1. **Node.js的作用：**
2. 开发服务器应用
3. 开发工具类应用（Webpack,Vite,Babel）
4. 开发桌面端应用（VSCode,Figma,Postman）==》借助于electron框架==》electron是借助于nodejs开发的
5. **Node.js的下载和安装：官网：nodejs.org/nodejs.cn**

检测安装成功：cmd：node -v

1. **CMD常用命令**

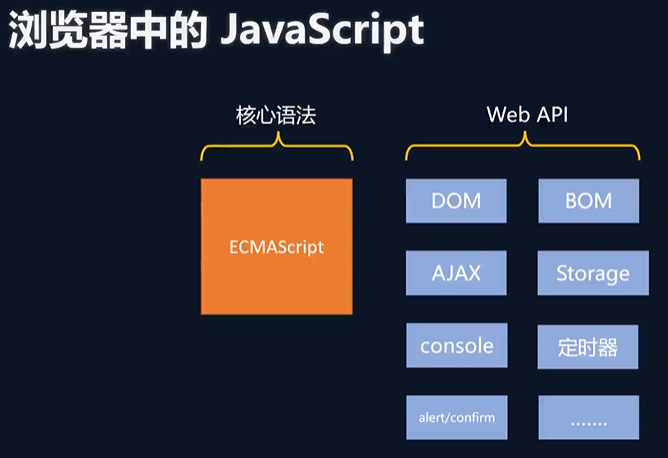
切换盘符： C: D:

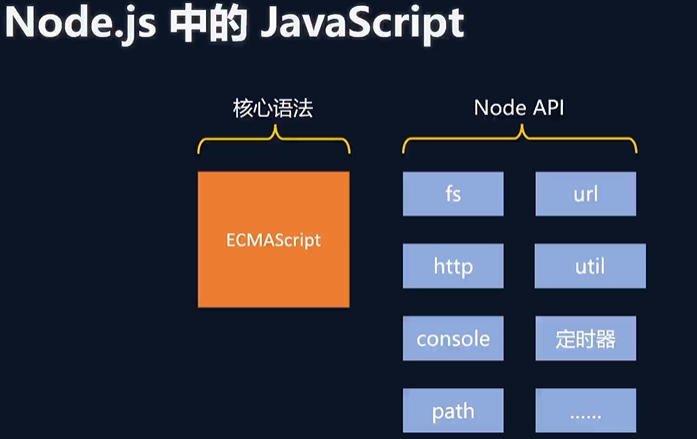
切换工作目录：cd 指定目录

查看目录文件： dir

1. **Node.js注意点：**

不能使用BOM和DOM还有ajax相关的API，顶级对象为global，也可以使用globalThis来访问顶级对象





1. **Buffer：**

Buffer中文译为缓冲区，是一个类似于Array的对象，用于表示固定长度的字节序列，换句话说，Buffer就是一段固定长度的内存空间，用于处理二进制数据

**特点：**

1. 大小固定且无法调整
2. Buffer性能较好，可以直接对计算机内存进行操作
3. 每个元素的大小为1字节（byte)

**创建：**

// 1.alloc

let buf =Buffer.alloc(10)

console.log(buf);

// 2.allocUnsafe

let buf2 =Buffer.allocUnsafe(10)

console.log(buf2);

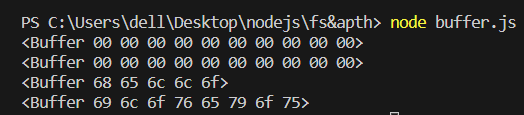
// 3.from

let buf3 =Buffer.from('hello')

console.log(buf3);

let buf4 =Buffer.from([105,108,111,118,101,121,111,117])

console.log(buf4);



用alloc创建的buffer每一个二进制位都会归零

用allocUnsafe创建的buffer会包含旧的内存数据

用from会将字符转为unicode码表中的十六进制

**操作：**

let buf4 =Buffer.from([105,108,111,118,101,121,111,117])

console.log(buf4.toString());

// []

let buf = Buffer.from('hello')

console.log(buf[0].toString(2));//转成二进制

buf[0]=95

console.log(buf.toString());

// 溢出

let buf = Buffer.from('hello')

buf[0]=361;//二进制最多到255 舍弃高位的数字 361转成二进制0001 0110 1001 =》0110 1001=》69

console.log(buf);

// 中文

let buf = Buffer.from('你好')

console.log(buf);

1. **进程与线程**

进程：进行中的程序/程序的一次执行过程

线程：线程是一个进程中执行的一个执行流，一个线程是属于某个进程的



1. **fs模块 (文件系统模块）**

Fs模块可以实现与硬盘的交互，例如文件的创建，删除，重命名，移动，还有文件内容的写入和读取以及文件夹的相关操作

1. **文件写入**

* **异步写入**

//新建一个文件，座右铭.txt，写入内容，三人行，则必有我师焉

//1.导入fs模块

const fs = require('fs');

//2.写入文件:文件名，待写入的数据，选项配置（可选），回调

fs.writeFile('./座右铭.txt','三人行，则必有我师焉',err=>{

    //写入失败err是错误对象，写入成功err是null

    if(err){

        console.log('写入失败');

        return;

    }

    console.log('写入成功');

})

* **同步写入**

//同步写入

fs.writeFileSync('./data.txt','11111111111')

* **追加写入**

appendFile/appedFileSync

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

语法： appendFile(file,data{,optons},callack)；appendFileSync(file,data{,options})

返回值都为undefined

‘\r\n’表示换行

fs.appendFile('./座右铭.txt','则其善者而从之',err=>{

    if(err) throw err;

    console.log('追加成功')

})

fs.appendFileSync('./座右铭.txt','\r\n温故而知新')

writeFile也可以实现追加写入

fs.writeFile('./座右铭.txt','白毛浮绿水',{flag:'a'},err =>{

    if(err){

        console.log('写入失败');

        return;

    }

    console.log('写入成功')

})

* **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

//1.导入fs

const fs = require('fs');

//2.创建写入流对象

const ws = fs.createWriteStream('./观书有感.txt')

//3.write

ws.write('111\r\n')

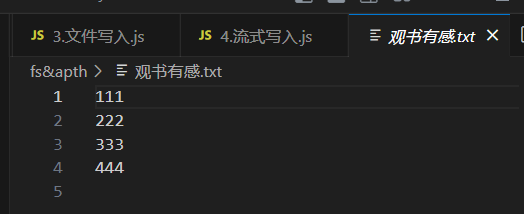
ws.write('222\r\n')

ws.write('333\r\n')

ws.write('444\r\n')

//4.关闭通道

ws.close()



**程序打开一个文件是需要小号资源的，流式写入可以减少打开关闭文件的次数。**

**流式写入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**

* **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

1. 下载文件
2. 安装文件
3. 保存程序日志，如Git
4. 编辑器保存文件
5. 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**

1. **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：

readFile异步读取

readFleSync同步读取

createReadStream流式读取

* **异步读取**

// 1.引入fs模块

const fs = require('fs')

//2.异步读取，文件路径，配置对象（可选），回调函数(err错误信息，data读取到的文件内容(buffer))

fs.readFile('./观书有感.txt', (err, data) => {

    if (err) {

        console.log('读取失败');

        return

    }

    console.log(data.toString());

})

* **同步读取**

let data = fs.readFileSync('./观书有感.txt')

console.log(data.toString());

* **读取文件的应用场景**

1. 电脑开机
2. 程序运行
3. 编辑器打开文件
4. 查看图片
5. 播放视频
6. 播放音乐
7. Git查看日志
8. 上传文件
9. 查看聊天记录

* **流式读取**

//1.导入fs

const fs = require('fs');

//2.创建读取流对象

const rs = fs.createReadStream('./观书有感.txt')

//3.绑定data事件 chunk=>块

rs.on('data',chunk=>{

    console.log(chunk);

})

//end可选事件

rs.on('end',()=>{

    console.log('done');

})

**每次读取最多65536字节=》64kb**

**处理大文件时候可以提高效率**

1. **文件复制**

//复制座右铭文件

//导入fs

const fs = require('fs');

//方式一 readFile

//读取文件内容

let data = fs.readFileSync('./座右铭.txt');

//写入文件

fs.writeFileSync('./座右铭copy.txt', data)

console.log(process.memoryUsage()) //rss

//方式二 流式操作

//创建读取流对象

const rs = fs.createReadStream('./座右铭.txt')

//创建写入流对象

const ws = fs.createWriteStream('./座右铭2.txt')

//为读取流绑定事件

rs.on('data', chunk =>{

    ws.write(chunk)

})

rs.on('end',()=>{

    console.log(process.memoryUsage())

})

**文件大的情况下流式操作需要的内存空间更小**

//方式三

rs.pipe(ws)//读取流传递给写入流，使用频率低

1. **文件的移动与重命名**

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：文件当前路径

newPath：文件新路径

callback：操作后的回调

* **重命名**

//1.导入fs

const fs = require('fs');

//调用rename方法，旧路径，新路径

fs.rename('./座右铭copy.txt','./座右铭111.txt',err=>{

if(err){

    console.log('操作失败')

    return

}

console.log('成功');

})

* **移动**

fs.rename('./data.txt','../资料/座右铭111.txt',err=>{

    if(err){

        console.log('操作失败')

        return

    }

    console.log('成功');

    })

1. **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)

fs.rm(path, callback)

fs.rmSync(path)

* **参数说明：**

path：文件路径

callback：操作后的回调

//1.导入fs

const fs = require('fs');

//调用unlink

fs.unlink('./座右铭111 copy.txt',err=>{

    if(err){

        console.log('回调失败');

        return

    }

    console.log('成功');

})

//调用rm方法 Node.js 14.4引入的新方法

fs.rm('座右铭111 copy.txt',err=>{

    if(err){

        console.log('回调失败');

        return

    }

    console.log('成功');

})

1. **文件夹操作**

mkdir/medirSync 创建文件夹

readdir/readdirSync 读取文件夹

rmdir/rmdirSync 删除文件夹

* **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）recursive:true递归配置

callback：操作后的回调

//1.导入fs

const fs = require('fs');

//2.创建文件夹

fs.mkdir('./html',err =>{

    if(err){

        console.log('失败');

        return;

    }

    console.log('创建成功')

})

//2-2 递归创建

fs.mkdir('./a/b/c',{recursive: true},err =>{

    if(err){

        console.log('失败');

        return;

    }

    console.log('创建成功')

})

//2-3 读取文件夹

fs.readdir('../../资料',(err, data) =>{

    if(err){

        console.log('失败');

        return;

    }

    console.log(data)

})

**读取可以得到目标文件夹下的名称列表数组**

//2-4删除文件夹

fs.rmdir('./html',err=>{

    if(err){

        console.log('操作失败');

        return;

    }

    console.log('操作成功')

})

**目标文件夹中还有文件夹时删除会导致报错，此时想要清空文件夹的话用递归删除（不推荐使用）会有报错提示如下图**

//2-5 递归删除文件夹

fs.rmdir('./a',{recursive: true},err =>{

    if(err){

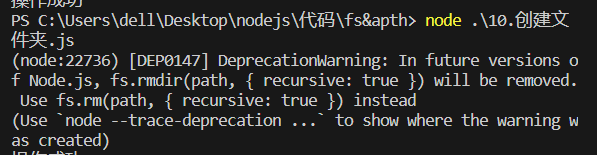
        console.log('操作失败');

        return;

    }

    console.log('操作成功')

})

****

**建议使用**

//2-6 递归删除文件夹的建议使用

fs.rm('./a',{recursive: true},err =>{

    if(err){

        console.log('操作失败');

        return;

    }

    console.log('操作成功')

})

1. **查看资源状态**

* **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

//1.导入fs

const fs = require('fs');

//2.查看资源状态

fs.stat('../',(err, data) =>{

    if(err){

        console.log('失败');

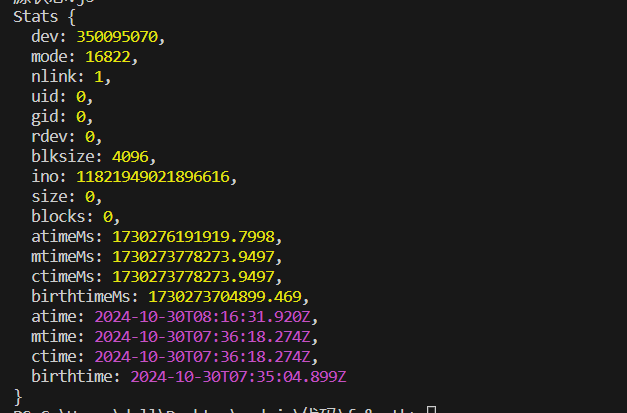
        return;

    }

    console.log(data)

})

* **输出**



* **结果值对象结构：**

size：文件体积

birthtime：创建时间

atime：最后访问时间

mtime：最后修改时间

    console.log(data.isFile())

    console.log(data.isDirectory())

isFile：检测是否为文件

isDirectory：检测是否问文件夹

1. **两种路径**

* **相对路径：（fs里参照的是命令行的工作目录）**

./或者什么都不写 =》当前文件夹下，../上一级文件夹

* **绝对路径**

盘符开头：D:/index.html（写C盘开头的话权限不够）

/开头，也是在文件根目录下

\_\_dirname =》始终保存的是所在文件的所在目录的绝对路径

1. **Path模块（操作路径）**
2. **API：**

path.resolve： 拼接规范的绝对路径 参数：（绝对路径，相对路径）

path.sep： 获取操作系统的路径分隔符

path.parse： 解析路径并返回对象

path.basename： 获取路径的基础名称

path.dirname： 获取路径的目录名

path.extname： 获取路径的扩展名

//1.导入fs

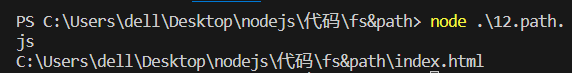
const fs = require('fs');

const path = require('path');

// resolve

console.log(path.resolve(\_\_dirname,'./index.html'))

打印结果



// sep 分隔符 windows系统=》'\'  Linux系统=》'/'

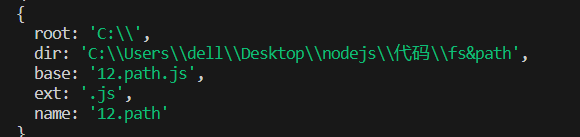
console.log(path.sep)

// parse 方法，用来解析路径 (\_\_filename：文件的绝对路径）

let str = \_\_filename

console.log(path.parse(str))

打印结果



//basename

console.log(path.basename(str))

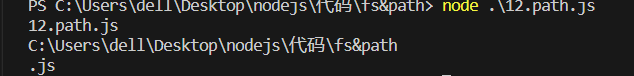
//dirname

console.log(path.dirname(str))

//extname

console.log(path.extname(str))

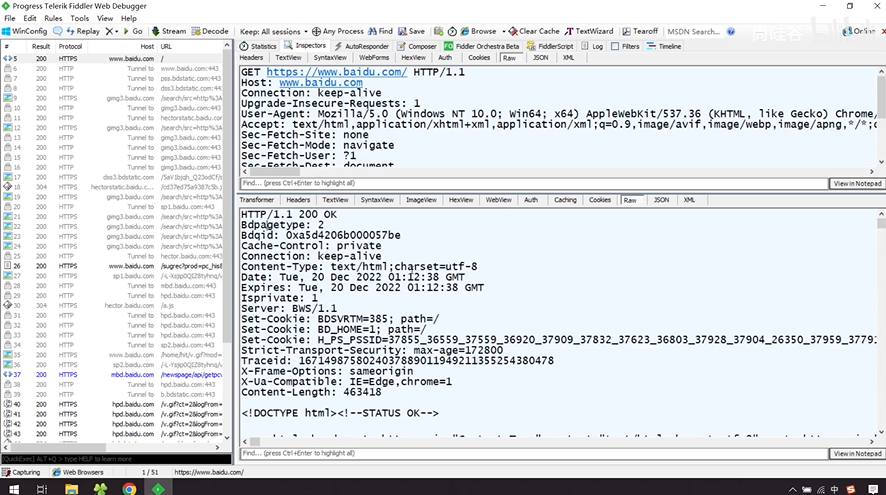
打印结果



1. **HTTP协议（Hypertext Transfer Protocol）=》超文本传输协议**
2. **协议：双方必须共同遵从的一组约定**



1. **HTTP协议报文 =》fiddler软件**



1. **请求报文的结构**



* **请求行：**



* **常见请求方法：**

GET：主要用于获取数据

POST：主要用于新增数据

PUT / PATCH：主要用于更新数据

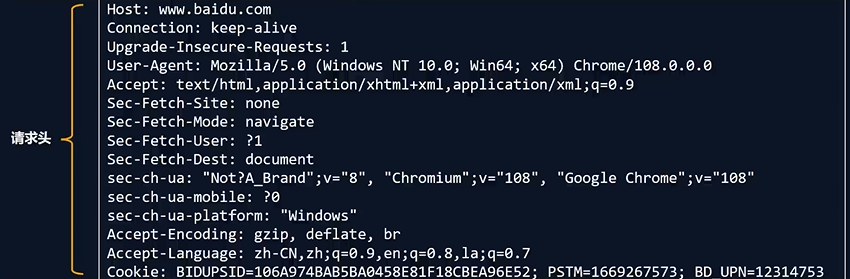
DELETE：主要用于删除数据

* **URL： =》Uniform Resource Locator 统一资源定位符 其本身也是一个字符串**

用来定位资源



* **请求头：由键值对组成，记录浏览器相关信息**



查看请求头含义参考：

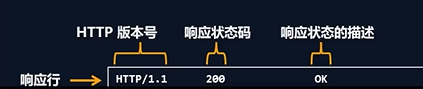
<https://developer.mozilla.org/zh-CN/Web/HTTP/Headers>

* **请求体：格式灵活，理论上可以写任何内容**

1. **响应报文的结构**



* **响应行**



**响应状态码： 响应状态描述：**

200：请求成功 OK

403：禁止请求 Forbidden

404：找不到资源 Not Found

500：服务器内部错误 Internal Server Error

**状态码分类：**

1XX：信息响应

2XX：成功响应

3XX：重定向消息

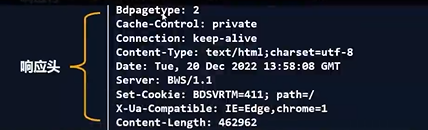
4XX：客户端错误响应

5XX：服务端错误响应

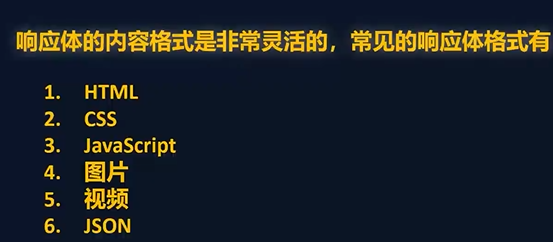
状态描述含义参考：

[HTTP 响应状态码 - HTTP | MDN (mozilla.org)](https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status)

* **响应头**



* **响应体**



1. **IP**
2. **IP的作用：**

IP用来标识网络中的设备，实现设备间通信。IP本身是一个数字标识。

1. **IP的分类：**

类型：

1. 本机回环IP地址：127.0.0.1~127.255.255.254
2. 局域网IP（私网IP）：192.168.0.0~192.168.255.255.255

172.16.0.0~172.31.255.255

10.0.0.0~10.255.255.255

1. 广域网IP（公网IP）：除上述之外
2. **端口（应用程序的数字标识）**
3. **端口的主要作用：**

实现不同主机应用程序之间的通信。

1. **创建一个HTTP服务**

// 1.导入HTTP模块

const http = require('http')

//2.创建服务对象

const server = http.createServer((request, response)=>{

    response.end('Hello HTTP Server'); //设置响应体

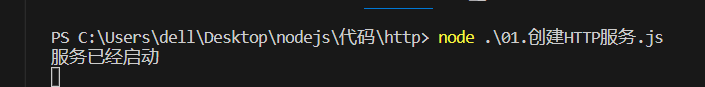
})

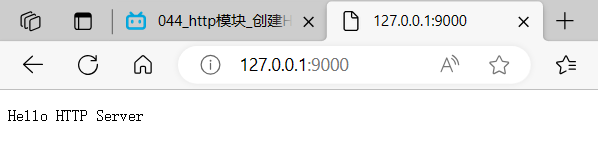
// 3. 监听端口，启动服务

server.listen(9000, ()=>{

    console.log('服务已经启动')

})





* **注意事项：**

1. 命令行ctrl + c停止服务
2. 当服务启动后，更新代码必须重启服务才能生效
3. 响应内容中文乱码的解决办法：

response.setHeader('content-type', 'text/html; charset = utf-8');

1. 端口号被占用：

ERROR:listen EADDRINUSE: address already in use :::9000

1. 关闭当前正在运行监听端口的服务（使用较多）
2. 修改为其他端口号
3. HTTP协议默认端口是80，HTTP服务开发常用端口有3000, 8080, 8090, 9000等

**如果端口被其他程序占用，可以使用资源监视器找到占用端口的程序，然后使用任务管理器关闭对应的程序**