**node笔记整理**

js包含ECMAScript（定义了js的基础），BOM（扩展了控制浏览器的API），DOM（扩展了页面操作的API）  
**区别**：NodeJs是面对系统文件等做操作。Js是主要针对浏览器做操作

顶层对象：nodeJs是global，js是window

用node去运行程序，node path/filename.\* || filename.\*

**Node模块作用域**：一个文件就是一个模块，每个模块都有自己的作用域。用var等声明的只能是模块下的变量。可以直接用global.name = 来声明

**模块加载系统**：require(模块相对路径或绝对路径) 当为相对路径时，会加载node\_modules或者node核心模块，且会自动补全文件后缀来查找（顺序是文件名=>.js=>.json=>.node）,如果都没有则报错

**引入后调用其他模块变量：**1，把变量挂载到global变量下。2，使用模块对象module(保存提供和当前模块有关的一些信息)，require的返回值就是引入模块的module.exports对象。尽量使用在exports上添加，而不要去修改他的指向

**内置API**

\_\_filename：当前文件被解析后的绝对路径，该属性是在模块作用域下的

\_\_dirname：返回文件所在目录的绝对路径，该属性也不是全局，是模块下的

**Global：**

**process：**进程对象，这个是全局对象，prccess=global.process

argv:一个包含命令行参数的数组，一个参数是node，当前运行的js文件

execPath：开启当前进程的绝对路径

env：返回用户的环境信息

version：返回node版本信息

versions：返回node及node依赖包版本信息

pid：当前进程pid（可以在任务管理器-进程看）

title： 当前进程的显示名称（getter和setter）

arch：返回当前CPU处理器架构arm/ia32/x64

platform：返回当前操作系统平台

cwd():返回当前进程的工作目录

chdir(directory):改变当前进程的工作目录

memoryUsage():返回node进程的内存使用情况，单位是byte

exit(code):退出

kill(pid):向进程发送信息

stdin（输入）、stdout（输出）：标准输入输出流（IO）。提供了操作输入数据和输出数据的方法。

例process.stdout.write(xxx), // 打印

Process.stdin.resume() //开启输入流

process.stdin.on('data',function(chunk){ console.log(chunk) }) // 监听用户输入时间与数据

process.stdout.write(`请输入a`)

if(!a){

a = Number(chunk)

process.stdout.write(`请输入b`)

} else {

b=Number(chunk)

process.stdout.write(`结果为：${a+b}`)

} //简易输入计算

**Buffer**类：是一个全局的类，在global下，有三种构造方式，number，array，string+encodeing(默认为utf-8)

Ps：分配空间大小以后其长度为固定且不可更改。一个中文占三个字节长度（字符截取）

new Buffer(10 || [1,2,3] ||’string’， ‘utf-8’)

buffer.write // 写入对象

buffer.toString(‘utf-8’, 1,3) // 把buffer对象转为string

buffer.toJson() // 转为json对象

slice // 截取，会改变原对象

bf.copy(targetBf,start,sourceStart,sourceEnd ) // 拷贝

write(string,buffer\_start,str\_length,str\_encoding)

以上方法都是实例化buffer才可使用

以下为类方法（静态方法）

isEncoding(‘utf-8’) // 如果传入的方法是有效的，则返回true

byteLength(str，’ascii’) // 返回字节长度

concat(list, strLength) // 拼接数组内的buffer

**FileSystem**：文件系统 IO

`Require(‘fs’)

`方法

Open(path(文件路径),flags(打开方式),[mode]（设置文件模式，读/写/执行）,callback(回调，err错误保存，fd打开标识类似于定时器返回值))，异步打开文件

openSync // 同步打开文件

fs = require(‘fs’)

fs.open(‘1.txt’,’r+’,function(err, fd){})

fs.read(fd(文件标识), buffer(buffer对象), offset(偏移量，填入的位置)，length(读取的内容长度),position(读取的文件中的起始位置),callback) // 读取文件内容

fs.readSync

var bf1 = new Buffer()

fs.read(fd,bf1,0,4,null,function(err, bufferObjLength,bufferObj){})

write(fd,bf,0,3,0,function(err, dataLength,bufferObj){}) //写入文件

write(fd,’1234’ ,5(位数),’utf-8’ , function(err, dataLength,bufferObj){})

close(fd,function(){}) // 关闭文件

writeFile(filename,data,[option],callback) //向指定文件写入数据，如文件不存在则新建

writeFileSync // 同步写入

fs.writeFile(filename,’hello’,function(){})

**appendFile**(filename,data,callback) // 向文件插入内容，但不新建文件

**exists**(filename,callback) // 检测文件是否存在

exists(filename,function(isExists){

console.log(isExists) // 如果不存在则为false

})

readFile(filename,callback) // 读取文件

fs.readFile(filename,function(err, data){})

unlink(filename,callback) // 删除文件

fs.unlink(filename,function(err){})

rename(oldName,newName,callback) // 重命名

stat(filename,callback) // 返回文件状态信息

watch(filename,callbacl) // 查看文件是否改变，监听例如重命名，文件更改。

fs.watch(filename,function(ev,file){ // file不是总会返回，所以用之前最好都先判断

// ev是触发的事件，fn是改变的状态

})

mkdir(path,[mode], callback) // 创建文件夹

redir(path,[mode], callback) // 删除文件夹

readdir(path,[mode], callback) // 读取文件夹

readdir(path,function(err,fileList){ })