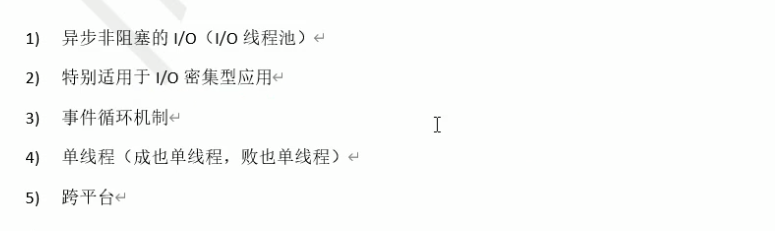
## Node.js 特点：

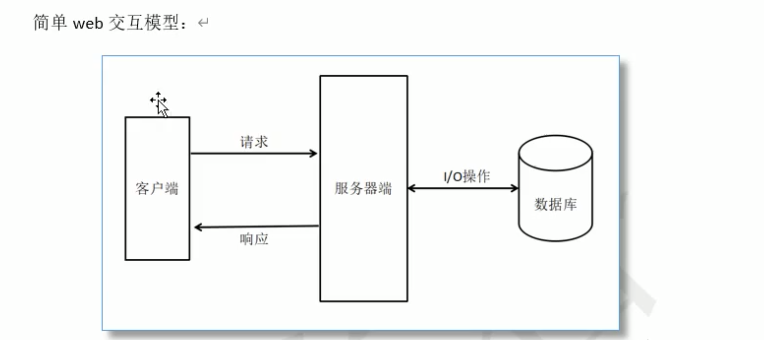
### 优点：



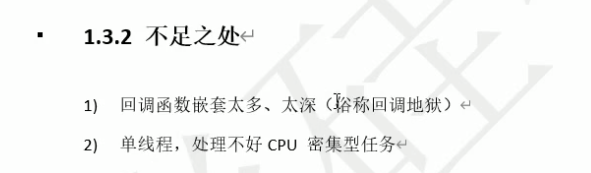
I/O密集型：频繁操作I/O；服务器端只有一个线程处理请求、响应和IO

单线程运行“东西”，如果想实现“异步”，就必须有自己的“事件循环模型”

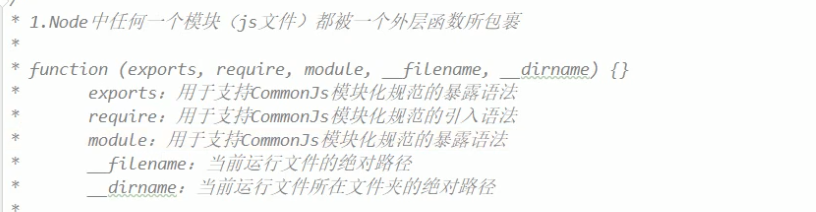
Java是多线程，适用于CPU密集型。服务器端相当于有多个线程处理请求、响应和IO

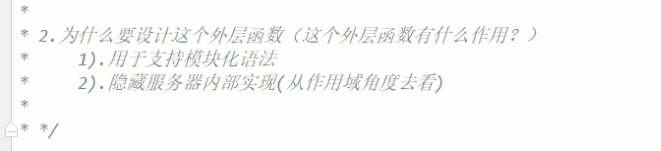


### 缺点：



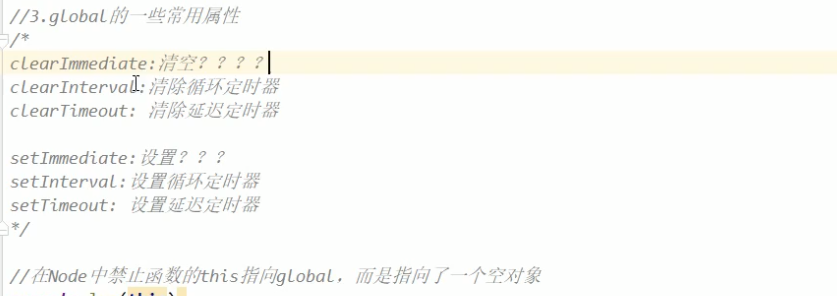
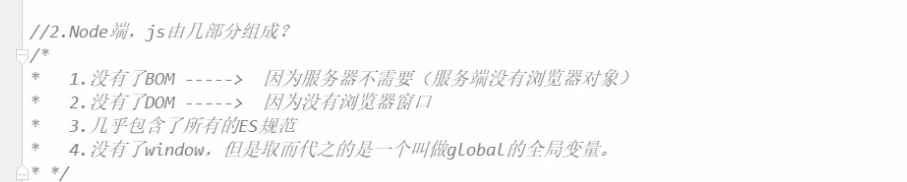
## Node中函数的特点：



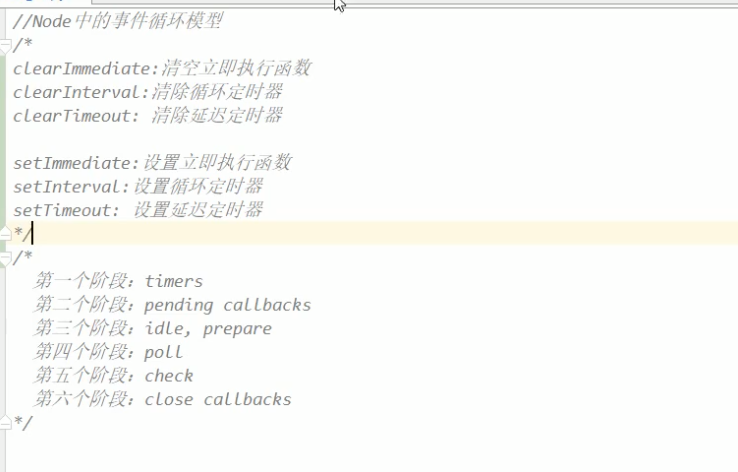


## Node中的global：





## Node中的事件循环模型：



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，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，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。

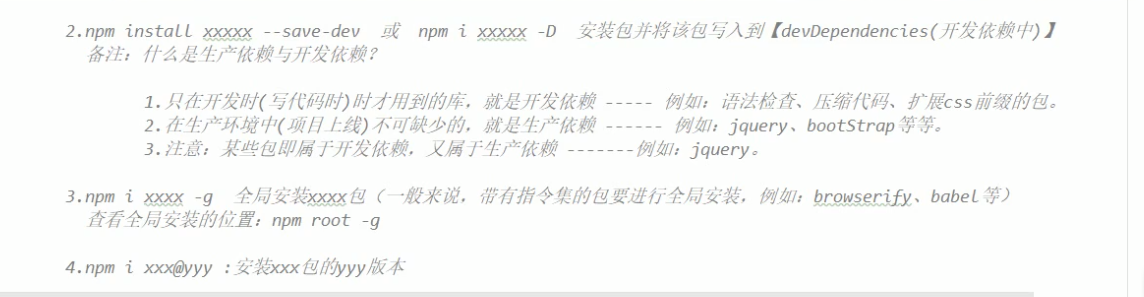
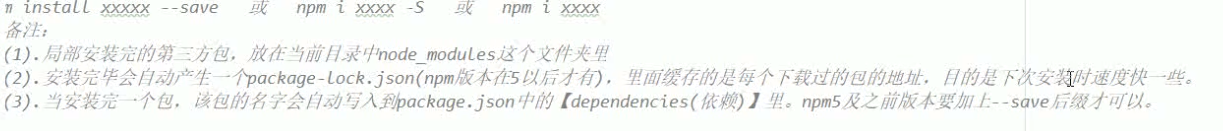
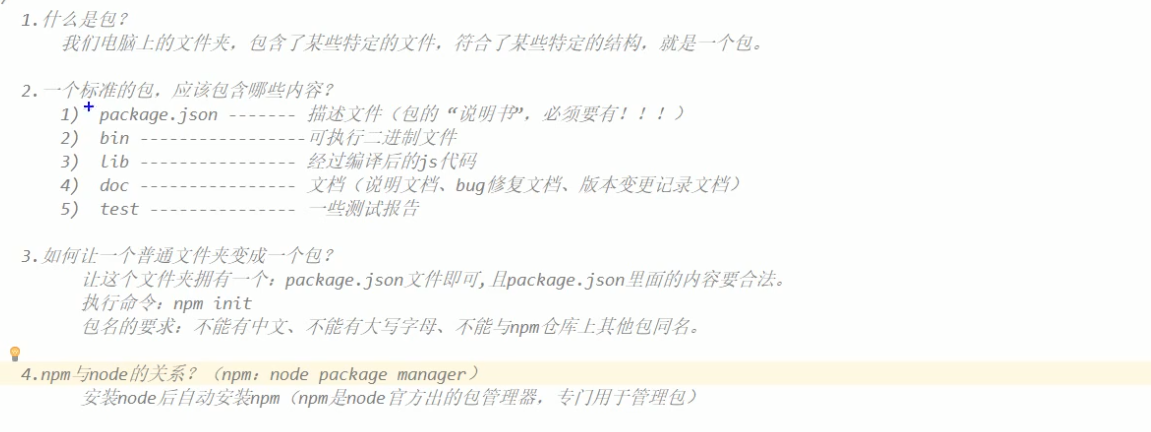
若定时器到点了，进入下一个check阶段，原因：为了走第五第六阶段，随后第一阶段（最终目的）

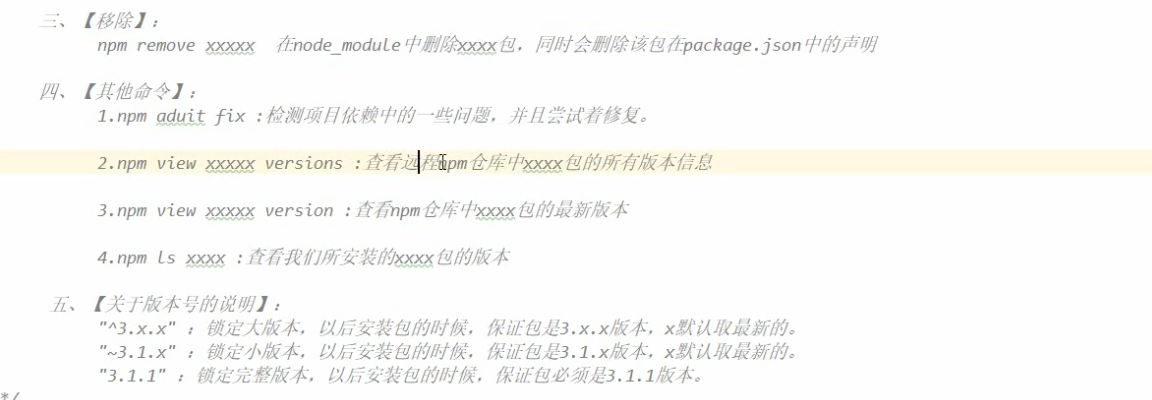
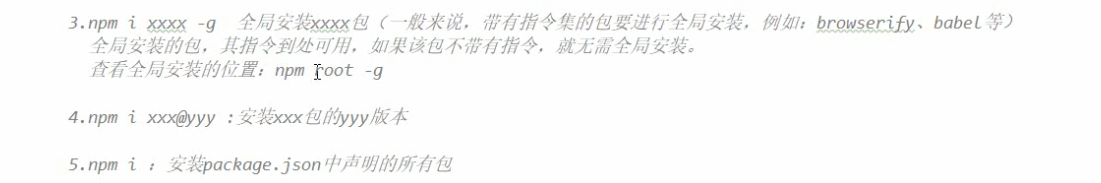
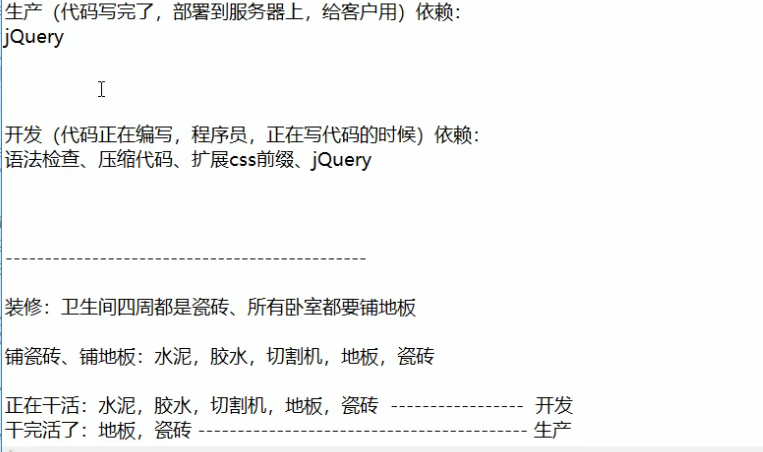
第五个阶段：check（专门用于执行setImmediate所设置的回调）

第六个阶段：close callbacks（关闭回调阶段）

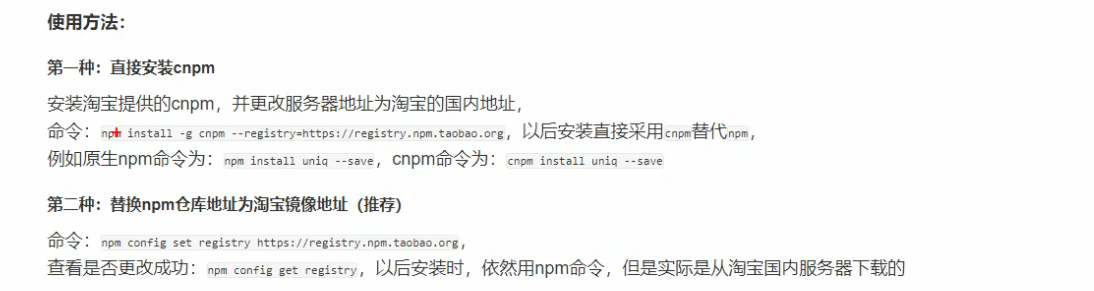
process.nextTick() ---用于设置立即执行函数（“VIP”，能在任意阶段优先执行）

## 包与包管理器：

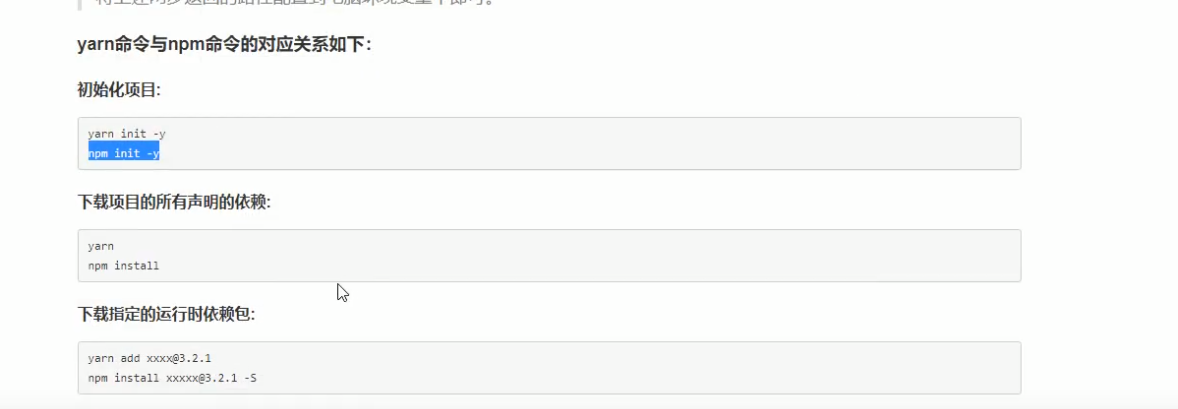




## cnpm的使用：

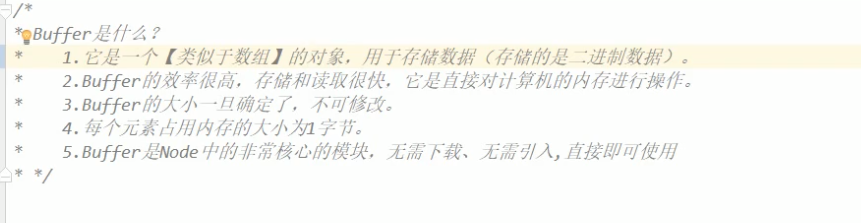


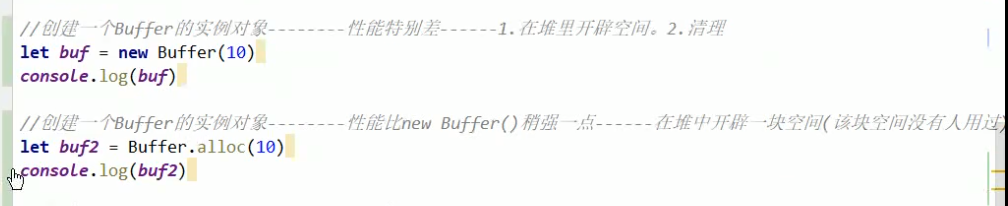
## yarn的使用：



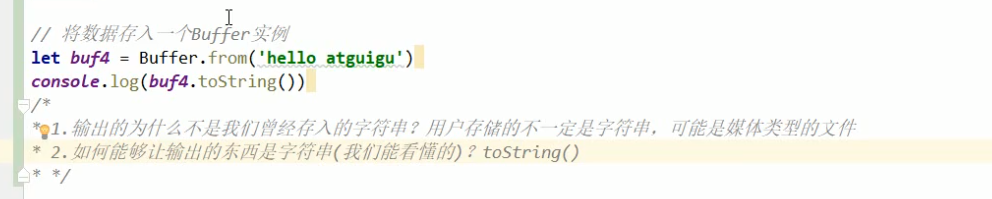
## Buffer缓冲器：





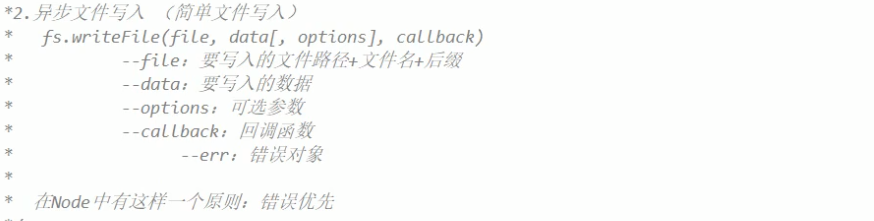
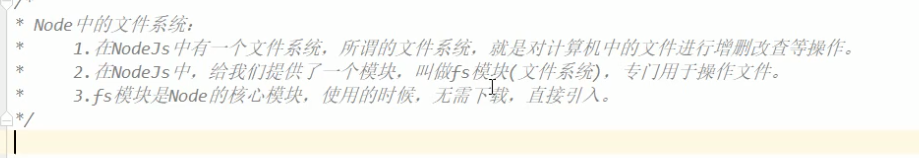


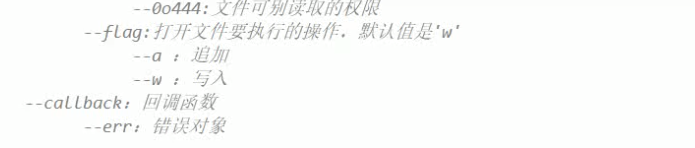
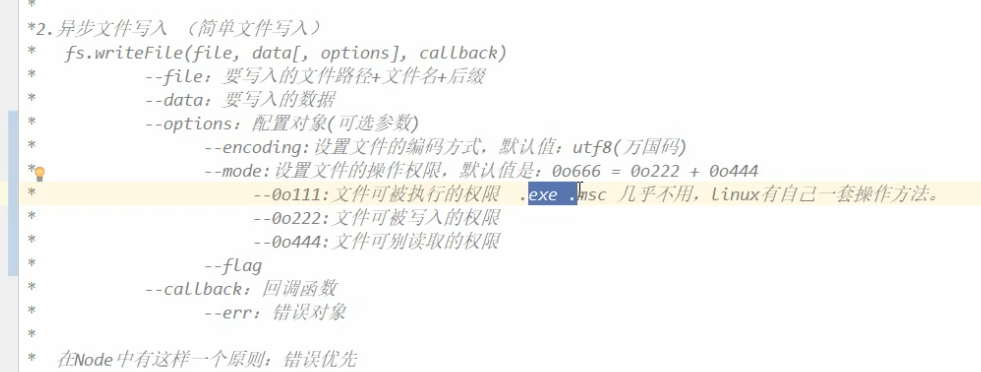


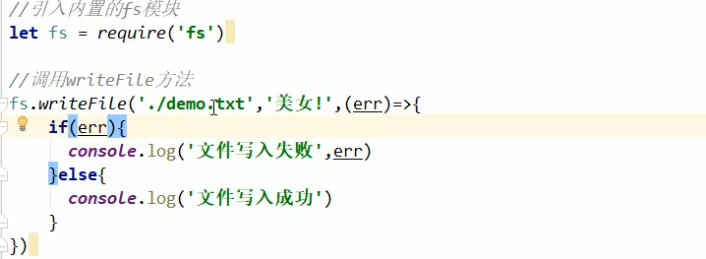


## Node中的文件写入操作：

### 简单文件写入.js：







let fs = require('fs');

fs.writeFile(\_\_dirname+'/demo.txt','wtf',(err)=>{

    if(err){

        console.log('文件写入失败！',err)

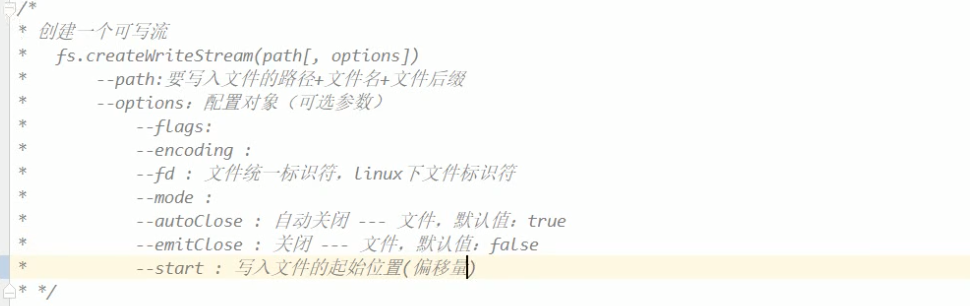
    }else{

        console.log('文件写入成功')

    }

})

### 流式文件写入.js：





    let fs = require('fs');

    let ws = fs.createWriteStream(\_\_dirname+'/demo.txt')

    ws.on('open',()=>{

        console.log('可写流打开了')

    })

    ws.on('close',()=>{

        console.log('可写流关闭了')

    })

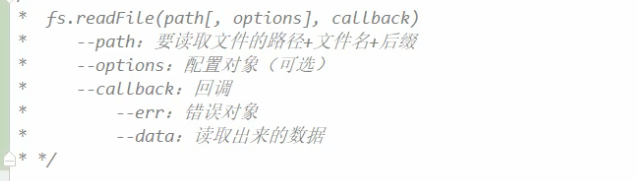
    ws.write('wtf!\n')

    ws.write('hahaha')

    ws.close();

## Node中的文件读取操作：

### 简单文件读取.js:



简单文件写入和简单文件读取，都是一次性把所有要读取或要写入的内容加到内存中，容易造成内存泄露。



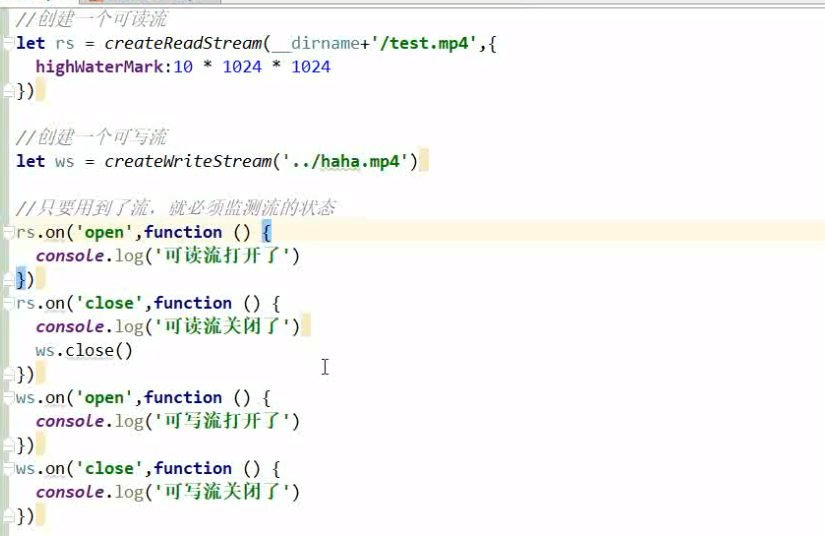
### 流式文件读取.js:

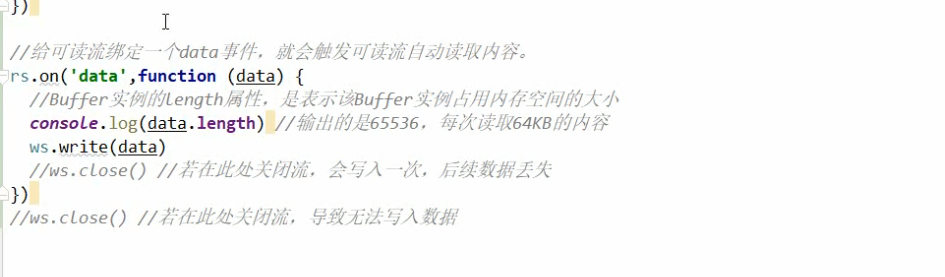




{createReadStream} = require(‘fs’)，对象解构赋值







读取+写入

如果写在回调函数里面，只写入了一次，因为每次读取数据大小有限。

如果写在回调函数外面，就是在主线程上，回调函数运行在主线程结束之后。

所以要写在可读流关闭的回调函数里，同步关闭。