console.log('写入失败')
    }
}')
```

相对路径版本

```
fs.writeFile('./index.html','面子都是互相给的','function(err){
    if(err){
        console.log('写入失败')
        return
    }else{
        console.log('写入失败')
    }
}')
```

3. 写入的同步api

```
fs.writeFileSync('路径', '内容');//同步api的执行效率低，不用设置回调
```

写测试demo，本地的一些文件操作，用同步api
做服务功能的时候，用异步api

2. 用写入流（fs.createWriteStream()）写入文件

写入流程

1. 引入fs模块

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

2. 创建写入流对象

```
const ws = fs.createWriteStream('./app.js')
```

3. 写入文件

```
ws.write('var a = 100')
ws.write('我是小左')
```

4. 在写入时可以绑定事件 当写入对象创建成功时触发

```
ws.on('open',() = >{
    console.log('写入流创建成功')
})
ws.on('close',() = >{
    console.log('写入流关闭')
})
```

5. 关闭写入流

```
ws.close();
```

写入文件的两种方式的选择

- writeFile 简单的文件写入，写入次数较少
- createWriteStream 多次内容写入，用这个

在windows下要换行时，使用\r\n进行换行，不能使用br标签

2. 使用fs读取文件

1. 普通（fs.readFile()）读取

1. fs.readFile()语法

```
fs.readFile(file, callback)
```

`file`:要读取的文件的路径，可以是相对也可以是绝对路径
`callback`： 回调函数，一般有两个参数
- err:错误提示
- data:读取的数据

2. 具体流程

引入fs模块

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

调用read读取file文件(相对路径)

```
fs.readFile('.index.html',(err,data) = >{
    if (err) throw err;//如果出现错误，就弹出具体错误
    console.log(data.toString());//将目标数据以字符串的方式输出

})
```

绝对路径版本

```
const fs = require('fs');
fs.readFile('d:/index.html',(err, data)=>{
    if(err) throw err;
    console.log(data.toString());
})
```

普通读取的同步API

```
fs.readFileSync(file)
```

2. 使用读取流（fs.createReadStream（））读取文件

读取流程

引入fs对象

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

创建读取对象流对象

```
const rs = fs.createReadStream('文件路径');
```

绑定事件
data事件可以对文件进行分块处理，提高处理效率

```
rs.on('data',chunk = >{
    console.log(chunk.toString())

})

rs.on('open',() =>{
    console.log('读取流打开啦')
})

rs.on('end',() =>{
    cosnole.log('读取流结束啦')
})
```

关于文件读取两种方式的选择

readFile 小文件读取
creatReadStream 大文件读取 用这种方式读取文件时，占用的内存更小

流式文件复制的练习

```
const fs = require('fs');
//创建读取流
const rs = fs.createReadStream('./file/刻意练习.mp3');
//创建写入流对象
const ws = fs.createWriteStream('../../制胜法宝.mp3');
//绑定事件
rs.on('data' chunk =>{
    //将数据写入到目标文件中
    ws.write(chunk);
});
//简便操作
rs.pipe(ws);
```

3. 文件删除

fs.unlink()语法

```
fs.unlink(file,callback)
```

`file`:要删除的文件
`callback`： 回调函数，一般只有一个err参数

删除文件练习

```
const fs = require('fs');
fs.unlink('../../制胜法宝.mp3',err =>{
    if(err) throw err;
    console.log('delete success');
})
```

同步api

```
fs.unlinkSync(file)
```

4. 移动/重命名文件

fs.rename()语法

```
fs.rename(file,targetfile,callback)
```

`file`:要移动或者重命名的文件
`targetfile`: 当这个参数和第一个参数处于同一文件夹下时，为重命名；处于不同文件夹下时，为移动；当然也可以是移动并重命名。
`callback`： 回调函数，一般只有一个err参数

```
const fs = require('fs');
fs.rename('./index.html','./project/new_index.html', err =>{
    if(err) throw err;
    console.log('move and rename success');
})
```

同步api

```
fs.renameSync('1.log','2.log')
```

5. 文件夹操作

fs.readdir(要读取的文件夹路径，回调（err,data））

fs.rmdir(要删除的文件夹路径，回调（err））

- 可以设置`{recursive:true}`进行递归创建

fs.mkdir(要创建的文件夹路径，回调（err））

- 可以设置`{recursive:true}`进行递归创建

```
// 引入fs模块
const fs = require('fs')
// 读取文件夹
fs.readdir('文件夹路径',(err,data)=>{
    if(err) throw err;
    cosole.log(data);//data 为当前文件下的内容
})

// 删除文件夹
fs.rmdir('要删除的文件夹',{recursive:true},err=>{
    if(err) throw err;
    cosole.log('删除成功');
})
// {recursive:true}：递归删除


// 创建文件夹
fs.mkdir('文件夹路径',err=>{
    if(err) throw err;
    cosole.log('创建成功');
})
```

```
fs.mkdir('文件夹路径',{recursive:true},err=>{
    if(err) throw err;
    cosole.log('创建成功');
})

// {recursive:true}：递归创建
```

6. fs.stat

查看目标文件的状态，返回的结果是一个对象。包含了文件的大小、类型、相关时间等内容。

```
const fs = require('fs');
fs.stat('文件路径',（err,stats）=>{
    if(err) throw err;
    console.log(stats)
    if(stats.isFile()){
        console.log('是一个文件')
    }
    if(stats.isDirectory()){
        console.log('是一个文件夹')
    }
})
```

unicode 字符集

- https://www.tamasoft.co.jp/en/general-info/unicode.html
- https://www.cnblogs.com/whiteyun/archive/2010/07/06/1772218.html

读取和写入的应用场景

写入文件场景

1. 下载文件
2. 安装文件
3. 日志(程序日记) 如 Git
4. 数据库
5. 网盘
6. 编辑器保存文件
7. 视频录制

读取文件场景

1. 下载文件
2. 程序运行
3. 数据读取(数据库)
4. 日志 (git log)
5. 编辑器打开文件

7. 操作系统的路径

1. 绝对路径
    - E:img/xiaoyuan/图书馆.jpg
    - /user/xxx/xxx　Linux的根目录为/
    - 优点：总能找到对应的文件。
2. 相对路径

- ./img/xiaoyuan/图书馆.jpg
- ../../../img/xiaoyuan/图书馆.jpg

3. 路径分割符

- windows \
- linux /
- 在编程时，推荐/

fs综合练习

通过 fs 模块在D盘下创建下列文件结构：

```
- project
  - images
    - logo.png
  - css
    - app.css
  - js
    - app.js
  -index.html
```

用同步api创建

```
const fs = require('fs');
//同步API
fs.mkdirSync('D:/Project');
fs.mkdirSync('D:/Project/images');
fs.mkdirSync('D:/Project/css');
fs.mkdirSync('D:/Project/js');

fs.writeSync('d:/project/index.html');
fs.writeSync('d:/project/iamges/logo.png');
fs.writeSync('d:/project/css/app.css');
fs.writeSync('d:/project/js/app.js');
```

异步创建 实现回调地狱 (问题：错误处理不方便 阅读性差 -> promise解决)

```
const fs = require('fs');
fs.mkdir('D:/project', err=>{
    if(err && err.code === 'EEXIST'){
        fs.rmdir('D:/project', {recursive:true}, err =>{
            if(err){
                console.log('failed');
                return;
            }else{
                fs.mkdir('D:/project', err => {
                    if(err) throw err;
                    //创建project内部结构
                    createStruct();
                })
            }
        })
    }else{
        //调用函数 创建 project 内部结构
        createStruct();
    }
```

```
});
function createStruct(){
    //显示成功
    fs.mkdir('D:/project/images', err => {
        if(err) throw err;
        //创建 logo.png
        fs.writeFile('D:/project/images/logo.png', '', err=>{
            if(err) throw err;
            console.log('logo.png 创建成功')
        })
    });

    fs.mkdir('D:/project/css', err => {
        if(err) throw err;
        //创建 logo.png
        fs.writeFile('D:/project/css/app.css', '', err=>{
            if(err) throw err;
            console.log('app.css 创建成功')
        })
    });

    fs.mkdir('D:/project/js', err => {
        if(err) throw err;
        //创建 logo.png
        fs.writeFile('D:/project/js/app.js', '', err=>{
            if(err) throw err;
            console.log('app.js 创建成功')
        })
    });

    fs.writeFileSync('D:/project/index.html', '', err => {
        if(err) throw err;
        console.log('index.html 创建成功');
    })
}
```