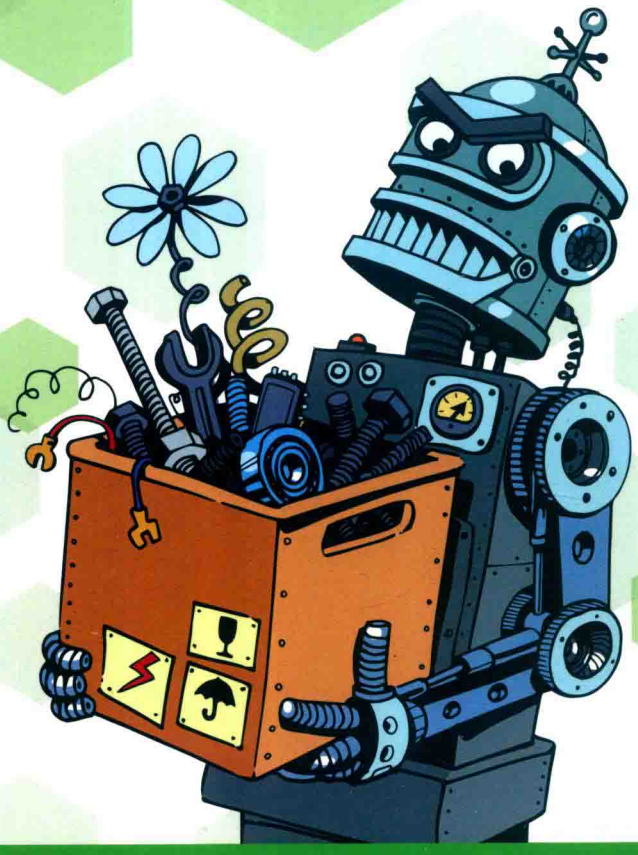
Broadview®
www.broadview.com.cn



Node.js 调试指南

赵坤
编著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Node.js 调试指南

赵坤
编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内容简介

本书从 CPU、内存、代码、工具、APM、日志、监控、应用这 8 个方面讲解如何调试 Node.js，大部分小节都会以一段经典的问题代码为例进行分析并给出解决方案。其中，第 1 章讲解 CPU 相关的知识，涉及各种 CPU 性能分析工具及火焰图的使用；第 2 章讲解内存相关的知识，例如 Core Dump 及如何分析 heapsnapshot 文件；第 3 章讲解代码相关的知识，例如如何从代码层面避免写出难以调试的代码，并涉及部分性能调优知识；第 4 章讲解工具相关的知识，涉及常用的 Node.js 调试工具和模块；第 5 章讲解 APM（Application Performance Management）相关的知识，例如两个不同的应用程序性能管理工具的使用；第 6 章讲解日志相关的知识，例如如何使用 Node.js 的 async_hooks 模块实现自动日志打点，并结合各种工具进行使用；第 7 章讲解监控相关的知识，例如如何使用 Telegraf + InfluxDB + Grafana 搭建一个完整的 Node.js 监控系统；第 8 章讲解应用相关的知识，给出了两个完整的 Node.js 应用程序的性能解决方案。

本书并不适合 Node.js 初学者，适合有一定 Node.js 开发经验的人阅读。笔者倾向于将本书定位成参考书，每一小节基本独立，如果遇到相关问题，则可以随时翻到相应的章节进行阅读。

图书在版编目（CIP）数据

Node.js 调试指南 / 赵坤编著 . —北京：电子工业出版社，2018.6
ISBN 978-7-121-34146-5

Ⅰ . ① N… Ⅱ . ①赵… Ⅲ . ① JAVA 语言 – 程序设计 Ⅳ . ① TP312.8

中国版本图书馆 CIP 数据核字（2018）第 088378 号

策划编辑：张国霞
责任编辑：徐津平
印　　刷：北京富诚彩色印刷有限公司
装　　订：北京富诚彩色印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787 × 980　1/16　印张：15.25　字数：320 千字
版　　次：2018 年 6 月第 1 版
印　　次：2018 年 6 月第 1 次印刷
印　　数：3000 册　定价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前言

笔者和同事在过去一年多主要把工作精力放在如何提高 Node.js 服务端的性能、稳定性和基础设施搭建上，随着公司业务量的快速增长，我们遇到了各种各样的挑战，也取得了不错的成绩：从起初啥都没有，到后来建立了比较完善的日志、监控和报警系统；从起初遇到问题不知道如何下手，到后来在遇到问题时能及时发现并定位问题。总之，付出得到了回报。

笔者曾将在这一段时间内遇到的调试、调优过程整理并记录成文章，发表在公司的知乎专栏上，本书就是在其基础上修改、补充和润色而成的，算是笔者对这几年在 Node.js 开发中进行调试的经验和思路的总结，笔者希望授人以鱼，亦能授人以渔。

本书概要

本书从 CPU、内存、代码、工具、APM、日志、监控、应用这 8 个方面讲解如何调试 Node.js，大部分小节都会以一段经典的问题代码为例进行分析并给出解决方案。

第 1 章讲解 CPU 相关的知识，涉及各种 CPU 性能分析工具及火焰图的使用。

第 2 章讲解内存相关的知识，例如 Core Dump 及如何分析 heapsnapshot 文件。

第 3 章讲解代码相关的知识，例如如何避免在代码层面写出难以调试的代码，并涉及部分性能调优知识。

第 4 章讲解工具相关的知识，涉及常用的 Node.js 调试工具和模块。

第 5 章讲解 APM（Application Performance Management）相关的知识，例如两个不同的应用程序性能管理工具的使用。

第 6 章讲解日志相关的知识，例如如何使用 Node.js 的 async_hooks 模块实现自动日

志打点，并结合各种工具进行使用。

第 7 章讲解监控相关的知识，例如如何使用 Telegraf + InfluxDB + Grafana 搭建一个完整的 Node.js 监控系统。

第 8 章讲解应用相关的知识，给出了两个完整的 Node.js 应用程序的性能解决方案。

本书定位

本书并不适合 Node.js 初学者，适合有一定 Node.js 开发经验的人阅读。笔者倾向于将本书定位成参考书，每一小节基本独立，如果遇到相关问题，则可以随时翻到相应的章节进行阅读。

开发环境

- MacOS 或 Linux（Ubuntu@16.04 64 位），Windows 用户请在使用虚拟机安装 Ubuntu 后进行操作。
- Node.js@8.9.4。

致谢

感谢石墨文档为笔者提供了良好的成长环境和技术氛围，感谢一起努力并解决问题的同事们，感谢张国霞编辑的耐心指导，感谢寸志、老雷、Yorkie、王政、杨海剑、黄一君在百忙之中抽出时间审阅本书并给出反馈。谢谢你们。

交流 & 勘误

扫描下方的二维码，便可与笔者交流并提交勘误，您的反馈及意见对笔者来说非常重要，再次感谢！

目录

第1章

CPU

Node.js 是单线程异步 I/O 模型，适合 I/O 密集型的场景，这也意味着 Node.js 并不适合 CPU 密集型计算的场景，因为 Node.js 不是收到一个请求就启动一个线程，假如有一个计算任务长时间地占用 CPU，整个应用就会“卡住”，无法处理其他请求。本章将讲解在 CPU 出现性能瓶颈时如何调试、发现及解决问题，并讲解相关的基础知识。

1.1 理解 perf 与火焰图（FlameGraph）

当程序出现性能瓶颈时，我们通常通过表象（比如在请求某个接口时 CPU 使用率飙涨）并结合代码去推测可能出问题的地方，却不知道问题是由什么引起的。如果有一个可视化的工具能直观地展现程序的性能瓶颈就好了，幸好 Brendan D. Gregg 发明了火焰图。

火焰图（Flame Graph）看起来就像一团跳动的火焰，因此得名，它可以将 CPU 的使用情况可视化，使我们直观地了解到程序的性能瓶颈。我们通常要结合操作系统的性能分析工具（Profiling Tracer）使用火焰图，常见的操作系统的性能分析工具如下。

- Linux：perf、eBPF、SystemTap 和 ktap。
- Solaris：illumos、FreeBSD 和 DTrace。
- Mac OS X：DTrace 和 Instruments。
- Windows：Xperf.exe。

1.1.1 perf

perf_events（简称 perf）是 Linux Kernal 自带的系统性能分析工具，能够进行函数级与指令级的热点查找。它基于事件采样原理，以性能事件为基础，支持针对处理器与操作系统相关的性能指标的性能剖析，常用于查找性能瓶颈及定位热点代码。

测试机器：

```
$ uname -a
Linux nswbmw-VirtualBox 4.10.0-28-generic #32~16.04.2-Ubuntu SMP Thu Jul 20
10:19:48 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
```

注 意

非 Linux 用户需要在通过虚拟机安装 Ubuntu 16.04 和 node@8.9.4 后再进行后面的操作。

安装 perf：

```
$ sudo apt install linux-tools-common
$ perf # 根据提示安装对应的内核版本的 tools，如下
$ sudo apt install linux-tools-4.10.0-28-generic linux-cloud-tools-4.10.0-28-
generic
```

创建测试目录 ~/test 和测试代码。

app.js：

```
const crypto = require('crypto')
const Paloma = require('paloma')
const app = new Paloma()
const users = {}

app.route({ method: 'GET', path: '/newUser', controller (ctx) {
  const username = ctx.query.username || 'test'
  const password = ctx.query.password || 'test'

  const salt = crypto.randomBytes(128).toString('base64')
  const hash = crypto.pbkdf2Sync(password, salt, 10000, 64, 'sha512').
toString('hex')

  users[username] = { salt, hash }

  ctx.status = 204
}})

app.route({ method: 'GET', path: '/auth', controller (ctx) {
  const username = ctx.query.username || 'test'
  const password = ctx.query.password || 'test'

  if (!users[username]) {
    ctx.throw(400)
  }
  const hash = crypto.pbkdf2Sync(password, users[username].salt, 10000, 64,
'sha512').toString('hex')

  if (users[username].hash === hash) {
    ctx.status = 204
  } else {
    ctx.throw(403)
  }
}})
```

```
app.listen(3000)
```

添加 --perf_basic_prof（或者 --perf-basic-prof）参数运行此程序，会对应生成一个 /tmp/perf-<PID>.map 文件。命令如下：

```
$ node --perf_basic_prof app.js &
[1] 3590
$ tail /tmp/perf-3590.map
51b87a7b93e 18 Function:~emitListeningNT net.js:1375
51b87a7b93e 18 LazyCompile:~emitListeningNT net.js:1375
51b87a7bad6 39 Function:~emitAfterScript async_hooks.js:443
51b87a7bad6 39 LazyCompile:~emitAfterScript async_hooks.js:443
51b87a7bcbe 77 Function:~tickDone internal/process/next_tick.js:88
51b87a7bcbe 77 LazyCompile:~tickDone internal/process/next_tick.js:88
51b87a7bf36 12 Function:~clear internal/process/next_tick.js:42
51b87a7bf36 12 LazyCompile:~clear internal/process/next_tick.js:42
51b87a7c126 b8 Function:~emitPendingUnhandledRejections internal/process/
promises.js:86
51b87a7c126 b8 LazyCompile:~emitPendingUnhandledRejections internal/process/
promises.js:86
```

map 文件中的三列依次为：16 进制的符号地址（Symbol Address）、大小（Size）和符号名（Symbol Names）。perf 会尝试查找 /tmp/perf-<PID>.map 文件，用来做符号转换，即把 16 进制的符号地址转换成人能读懂的符号名。

当然，在这里使用 --perf_basic_prof_only_functions 参数也可以，但笔者在尝试后发现生成的火焰图信息不全（不全的地方显示 [perf-<PID>.map]），所以使用了 --perf_basic_prof。但是，使用 --perf_basic_prof 有个缺点，就是会导致 map 文件一直增大，这是由于符号（symbols）地址的不断变换导致的，用 --perf_basic_prof_only_functions 可以缓解这个问题。关于如何取舍，还请读者自行尝试。

接下来 clone（克隆）用来生成火焰图的工具：

```
$ git clone http://github.com/brendangregg/FlameGraph ~/FlameGraph
```

我们先用 ab 压测：

```
$ curl "http://localhost:3000/newUser?username=admin&password=123456"
$ ab -k -c 10 -n 2000 "http://localhost:3000/auth?username=admin&passwo
```

```
rd=123456"
```

重新打开另一个终端，在 ab 开始压测后立即运行：

```
$ sudo perf record -F 99 -p 3590 -g -- sleep 30
$ sudo chown root /tmp/perf-3590.map
$ sudo perf script > perf.stacks
$ ~/FlameGraph/stackcollapse-perf.pl --kernel < ~/perf.stacks | ~/FlameGraph/
flamegraph.pl --color=js --hash> ~/flamegraph.svg
```

注 意

第 1 次生成的 svg 可能不太准确，最好重复几次以上步骤，使用第 2 次及以后生成的 flamegraph.svg。

有几点需要解释一下。

（1）关于 perf record：

- -F 指定采样频率为 99Hz（即每秒 99 次，如果 99 次都返回同一个函数名，就说明 CPU 在这一秒钟都在执行同一个函数，可能存在性能问题）；
- -p 指定进程的 pid；
- -g 启用 call-graph 记录；
- sleep 30 指定记录 30s。

（2）sudo chown root /tmp/perf-3009.map 用于将 map 文件更改为 root 权限，否则会报如下错误：

```
 > File /tmp/perf-PID.map not owned by current user or root, ignoring it
(use -f to override). > Failed to open /tmp/perf-PID.map, continuing without
symbols
```

（3）perf record 会将记录的信息保存到当前执行目录的 perf.data 文件中，将使用 perf script 读取的 perf.data 的 trace 信息写入 perf.stacks。

（4）--color=js 指定生成针对 JavaScript 配色的 svg，其中，green 代表 JavaScript，blue 代表 Builtin，yellow 代表 C++，red 代表 System。

ab 压测用了 30s 左右，用浏览器打开 flamegraph.svg，截取关键的部分，如图 1-1 所示。

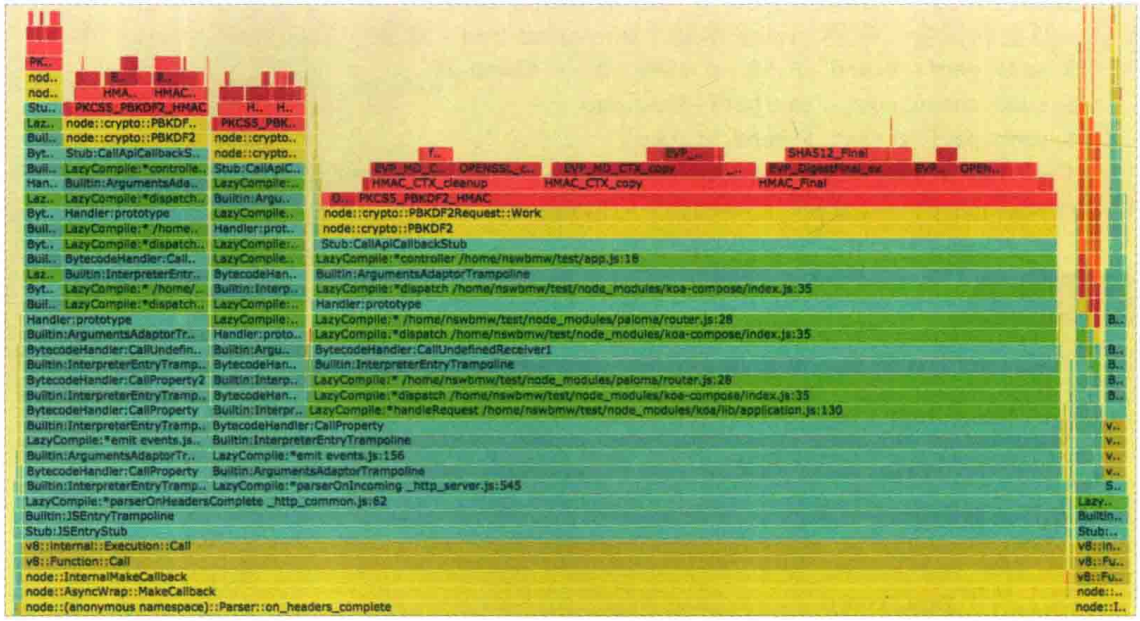


图 1-1

1.1.2 火焰图

火焰图的含义如下。

- 每一个小块都代表一个函数在栈中的位置（即一个栈帧）。
- *Y* 轴代表栈的深度（栈上的帧数），顶端的小块显示了占据 CPU 的函数。每个小块的下面是它的祖先（即父函数）。
- *X* 轴代表总的样例群体。它不像绝大多数图表那样从左到右表示时间的流逝，其左右顺序没有特殊含义，仅仅按照字母表的顺序排列。
- 小块的宽度代表 CPU 的使用时间，或者说相对于父函数而言使用 CPU 的比例（基于所有样例），越宽则代表占用 CPU 的时间越长，或者使用 CPU 很频繁。
- 如果采取多线程并发运行取样，则取样数量会超过运行时间。