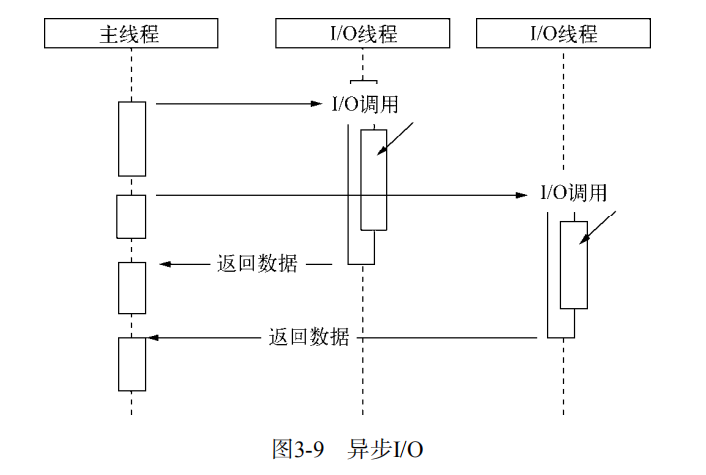
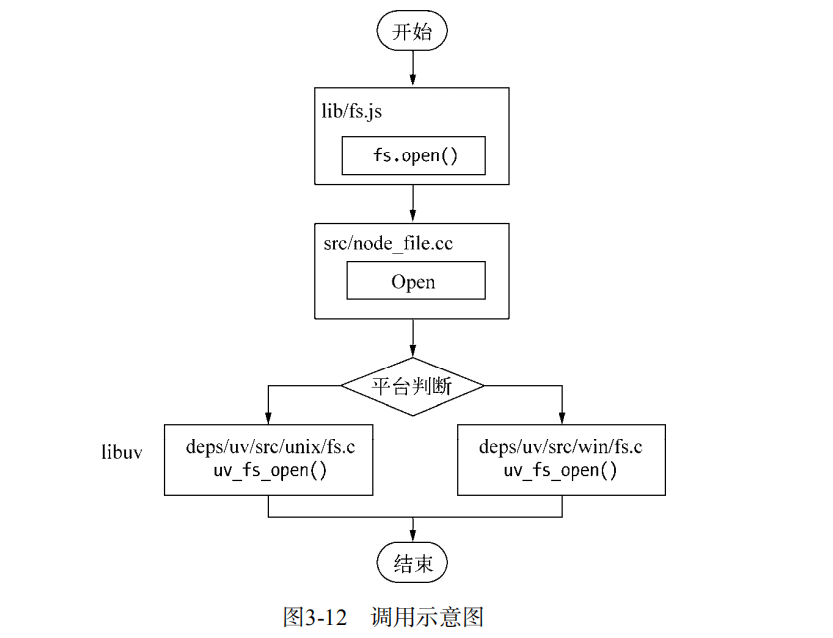
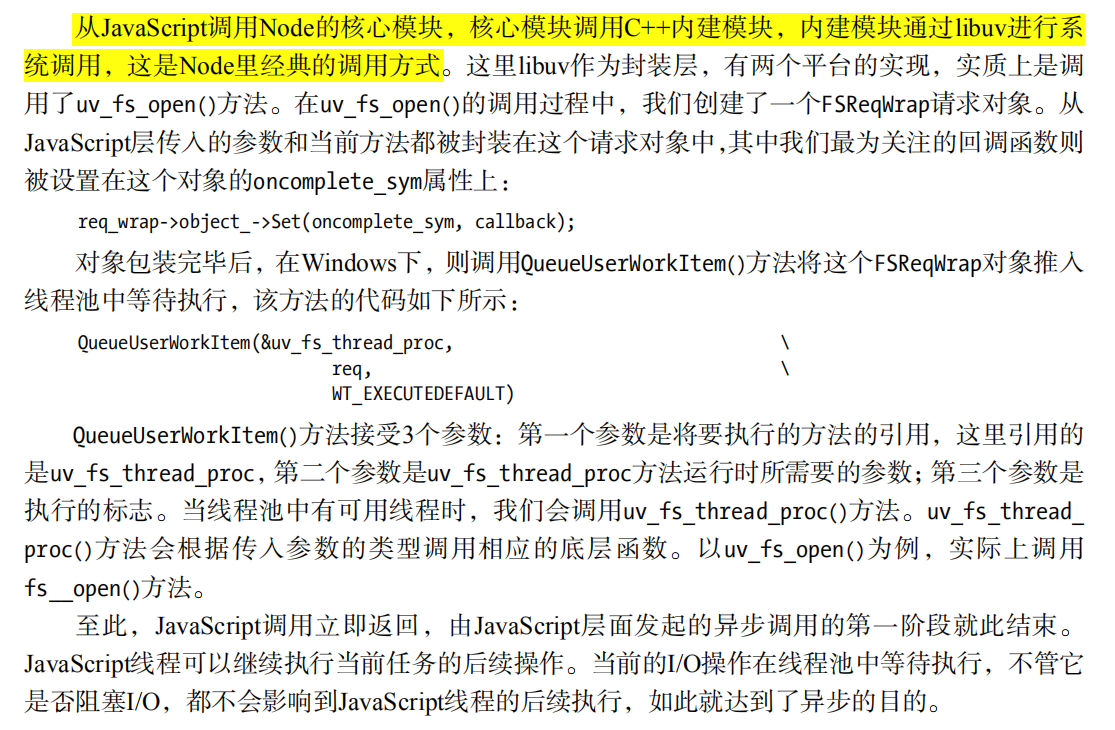
# 异步io

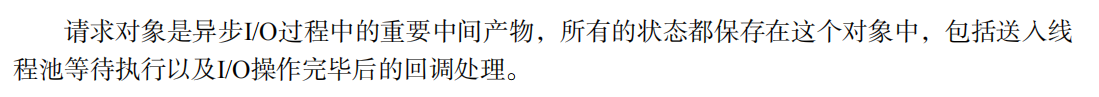
是一个伪造的异步io。而并非是操作系统的异步io模型。

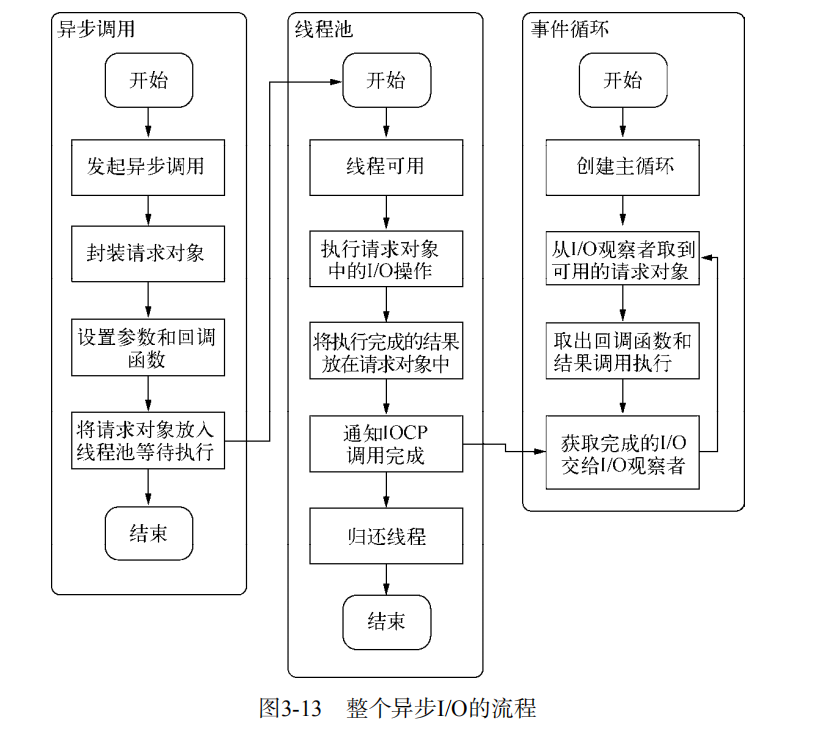


在调用时，会去调用node核心模块，然后调用C++内建模块，再通过libuv判断系统，根据不同的操作系统调用不同的实现。比如一个调用fs.open()根据路径获取文件描述符的操作：









通过这张图可以看出，nodejs执行用户代码都是在单线程执行的。

# 异步调用

有多个观察者

# 异步编程

高阶函数：将函数作为参数或者是返回值的函数就是高阶函数

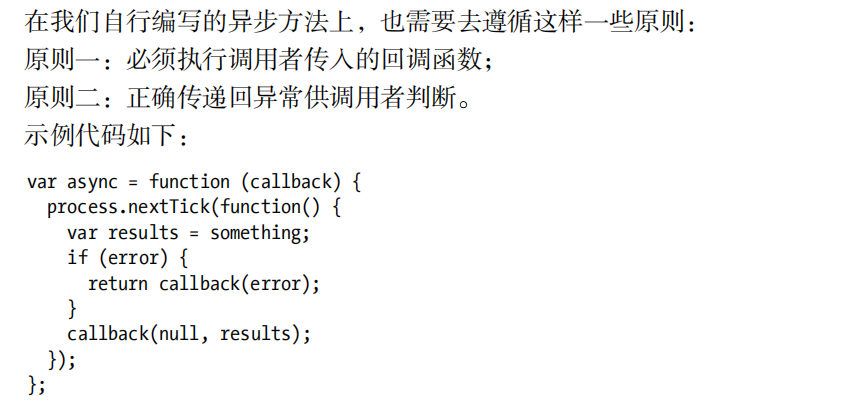
偏函数：通过参数将定制返回特定的函数，就是偏函数

利用事件循环的方式，javascript线程和io线程池里的各个线程

## 异步的问题

1. 异常处理难

应该有个要求就是：代码执行没问题，就执行回调函数，如果有问题就返回回调函数，同时回调函数必须包含错误的参数。



这里是指执行nextTick出现错误时，会将err传递给它包含的回调函数。

而我们写的callback也应该包含error参数，去反馈var results=something这一层出现的问题。反馈给callback执行的方法。

1. 代码嵌套
2. ~~阻塞代码~~
3. ~~多线程编程~~
4. ~~异步转同步~~

# 异步控制

未看

# 堆内存

Node 由于使用V8作为脚本引擎，虽然性能提高了，但是V8只能调用系统（32位、64位）堆内存（0.7GB、1.4GB）

使用以下命令，可以在执行时放宽V8的默认限制

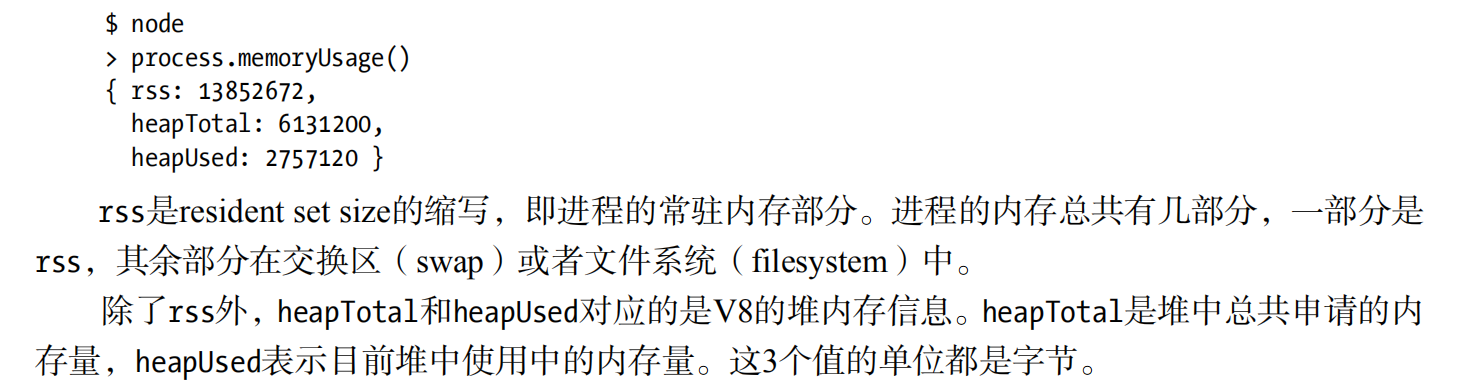
node --max-old-space-size=1700 test.js // 单位为MB ，设置老年代

node --max-new-space-size=1024 test.js // 单位为KB，设置新生代

生效后就不能再动态改变。

## 查看当前堆内存情况：

Process.memoryUsage()



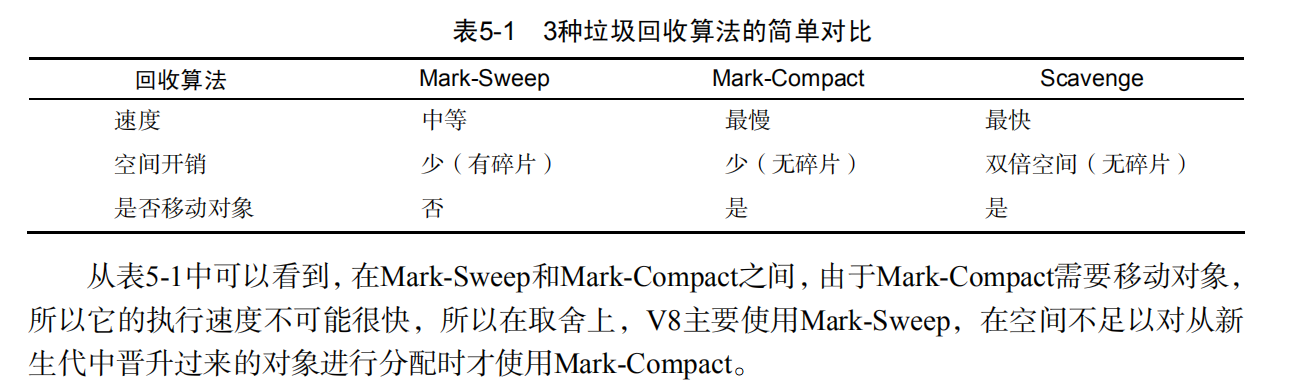
上面说一个V8实例只能调用系统内存的 0.7或者是1.4GB，而heapTotal虽然不会很大，但如果不够用，就会继续申请，最大不能超过0.7或1.4GB。

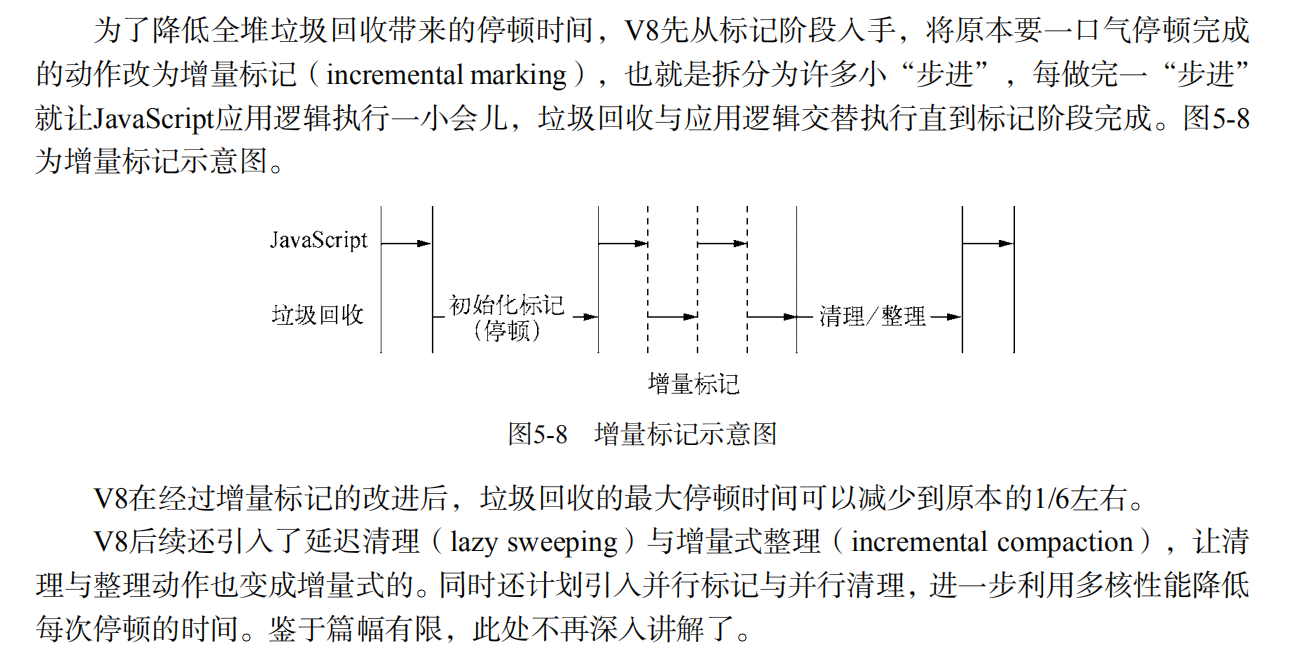
而rss不和堆内存相关，它应该是堆内存和堆外内存的总和，因为存在堆外内存，所以大小可以超过V8的限制。Buffer就是存放在堆外内存的。

年轻代使用复制算法（scavenge ），

晋级老年代（两个条件符合一个）：1.是否经历过scavenge 2.to占比25%

老年代使用mark-sweep和mark-compact





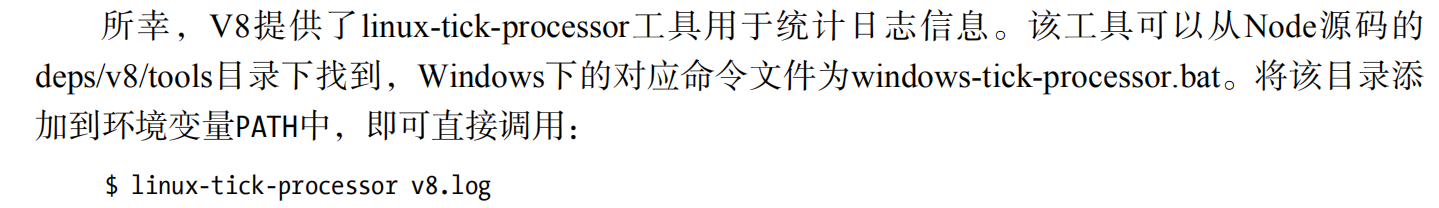
重点：chrome浏览器每开一个选项卡，就会新建一个V8实例。怪不得那么卡。

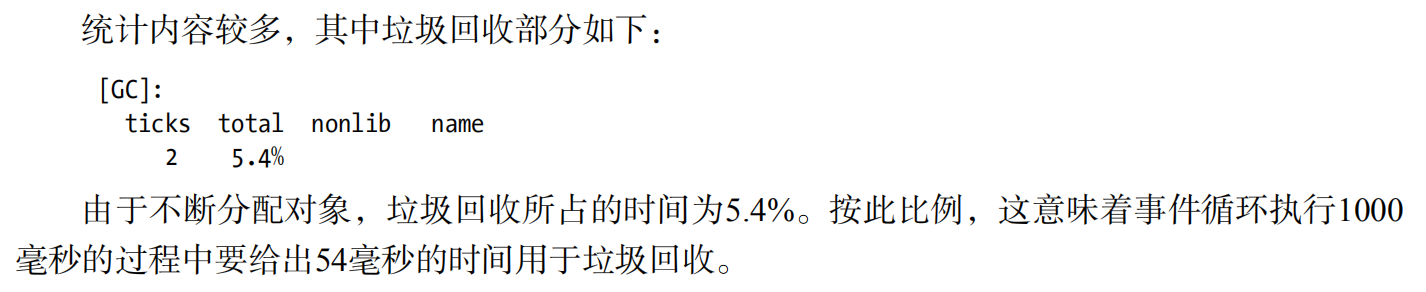
# 日志

node --trace\_gc -e "var a=[];for(var i=0;i<1000000;i++) a.push(new Array(100));" > gc.log

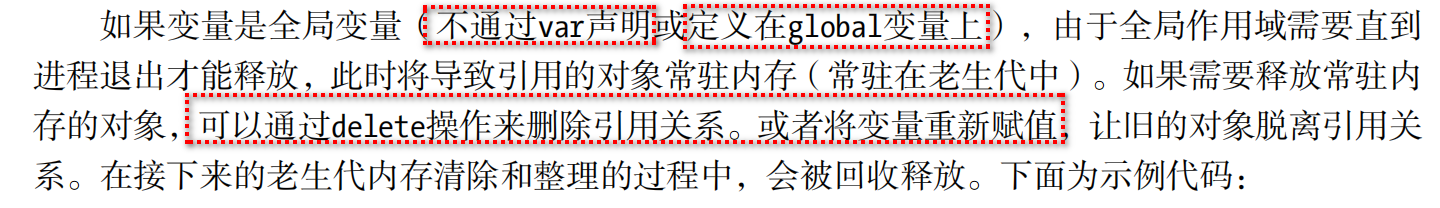
Node --prof 得到V8执行时的性能分析数据，得出的数据不具可读性

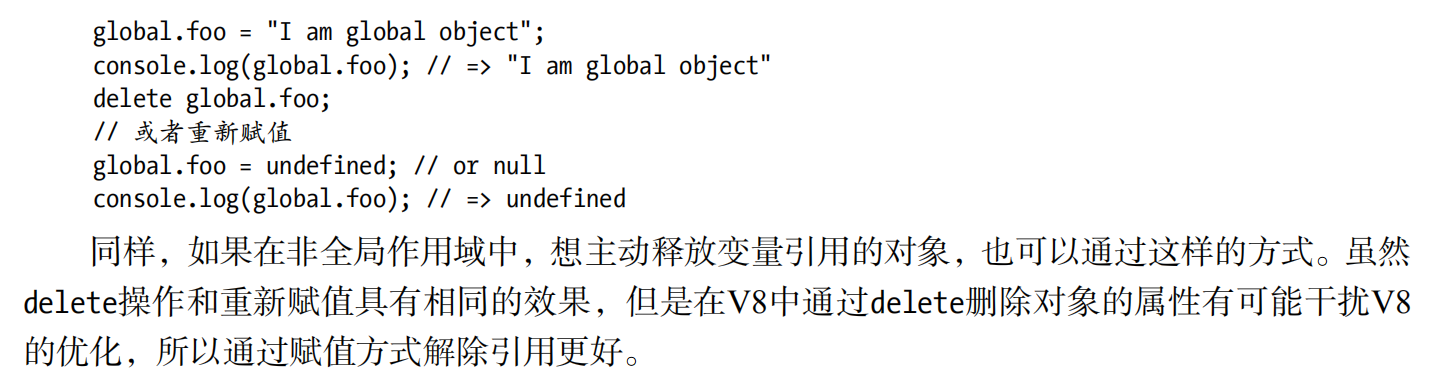
使用linux-tick-processor



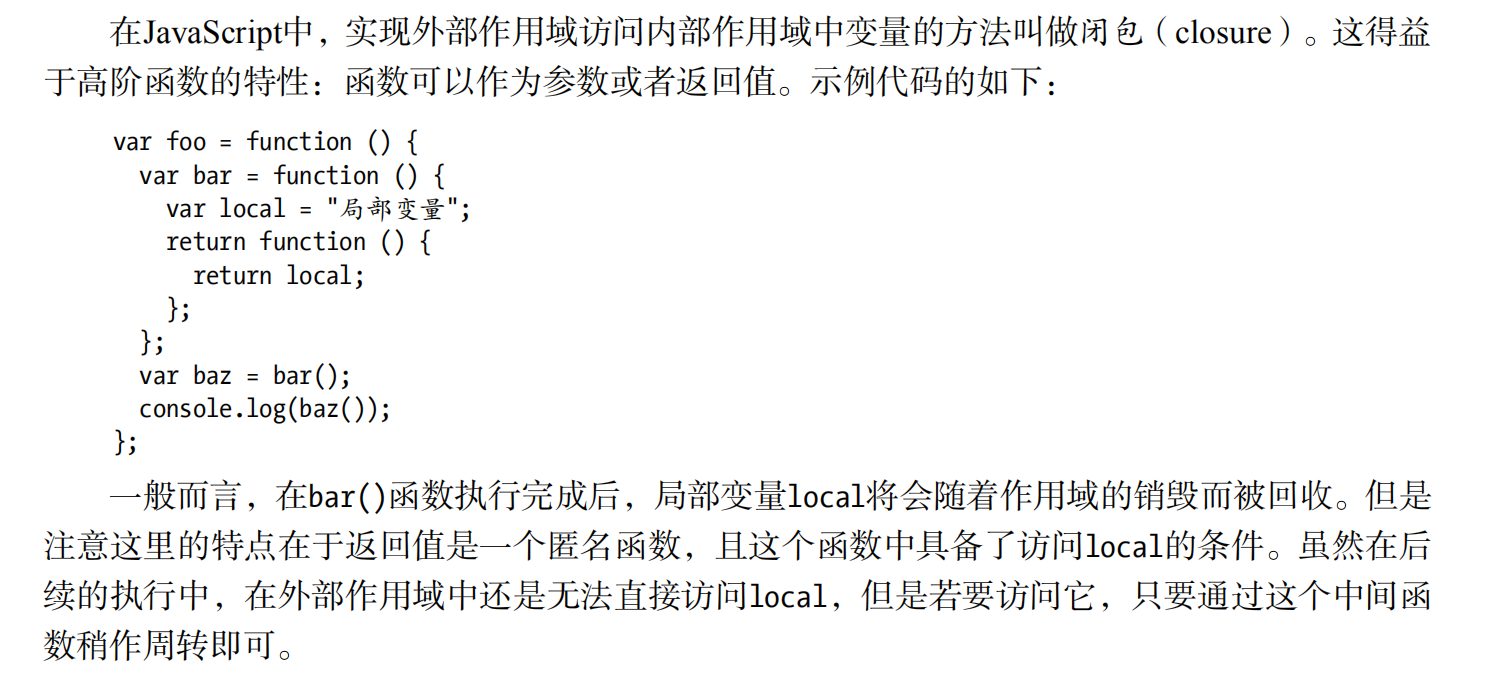


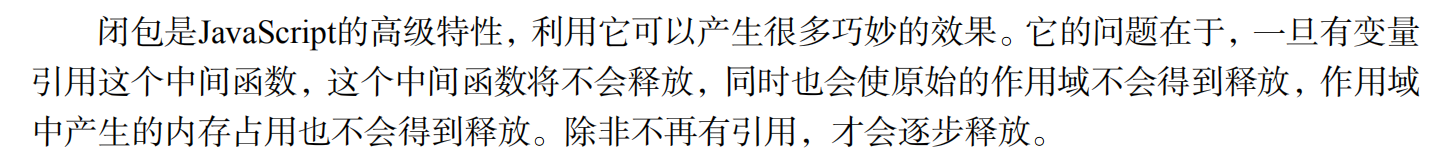
# 全局变量解除引用





# 闭包



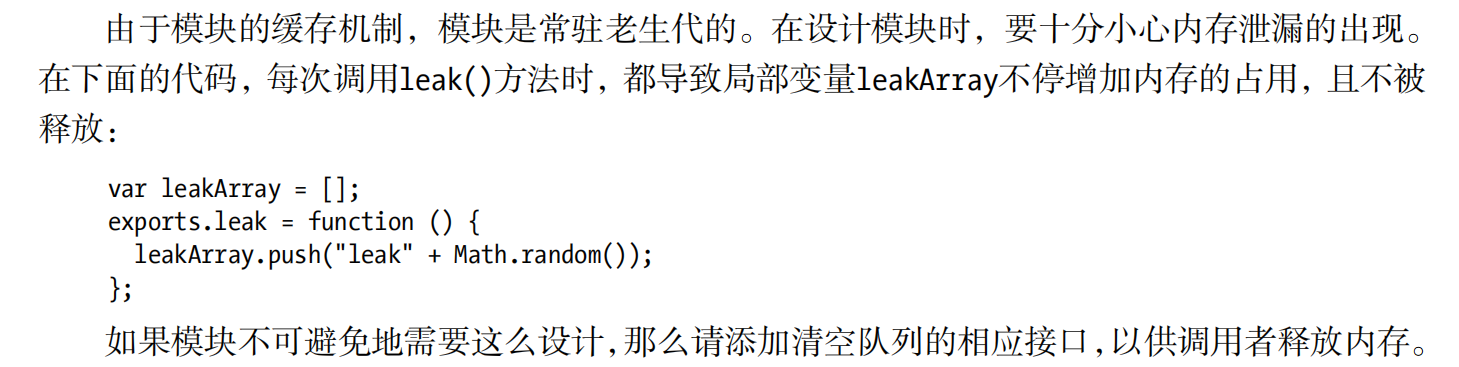


**因此老年代存在的大多是：全局变量关联着引用和闭包导致关联引用**

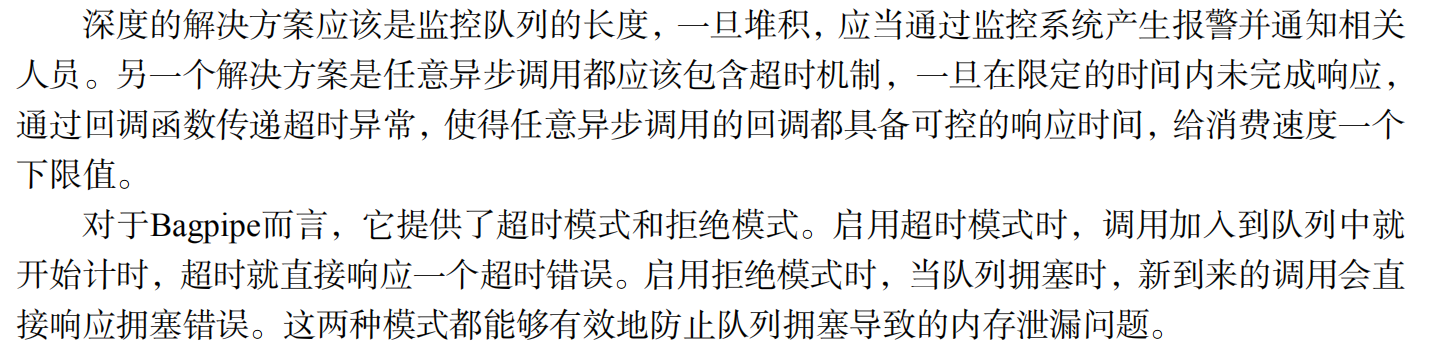
# 内存泄漏

因被回收的垃圾却没有被回收，变成了常驻老年代的对象。

1. 缓存无限制
2. 模块都是常驻老年代。所以要注意：

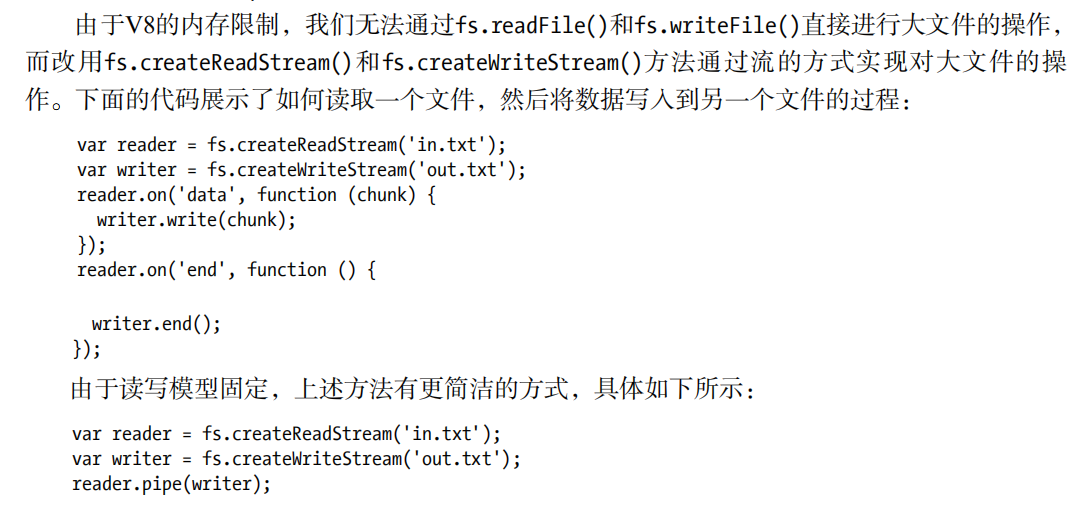


1. 队列任务堆积（Bagpipe）



# 大内存的应用

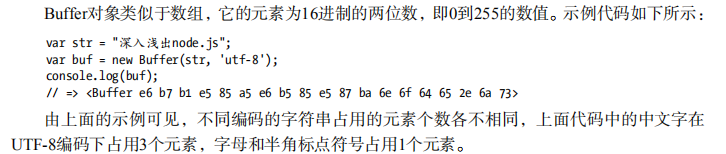
由于V8对内存的限制，我们无法使用fs.readFile() 对大文件进行读写。

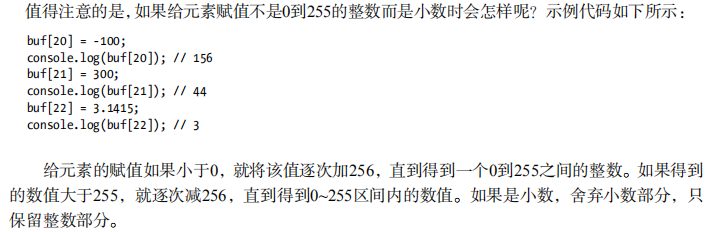


# 内存泄漏排查工具

具体略，等有实际场景

# Buffer





## Buffer内存分配

在c++层面申请，在javascript层面分配。具体看书

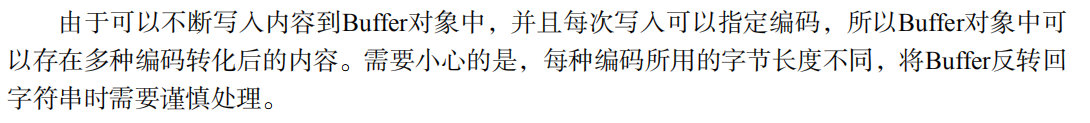
## Buffer的编码

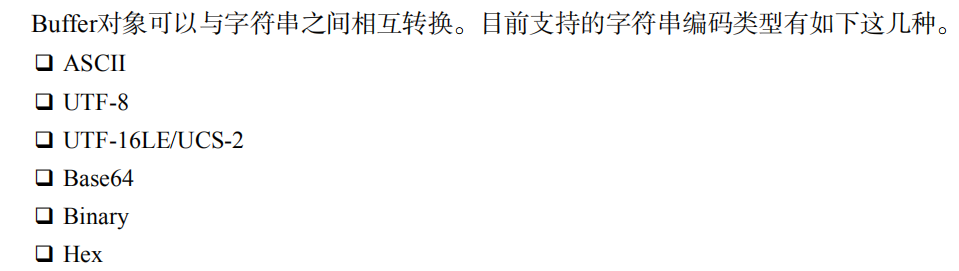
New buffer(str,[encoding])

Buf.write(str,[offset],[length],[encoding])

Buf.toString([encoding],[start],[end])

写和读都存在encoding：





Buffer截取问题，使用以下方法：

|  |
| --- |
| 使用的是chunks数组通过push进行保存，然后再完全接收后使用concat将chunks里面的所有buffer进行拼接，再将其一次性转换。  使用datas+=chunk 进行接收会出现问题，具体看书。 |

创建可读流时使用highWaterMark设置读取的长度，可以有效提高性能。减少读取次数