Node.js

1. Node.js是什么

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。其底层由C/C++实现。

2. Node.js的特点

```
优点：
    1)    异步非阻塞的I/O（I/O线程池）
            I/O操作分别对应读写操作，线程池指的是将多个线程放在一块内存空间中等候调用。
    2)    特别适用于I/O密集型应用
            cpu密集型：要进行大量的计算，消耗CPU资源；
            I/O密集型：CPU消耗很少，任务的大部分时间都在等待IO操作完成。
    3)    事件循环机制（与浏览器的不一样）
    4)    单线程（成也单线程，败也单线程）
    5)    跨平台
不足：
    1)    回调函数嵌套太多、太深（俗称回调地狱）
    2)    单线程，处理不好CPU  密集型任务
```

3. Node中的函数

```
Node中任何一个模块（js文件）都被一个外层函数所包裹：
function (exports, require, module, __filename, __dirname) {}

这个函数是所有模块都有的，node编译时往其中注入5个参数：
    exports       暴露模块
    require       引入模块
    module        exports属性暴露模块
    __filename    文件的绝对路径
    __dirname     文件夹的绝对路径

为什么要设计这个外层函数（这个外层函数有什么作用？）
    1).用于支持模块化语法
    2).隐藏服务器内部实现(从作用域角度去看)

查看这个外层函数的方法：
console.log(arguments.callee.toString()) //输出外层函数
```

4. Node中的global

```
Node端，js的组成：
  1.没有了BOM ----->  因为服务器不需要（服务端没有浏览器对象）
  2.没有了DOM ----->  因为没有浏览器窗口
  3.几乎包含了所有的ES规范
  4.没有了window，但是取而代之的是一个叫做global的全局变量。

global的一些常用属性：
clearImmediate:清空立即执行函数
clearInterval:清除循环定时器
clearTimeout: 清除延迟定时器
```

```
setImmediate:设置立即执行函数
setInterval:设置循环定时器
setTimeout: 设置延迟定时器

在Node中禁止函数的this指向global，而是指向了一个空对象
console.log(this)    // {}
console.log(global)
```

5. Node的事件循环模型

```
  第一个阶段：timers(定时器阶段--setTimeout,setInterval)
            1.开始计时
            2.执行定时器的回调

  第二个阶段：pending callbacks （系统阶段）

  第三个阶段：idle, prepare （准备阶段）

  第四个阶段：poll(轮询阶段，核心)
            ---如果回调队列里有待执行的回调函数（除setTimeout,setInterval,setImmediate之外）
                从回调队列中取出回调函数，同步执行(一个一个执行)，直到回调队列为空了，或者达到系统最大限度。
            ---如果回调队列为空
                ---如果有设置过setImmediate
                    进入下一个check阶段，目的：为了执行setImmediate所设置的回调。
                ---如果未设置过setImmediate
                    在此阶段停留，等待回调函数被插入回调队列。
                    若定时器到点了，进入下一个check阶段，原因：为了走第五阶段，随后走第六阶段，随后第一阶段(最终目的)

  第五个阶段：check （专门用于执行setImmediate所设置的回调）

  第六个阶段：close callbacks （关闭回调阶段）

  process.nextTick() ---- 用于设置立即执行函数(“VIP”-----能在任意阶段优先执行，但不能快过主线程)
```

6. 包与npm包管理器

```
  1.什么是包？
      我们电脑上的文件夹，包含了某些特定的文件，符合了某些特定的结构，就是一个包。

  2.一个标准的包，应该包含哪些内容？
      1)    package.json -------- 描述文件（包的 “说明书”，必须要有！！！）
      2)    bin ------------------可执行二进制文件
      3)    lib ----------------- 经过编译后的js代码
      4)    doc ----------------- 文档（说明文档、bug修复文档、版本变更记录文档）
      5)    test ---------------- 一些测试报告

  3.如何让一个普通文件夹变成一个包？
        让这个文件夹拥有一个：package.json文件即可,且package.json里面的内容要合法。
        执行命令：npm init
        包名的要求：不能有中文、不能有大写字母、同时尽量不要以数字开头、不能与npm仓库上其他包同名。
```

4.npm与node的关系？（npm：node package manager）

安装node后自动安装npm（npm是node官方出的包管理器，专门用于管理包）

5.npm的常用命令？

一、【搜索】：

1.npm search xxxxx

2.通过网址搜索：www.npmjs.com

二、【安装】：(安装之前必须保证文件夹内有package.json，且里面的内容格式合法)

1.npm install xxxxx --save 或 npm i xxxx -S 或 npm i xxxx

备注：

(1).局部安装完的第三方包，放在当前目录中node_modules这个文件夹里

(2).安装完毕会自动产生一个package-lock.json(npm版本在5以后才有)，里面缓存的是每个下载过的包的地址，目的是 下次安装时速度快一些。

(3).当安装完一个包，该包的名字会自动写入到package.json中的【dependencies(生产依赖)】里。npm5及之前版本 要加上--save后缀才可以。

2.npm install xxxxx --save-dev 或 npm i xxxxx -D 安装包并将该包写入到【devDependencies(开发依赖 中)】

备注：什么是生产依赖与开发依赖？

1.只在开发时(写代码时)时才用到的库，就是开发依赖 ----- 例如：语法检查、压缩代码、扩展css前缀的包。

2.在生产环境中(项目上线)不可缺少的，就是生产依赖 ------ 例如：jquery、bootStrap等等。

3.注意：某些包即属于开发依赖，又属于生产依赖 -------例如：jquery。

3.npm i xxxx -g 全局安装xxxx包（一般来说，带有指令集的包要进行全局安装，例如：browserify、babel等）

全局安装的包，其指令到处可用，如果该包不带有指令，就无需全局安装。

查看全局安装的位置：npm root -g

4.npm i xxx@yyy ：安装xxx包的yyy版本

5.npm i ：安装package.json中声明的所有包

三、【移除】：

npm remove xxxxx 在node_module中删除xxxx包，同时会删除该包在package.json中的声明

四、【其他命令】：

1.npm aduit fix ：检测项目依赖中的一些问题，并且尝试着修复。

2.npm view xxxxx versions ：查看远程npm仓库中xxxx包的所有版本信息

3.npm view xxxxx version ：查看npm仓库中xxxx包的最新版本

4.npm ls xxxx ：查看我们所安装的xxxx包的版本

五、【关于版本 号的说明】：

"^3.x.x" ：锁定大版本，以后安装包的时候，保证包是3.x.x版本，x默认取最新的。

"~3.1.x" ：锁定小版本，以后安装包的时候，保证包是3.1.x版本，x默认取最新的。

"3.1.1" ：锁定完整版本，以后安装包的时候，保证包必须是3.1.1版本。

7. Buffer缓冲器

```
Buffer是什么：
    1.它是一个【类似于数组】的对象，用于存储数据（存储的是二进制数据）。
    2.Buffer的效率很高，存储和读取很快，它是直接对计算机的内存进行操作。
    3.Buffer的大小一旦确定了，不可修改。
    4.每个元素占用内存的大小为1字节。
    5.Buffer是Node中的非常核心的模块，无需下载、无需引入,直接即可使用。
```

8. Node中的文件操作

```
1. Node中的文件系统：
    1.在NodeJs中有一个文件系统，所谓的文件系统，就是对计算机中的文件进行增删改查等操作。
    2.在NodeJs中，给我们提供了一个模块，叫做fs模块(文件系统)，专门用于操作文件。
    3.fs模块是Node的核心模块，使用的时候，无需下载，直接引入。

2.异步文件写入 （简单文件写入）
  fs.writeFile(file, data[, options], callback)
           --file：要写入的文件路径+文件名+后缀
           --data：要写入的数据
           --options：配置对象(可选参数)
                 --encoding:设置文件的编码方式，默认值：utf8(万国码)
                 --mode:设置文件的操作权限，默认值是：0o666 = 0o222 + 0o444
                      --0o111:文件可被执行的权限   .exe .msc 几乎不用，linux有自己一套操作
方法。
                      --0o222:文件可被写入的权限
                      --0o444:文件可被读取的权限
                 --flag:打开文件要执行的操作，默认值是'w'
                      --a ：追加
                      --w ：写入
           --callback：回调函数
                --err：错误对象
在Node中有这样一个原则：错误优先。

3.创建一个可写流
  fs.createWriteStream(path[, options])
      --path:要写入文件的路径+文件名+文件后缀
      --options：配置对象（可选参数）
          --flags:打开文件要执行的操作，默认值是'w'
          --encoding :设置文件的编码方式，默认值：utf8(万国码)
          --fd : 文件统一标识符，linux下文件标识符
          --mode :设置文件的操作权限，默认值是：0o666 = 0o222 + 0o444
          --autoClose : 自动关闭 --- 文件，默认值：true
          --emitClose : 关闭 --- 文件，默认值：false
          --start : 写入文件的起始位置(偏移量)

    // 创建可写流案例：
    let fs = require('fs')

    //创建一个可写流----水管到位了
    let ws = fs.createWriteStream(__dirname+'/demo.txt',{start:10})

    //只要用到了流，就必须监测流的状态
    ws.on('open',function () {
      console.log('可写流打开了')
    })
```

```
    ws.on('close',function () {
      console.log('可写流关闭了')
    })

    //使用可写流写入数据
    ws.write('美女\n')
    ws.write('霉女？\n')
    ws.write('到底是哪一个？\n')
    //ws.close() //如果在Node的8版本中，使用此方法关闭流会造成数据丢失
    ws.end() //在Node的8版本中，要用end方法关闭流

4.简单文件读取
  fs.readFile(path[, options], callback)
    --path：要读取文件的路径+文件名+后缀
    --options：配置对象（可选）
    --callback：回调
        --err：错误对象
        --data：读取出来的数据

5.流式文件读取
  fs.createReadStream(path[, options])
    --path:要读取的文件路径+文件名+后缀
    --options:
        --flags:打开文件要执行的操作，默认值是'w'
        --encoding:设置文件的编码方式，默认值：utf8(万国码)
        --fd: 文件统一标识符，linux下文件标识符
        --mode:设置文件的操作权限，默认值是：0o666 = 0o222 + 0o444
        --autoClose: 自动关闭 --- 文件，默认值：true
        --emitClose: 强制关闭 --- 文件，默认值：false
        --start ：起始偏移量
        --end ： 结束偏移量
        --highWaterMark：每次读取数据的大小，默认值是64 * 1024

    let {createReadStream,createWriteStream} = require('fs')

    //创建可读流案例
    let rs = createReadStream(__dirname+'/music.mp3',{
      highWaterMark:10 * 1024 * 1024,
      //start:60000,
      //end:120000
    })

    //创建一个可写流
    let ws = createWriteStream('../haha.mp3')

    //只要用到了流，就必须监测流的状态
    rs.on('open',function () {
      console.log('可读流打开了')
    })
    //可读流读取完毕后会自动关闭
    rs.on('close',function () {
      console.log('可读流关闭了')
      ws.close()   // 在可读流关闭的时候关闭可写流
    })
    ws.on('open',function () {
      console.log('可写流打开了')
    })
    ws.on('close',function () {
```

```
  console.log('可写流关闭了')
})

//给可读流绑定一个data事件，就会触发可读流自动读取内容。
rs.on('data',function (data) {
  //Buffer实例的length属性，是表示该Buffer实例占用内存空间的大小
  console.log(data.length) //输出的是65536，每次读取64KB的内容
  ws.write(data)
  //ws.close() //若在此处关闭流，会写入一次，后续数据丢失
})
//ws.close() //若在此处关闭流，导致无法写入数据（在主线程上）
```