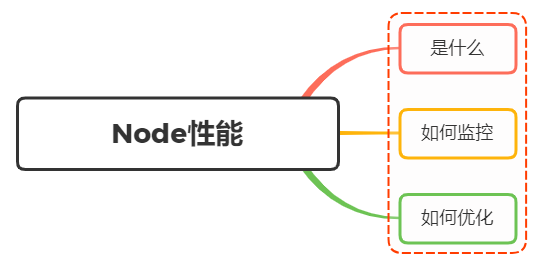
# 面试官：Node性能如何进行监控以及优化？



## 一、 是什么

Node作为一门服务端语言，性能方面尤为重要，其衡量指标一般有如下：

* CPU
* 内存
* I/O
* 网络

### CPU

主要分成了两部分：

* CPU负载：在某个时间段内，占用以及等待CPU的进程总数
* CPU使用率：CPU时间占用状况，等于 1 - 空闲CPU时间(idle time) / CPU总时间

这两个指标都是用来评估系统当前CPU的繁忙程度的量化指标

Node应用一般不会消耗很多的CPU，如果CPU占用率高，则表明应用存在很多同步操作，导致异步任务回调被阻塞

### 内存指标

内存是一个非常容易量化的指标。 内存占用率是评判一个系统的内存瓶颈的常见指标。 对于Node来说，内部内存堆栈的使用状态也是一个可以量化的指标

// /app/lib/memory.js  
const os = require('os');  
// 获取当前Node内存堆栈情况  
const { rss, heapUsed, heapTotal } = process.memoryUsage();  
// 获取系统空闲内存  
const sysFree = os.freemem();  
// 获取系统总内存  
const sysTotal = os.totalmem();  
  
module.exports = {  
 memory: () => {  
 return {  
 sys: 1 - sysFree / sysTotal, // 系统内存占用率  
 heap: heapUsed / headTotal, // Node堆内存占用率  
 node: rss / sysTotal, // Node占用系统内存的比例  
 }  
 }  
}

* rss：表示node进程占用的内存总量。
* heapTotal：表示堆内存的总量。
* heapUsed：实际堆内存的使用量。
* external ：外部程序的内存使用量，包含Node核心的C++程序的内存使用量

在Node中，一个进程的最大内存容量为1.5GB。因此我们需要减少内存泄露

### 磁盘 I/O

硬盘的IO 开销是非常昂贵的，硬盘 IO 花费的 CPU 时钟周期是内存的 164000 倍

内存 IO比磁盘IO 快非常多，所以使用内存缓存数据是有效的优化方法。常用的工具如 redis、memcached等

并不是所有数据都需要缓存，访问频率高，生成代价比较高的才考虑是否缓存，也就是说影响你性能瓶颈的考虑去缓存，并且而且缓存还有缓存雪崩、缓存穿透等问题要解决

## 二、如何监控

关于性能方面的监控，一般情况都需要借助工具来实现

这里采用Easy-Monitor 2.0，其是轻量级的 Node.js 项目内核性能监控 + 分析工具，在默认模式下，只需要在项目入口文件 require 一次，无需改动任何业务代码即可开启内核级别的性能监控分析

使用方法如下：

在你的项目入口文件中按照如下方式引入，当然请传入你的项目名称：

const easyMonitor = require('easy-monitor');  
easyMonitor('你的项目名称');

打开你的浏览器，访问 http://localhost:12333 ，即可看到进程界面

关于定制化开发、通用配置项以及如何动态更新配置项详见官方文档

## 三、如何优化

关于Node的性能优化的方式有：

* 使用最新版本Node.js
* 正确使用流 Stream
* 代码层面优化
* 内存管理优化

### 使用最新版本Node.js

每个版本的性能提升主要来自于两个方面：

* V8 的版本更新
* Node.js 内部代码的更新优化

### 正确使用流 Stream

在Node中，很多对象都实现了流，对于一个大文件可以通过流的形式发送，不需要将其完全读入内存

const http = require('http');  
const fs = require('fs');  
  
// bad  
http.createServer(function (req, res) {  
 fs.readFile(\_\_dirname + '/data.txt', function (err, data) {  
 res.end(data);  
 });  
});  
  
// good  
http.createServer(function (req, res) {  
 const stream = fs.createReadStream(\_\_dirname + '/data.txt');  
 stream.pipe(res);  
});

### 代码层面优化

合并查询，将多次查询合并一次，减少数据库的查询次数

// bad  
for user\_id in userIds   
 let account = user\_account.findOne(user\_id)  
  
// good  
const user\_account\_map = {} // 注意这个对象将会消耗大量内存。  
user\_account.find(user\_id in user\_ids).forEach(account){  
 user\_account\_map[account.user\_id] = account  
}  
for user\_id in userIds   
 var account = user\_account\_map[user\_id]

### 内存管理优化

在 V8 中，主要将内存分为新生代和老生代两代：

* 新生代：对象的存活时间较短。新生对象或只经过一次垃圾回收的对象
* 老生代：对象存活时间较长。经历过一次或多次垃圾回收的对象

若新生代内存空间不够，直接分配到老生代

通过减少内存占用，可以提高服务器的性能。如果有内存泄露，也会导致大量的对象存储到老生代中，服务器性能会大大降低

如下面情况：

const buffer = fs.readFileSync(\_\_dirname + '/source/index.htm');  
  
app.use(  
 mount('/', async (ctx) => {  
 ctx.status = 200;  
 ctx.type = 'html';  
 ctx.body = buffer;  
 leak.push(fs.readFileSync(\_\_dirname + '/source/index.htm'));  
 })  
);  
  
const leak = [];

leak的内存非常大，造成内存泄露，应当避免这样的操作，通过减少内存使用，是提高服务性能的手段之一

而节省内存最好的方式是使用池，其将频用、可复用对象存储起来，减少创建和销毁操作

例如有个图片请求接口，每次请求，都需要用到类。若每次都需要重新new这些类，并不是很合适，在大量请求时，频繁创建和销毁这些类，造成内存抖动

使用对象池的机制，对这种频繁需要创建和销毁的对象保存在一个对象池中。每次用到该对象时，就取对象池空闲的对象，并对它进行初始化操作，从而提高框架的性能

## 参考文献

* https://segmentfault.com/a/1190000039327565
* https://zhuanlan.zhihu.com/p/50055740
* https://segmentfault.com/a/1190000010231628