



Node.js 实战

使用 Egg.js + Vue.js + Docker
构建渐进式、可持续集成与交付应用

yugo © 著

内容简介

本书将帮助 Node.js 在 Web 开发方面的应用。以一个类 Dribbble 图片画廊应用为例，从内容数据库、前端、后端、部署、运维等方面，详细讲解 Node.js 框架核心、中间层、上层服务的构建、OAuth 登录、JWT 登录、负载均衡、异步、以及使用 TypeScript 和 Vue.js 实现前后端同构的前端界面，解决 SEO 问题，部署与持续集成。此外，还介绍了 Docker 实现 DevOps。最后还介绍了与 Node.js 相关的技术栈与工具链之后的最佳实践的推荐。

前言

本书适合从事 Web 开发的 Node.js 感兴趣的读者阅读。

本书内容全面、通俗易懂，可作为从事 Web 开发的入门教材，也可作为从事 Web 开发的进阶教材。

为什么要写这本书

前端生态圈的繁荣离不开 Node.js。Node.js 在制作工具方面的优势，在开发 Web 应用、构建服务端应用、构建后端应用等方面，都有广泛的应用。Node.js 的流行，使得 JavaScript 语言在服务器端的应用变得广泛。本书旨在帮助读者了解 Node.js 的生态，掌握 Node.js 的开发技能，构建渐进式、可持续集成与交付应用。

Node.js 实战

使用 Egg.js+Vue.js+Docker 构建渐进式、可持续集成与交付应用

yugo◎著

本书适合的对象

• 有 JavaScript 基础的开发者

• 需要体验完整开发流程

• 想学习 Node.js Web 开发的读者

北京人民邮电出版社
北京市西城区宣武门内大街 27 号
邮编 100045
电话 010-51097100
网址 www.pup.com.cn
ISBN 7-113-17317-1
定价 89.00 元

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书讲解 Node.js 在 Web 开发方面的实际应用，以一个类 Dribbble 图片画廊应用为实例，内容包括底层的 Koa.js/Egg.js 框架核心与实现原理，上层服务的构建、OAuth 服务、JWT 登录认证服务、前后端分离架构，以及使用 TypeScript 和 Vue.js 实现前后端同构的前端界面，解决 SEO 问题，部署与持续集成，使用时下流行的 Docker 实现 DevOps。最后还介绍了压力测试与上线之后的数据收集的注意事项，可解决日常企业需求。

本书适合从事 Web 开发并对 Node.js 感兴趣的读者阅读。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目 (CIP) 数据

Node.js 实战：使用 Egg.js+Vue.js+Docker 构建渐进式、可持续集成与交付应用 / yugo 著. —北京：电子工业出版社，2018.9
ISBN 978-7-121-34912-6

I. ①N… II. ①y… III. ①JAVA 语言—程序设计 IV. ①TP312.8

中国版本图书馆 CIP 数据核字 (2018) 第 187921 号

责任编辑：陈晓猛

印 刷：三河市双峰印刷装订有限公司

装 订：三河市双峰印刷装订有限公司

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：29.75

字数：571.2 千字

版 次：2018 年 9 月第 1 版

印 次：2018 年 9 月第 1 次印刷

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 zltts@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

为什么要写这本书

前端生态圈的繁荣离不开 Node.js。Node.js 在制作工具方面的表现极其优秀，在开发 Web 方面也有很多历史积累。Node.js 领域的图书很多，比如侧重 Node.js 语法、核心本身，或者侧重调试，而对于 Web 开发，提及 OAuth、JWT 原理的并不多，涉及前端范畴的在线支付、持续集成、Docker 等内容也较少。

笔者比较喜欢体验各种语言，在大学的时候学习和体验了各种语言，包括 C、C++、C#、Java、Python、Ruby、PHP 等，在笔者的网站 nodelover.me 你还会发现有 Go、Rust 的免费视频教程。笔者把大部分精力都花在了 JavaScript 上面，后来才有了这本书。

Node.js 底层还有许多内容笔者也没有弄懂，不过 80% 的业务场景，只需要 20% 的技术能力就可以解决。笔者跟读者一样，都是一个学习者。笔者希望更多的人学习 Node.js、使用 Node.js，使它更加强大。

本书适合的对象

- 有 JavaScript 基础的读者；
- 想要体验完整开发流程的读者；
- 想要精通 Node.js Web 开发的读者；
- 对 Koa.js 和 Egg.js 实现原理感兴趣的读者。

本书也可以作为 Node.js 的入门教程，但是需要你有一定的自学能力，对于一些基础的知识，笔者都会给出视频链接，读者可以自行学习。

3.2.2 使用 egg-nsg-flash 72

3.2.3 使用 egg-y-validator 73

本书结构

- 第 1 章：主要叙述了 Node.js 的历史，以及为什么要使用它。
- 第 2 章：讲解 JavaScript 的异步、函数式编程、Koa.js 实现原理，以及 Egg.js 是如何在 Koa.js 上面进行扩展的、Egg.js 是怎样的架构、如何开发出一个 Egg.js 插件并发布到 npmjs。
- 第 3 章：使用 Egg.js 对后端服务进行开发，设计数据库表，构建模型关系映射，建立模型之间的关系。构建安全的 API，使用 JWT 构建登录，使用 OAuth 给第三方开发者开发 API。
- 第 4 章：通过 Vue.js 构建一个简易的后台，通过百行代码实现从后台读取数据库关系，使用 Vue.js 动态地生成对应模型的表单，自动增删改查。
- 第 5 章：使用 TypeScript 与 Vue.js 搭建 SSR 服务端渲染环境，构建友好的 SEO，开发前端显示界面。
- 第 6 章：使用 Docker 部署我们的应用，讲解如何编写 Dockerfile、docker-compose.yml，如何实现通过修改一行代码提交修改，然后自动部署服务。
- 第 7 章：性能分析与优化，包括服务器性能优化、用户追踪、前端性能优化。

勘误与支持

由于部分 npm 组件 API 的变动与 Node.js 的发展，以及笔者的疏忽、水平有限，书中总会有一些不足之处，还望读者批评指正，可以通过以下方式与笔者联系。

GitHub issues: <https://github.com/MiYogurt/nodejs-shizhan>

QQ 群: 325568224

致谢

首先要感谢的是曾经努力的自己，对他说一句“你真棒”。其次感谢父母对我的支持，假如没有父母的支持，可能我就不会有那么多的精力来做这件事情。感谢陈晓猛编辑的耐心指导、审稿、修改，在他的修改下，使得本书有更好的阅读体验。最后感谢的是 Node.js 社区的各位开发者，我们都是站在巨人的肩膀上，感谢巨人们。

目录

第 1 章	Node.js 的优势	1
1.1	为什么是 JavaScript 语言	1
1.2	为什么经常说 Node.js 不适合大型应用	3
第 2 章	Egg.js 框架核心原理与实现	6
2.1	异步基础	6
2.2	Koa.js 基础知识	15
2.2.1	Koa.js 中间件核心代码	16
2.2.2	Koa.js 插件	18
2.3	Egg.js 基础知识	21
2.3.1	实现 egg-core	22
2.3.2	实现 egg-init	26
2.3.3	实现 egg-cluster	30
2.4	Egg.js 插件	33
2.4.1	egg-socket.io	33
2.4.2	原理解读	39
2.5	制作一个 Egg.js 插件	43
第 3 章	构建后端 API 服务	52
3.1	安装相关组件	52
3.2	发布一个插件	59
3.2.1	创建 Flash 插件	59
3.2.2	使用 egg-msg-flash	72
3.2.3	使用 egg-y-validator	73

3.3	规范化	73
3.3.1	添加新的 scripts 支持 ESLint 自修复	74
3.3.2	添加插件支持	74
3.3.3	prettier 格式化工具	76
3.3.4	同步代码编辑器配置	76
3.4	第一个 JSON 请求	77
3.4.1	给全局添加一些方法	77
3.4.2	全局化一些东西	84
3.4.3	自动路由	86
3.4.4	创建 PostMan 测试	88
3.5	注册服务	91
3.5.1	Invitation 模型	91
3.5.2	注释	93
3.5.3	User 模型	96
3.5.4	修改控制器	97
3.5.5	添加验证逻辑	98
3.5.6	帮助方法	99
3.5.7	User 服务	101
3.5.8	PostMan 测试	103
3.6	登录服务	104
3.7	邮件与调试	115
3.7.1	理解发送邮件的原理	115
3.7.2	安装邮件插件	115
3.7.3	环境与调试	116
3.7.4	全局调试	118
3.7.5	VSCode 全局调试	121
3.7.6	发送验证邮件	122
3.7.7	添加逻辑	125
3.7.8	验证	135
3.8	构建 RESTful API	137
3.8.1	什么是 RESTful API	137
3.8.2	创建 RESTController 基础类便于继承	138

3.8.3	测试 Images RESTful API	141
3.8.4	构建后台的 REST 路由	143
3.8.5	构建控制器	145
3.8.6	测试后台路由	148
3.8.7	关于验证	149
3.9	安全地开放 API	151
3.10	实现 OAuth 接口	158
3.10.1	实现授权码官方文档所要求的接口	158
3.10.2	实现刷新验证码接口	167
3.10.3	实现 authenticate 所需接口	169
3.11	完善 OAuth 与测试	170
3.11.1	发放 Token	170
3.11.2	新建客户端项目	172
3.11.3	测试 OAuth	173
3.12	支付宝支付	176
3.12.1	什么是非对称加密	176
3.12.2	注册支付宝	176
3.12.3	生成非对称秘钥	176
3.12.4	实现	177
3.12.5	添加路由	180
3.12.6	内网穿透	180
3.12.7	测试	182
3.13	社会化登录	183
第 4 章	构建后台管理页面	189
4.1	后端开发	189
4.1.1	安装 VSCode 插件	189
4.1.2	获取脚手架	189
4.1.3	安装依赖	190
4.1.4	修改代码	190
4.1.5	跨域请求	191
4.1.6	修改后端代码支持跨域	192
4.1.7	在前端添加存储	198

4.2	模型列表	200
4.3	添加数据	209
4.4	修改逻辑	220
第 5 章 前端界面设计与实现		228
5.1	搭建前端开发环境	228
5.1.1	开始	228
5.1.2	创建 Header 头部	229
5.1.3	将变量提取出来	234
5.1.4	添加路径重写	235
5.2	AppFooter 组件	237
5.2.1	做一些配置	237
5.2.2	创建 src/components/layouts/AppFooter.vue	238
5.2.3	网络识别信息	247
5.2.4	修改一下全局样式	247
5.2.5	查看页面	248
5.2.6	提升编译速度	248
5.3	首页	249
5.4	替换成为真实数据	269
5.4.1	完成后端 Image API	269
5.4.2	修改首页的代码	271
5.4.3	添加 API 逻辑	276
5.4.4	效果	278
5.5	图片详情页	278
5.5.1	创建路由	279
5.5.2	安装依赖	279
5.5.3	创建视图	279
5.5.4	添加插件	285
5.5.5	创建评论组件	286
5.5.6	测试	290
5.5.7	关于服务端访问 DOM	290
5.6	注册页面	294

5.6.1	注册路由	294
5.6.2	新建 signup.vue 页面	295
5.6.3	增强错误提示	299
5.7	登录页面	299
5.8	完善详情与评论	310
5.9	个人中心	321
5.10	创建图片	336
5.10.1	创建又拍云存储	336
5.10.2	添加后端 API	338
5.10.3	前端界面	340
5.10.4	测试	348
5.11	团队	349
5.11.1	功能是如何工作的	350
5.11.2	数据库	350
5.11.3	后端	356
5.11.4	前端	363
5.11.5	测试	372
第 6 章	部署与运维	374
6.1	认识 Docker	374
6.1.1	解决了什么问题	374
6.1.2	使用 Docker 的流程	375
6.1.3	安装 Docker	378
6.1.4	使用加速器	378
6.1.5	下载一个基础镜像	379
6.1.6	hello world	379
6.2	手动构建镜像	380
6.3	编写 Dockerfile 文件	384
6.4	Docker Compose	387
6.4.1	安装 docker-compose	387
6.4.2	命令行接口	388
6.4.3	Egg.js 简单实例	389
6.4.4	增加服务	391

6.5	集群	396
6.5.1	Docker 集群	396
6.5.2	集群初始化	396
6.5.3	实例	397
6.6	持续部署	400
6.6.1	部署主机免密码登录	400
6.6.2	客户端钩子	401
6.6.3	使用服务端钩子进行部署	403
6.6.4	使用 shipit	404
6.6.5	使用 Ansible 部署	406
6.7	持续集成	409
6.8	Kubernetes 集群	423
6.8.1	简单使用	423
6.8.2	如何创建应用	425
6.8.3	命令行管理	430
6.8.4	通过 UI 创建应用	433
6.8.5	添加持续集成	439
6.8.6	固定 IP 地址	441
6.8.7	部署前端	442
第 7 章	性能分析与优化	448
7.1	服务器性能分析与测试	448
7.2	用户追踪	458
7.2.1	百度分析	458
7.2.2	Google 分析	460
7.2.3	其他付费服务	461
7.3	前端性能分析与优化	461
7.3.1	lighthouse	461
7.3.2	sonarwhal	462
7.3.3	图片压缩	464
7.3.4	错误上报	465
7.3.5	接收用户反馈	466

1 chapter

第 1 章 Node.js 的优势

1.1 为什么是 JavaScript 语言

JavaScript 拥有低成本、非常流行和运用范围广三大优势。

1. 低成本

一个网站包括前端界面和后端数据存储，无论选择何种语言来实现后端的逻辑，前端的实现都离不开 HTML、CSS 和 JavaScript 这几种技术。

也就是说，这几种技术是必修课，假如后端的逻辑也可以使用 JavaScript 来写，就可以减少学习一门后端语言的成本。所以只要安装了 Node.js 运行环境，就可以在操作系统中运行 JavaScript，读者只需把 Node.js 当作运行 JavaScript 的一个软件即可。

其实这只是减少成本的一个方面，而且只是表面上减少学习一门语言的成本。想要写好后端的逻辑，一些基础知识同样是必修课，这些学习成本是无法减免的。

另一方面的低成本其实是在包(Package)上面，这里的包指的是其他开发者开发好的代码，这个代码可以提供一些好用的函数、功能等，然后放在 NPM (node package manager) 上，供任何人下载使用。在后面的章节里，笔者会使用大量来自 NPM 优秀的包来提高开发效率。

2. 非常流行

打开 Photoshop CC 的安装目录，会发现 Node.js 的运行环境，即 node 执行程序。在 Linux/UNIX 操作系统中它通常是一个叫 node 的二进制执行程序，在 Windows 中，它通常被称为 node.exe 的可执行程序。

使用 Node.js 的公司并非只有 Adobe 公司，如阿里巴巴、腾讯、Facebook、Google、百度等公司都在使用。

基本上只要跟 Web 有关的公司都会直接或间接地用到 Node.js，Node.js 通常还被用来作为前端的构建工具的一部分——用来压缩代码、减少文件体积。在压缩过程中，避免不了要读取文件内容，这就会用到 Node.js 提供的 fs（File System）文件系统模块。

另外一方面则体现在 Node.js package 的数量上面，从 npmjs.com 上面的数据来看，开源的 package 共计 475 000 个。目前笔者看到的当前 package 周下载量为 3 255 222 628 次，每个月超过 7 百万的开发者使用 npmjs.com 来寻找和分享 package。

3. 运用范围广

自从 Node.js 问世，JavaScript 的生态越来越完善，JavaScript 能做的事情越来越多。

目前 JavaScript 的主战场还是在浏览器端，虽然通过一些框架，例如，Facebook 的 React Native、Apache 的 Weex 等可以使用 JavaScript 编写移动端框架，但是相比较 Swift、Object-C 与 Java、Kotlin 所写的原生应用还是有一定差距的。例如，部分性能问题（需要自己调优）、只能调用封装好的组件（用原生语言封装好供 JavaScript 调用，对于特殊需求，官方并不一定都能满足）等。这些框架只是提供一种解决方案而已，没有任何一种解决方案是完美的。

所以有的初学者总是关注负面，聚焦于性能问题，而忽略了它的优点，其实笔者认为可能初学者都被自己误导了。首先要确认的一点是，很多有经验的人通常工作在一些大型企业里面，而这类企业的用户量其实是非常多的，那么在非常多用户量的情况下，提升一点点性能确实能节约非常多的成本，毕竟积少成多，所以他们才会特别关注性能，而初学者在不明确上下文的情况下，就很容易被带到“沟里”去。

其实在初创公司里最不用关心的就是性能问题，除了代码编写（如内存泄露）导致的问题，只要是合理的架构，就非常容易横向扩展。而且就算是框架的问题也是可以解决的，腾讯大量地将 React Native 应用到 QQ 中，而淘宝则在“双 11”活动中大量实践了 Weex，依然没有出现性能问题。

而桌面端也有 Electron 框架，GitHub 开源的代码编辑器 Atom 其实就是基于 Electron 框架的，后来由于性能问题，部分组件用 C++ 重写了，目前来说界面还是用 HTML、CSS、JavaScript 写的。而开发物联网应用，则有 iot.js，它是三星开发的嵌入式 JavaScript 执行环境。

而服务端，本书用到的 Egg.js 就是一个非常好的例子，对于 Egg.js 的更多知识，在后续的章节都将一一讲解。其实同类型的解决方案还有很多，这里只是抛砖引玉地阐述 JavaScript 在某一方面是有解决方案的。

从上面几点可以看出，JavaScript 是非常适合个人独立开发者、初创公司使用的一门语言。

1.2 为什么经常说 Node.js 不适合大型应用

首先要明确什么是大型应用，其实这是仁者见仁、智者见智的问题，并且它是一个哲学问题，不是一个技术问题。假如有人问你，一个可以进行线上销售的网站，比如优衣库，大不大？你可能会说大，因为这与你平常所见的博客、企业官网等逻辑相比较确实复杂很多。或者说小，那么说明你开发过比它还复杂的系统。那么相比较淘宝而言呢？大和小的对比是要有参照物的。

1. 应用的组成

一个完备的 Web 应用可能只由一门语言或者一种技术构成吗？不可能。因为一个完备的 Web 应用其实是多门技术的综合体，解决某个问题有非常多的解决方案，比如后端的逻辑解决方案就非常多，Java、PHP、Python、Ruby 等都可以。

简单地概述，应用的组成内容可能包括：

- Web 界面显示逻辑；
- 后端业务逻辑；
- 缓存；
- 数据库；
- 消息队列。

其实还可以加入日志分析、数据分析等，只是上面几个最广为人知而已。

2. 应用的种类

- I/O 密集型；
- CPU 密集型。

就常见的互联网产品而言，它的瓶颈并非在后端业务的逻辑上，而是在 I/O 上，即返回给用户看的数据的读入与输出。相对于应用程序而言，读入指的是从数据库里获取数据，而输出指的是将这些数据经过一定的处理输出到用户的浏览器，那么这就是 I/O 密集型。

而 CPU 密集型是指做频繁计算任务的应用，Node.js 在这方面确实是短板。

3. 应用服务的过程

如图 1-1 所示，用户通过浏览器发送请求，由网卡接收 TCP 连接，通知内核，内核再去调用相对应的服务端程序。

Request 请求过程



图 1-1

Response 返回过程

如图 1-2 所示，Web 应用要返回数据，首先要获取数据，通过内核调用磁盘的驱动程序，把数据读入缓存，这样就可以在 Web 应用程序中获取数据并进行数据处理，最终调用内核，将数据通过网卡发送给客户端。

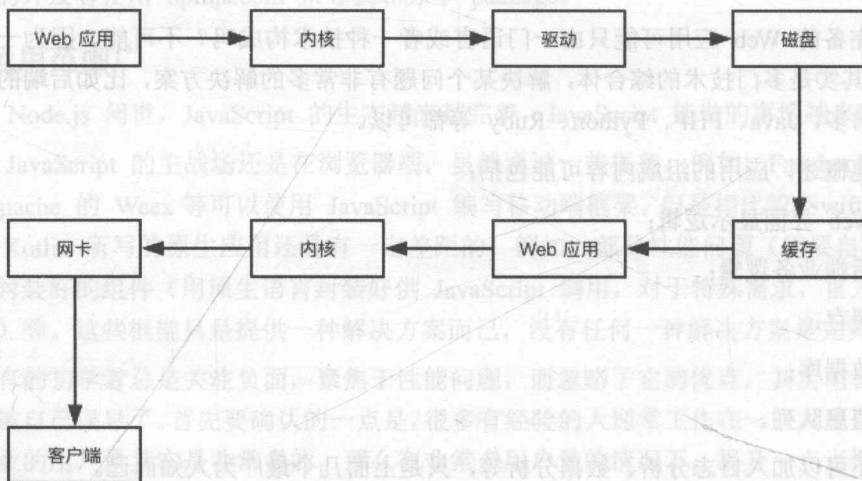


图 1-2

4. 应用的瓶颈

通常 I/O 密集型的瓶颈会在磁盘的读写上，所以在购买云服务器的时候可以购买 SSD 的磁盘来提升性能，一般数据库软件的数据都是存储在文件上面的。首先考虑添加内存型缓存来解决这个瓶颈，缓存经常访问的数据，看能否解决当前场景的问题，比如使用 Redis。其次才考虑搭建或扩充数据库集群来提高并发。

而 CPU 密集型的应用瓶颈则在 CPU 上，只能增加 CPU 处理核心来解决瓶颈。

5. 分布式应用

大型的普通应用与分布式应用其实是不同的概念。读者可以把分布式应用简单地理解为一个团队，每一个成员都是一个节点，一个大的项目要让成员合作完成，那么成员与成员之间就存在一些沟通成本，甚至有的成员与成员之间勾心斗角，说话阳奉阴违、推脱责任，也有可能成员生病在家休养，无法工作，等等。在面对这些问题的时候，Node.js 的优势并不能很好地呈现出来（并非不可以做，只是没有完善的基础设施）。

分布式的真正定义是，在多台不同的服务器中部署不同的服务模块，以进程为基本单位，派发到服务器上，通过远程调用（RPC）通信并协同工作，最终对外提供服务。

相比较 Node.js 目前的分布式基础设施，Go 语言的基础设施则完善多了，特别是在 Docker 这个项目上，充分证明了 Go 语言的优势，这也是为什么 Node.js 社区“大牛”TJ Holowaychuk 转向 Go 语言，因为他要开发分布式应用。

其实没必要过分地关心分布式的问题，毕竟 JavaScript 最初只是一个运行在浏览器端的脚本语言而已，JavaScript 不是万能的，为什么一定要把它用在操作系统级别的开发上呢？寻找一个更合适的语言不是更好吗？就像此刻我们选择 JavaScript 构建 Web 应用一样。

6. 多进程的 Node.js

了解了以上的一些知识点，现在读者应该知道，Node.js 跟大型应用关系不大。大多数学习 Node.js 的开发者是前端开发者，所以对后端的基础知识并不了解，在网络上搜寻一些资料的时候发现 Node.js 只能利用单核，而又听说 TJ Holowaychuk 转向 Go 的阵营，所以有的开发者就产生了 Node.js 不适合开发大型应用的疑问。

Node.js 只能利用单核的问题已经被解决了，后面使用的 Egg.js 框架中的 Egg-Cluster 模块就利用多进程非常好地解决了这个问题。

2 chapter

第 2 章

Egg.js 框架核心原理 与实现

2.1 异步基础

本节将介绍什么是异步，并阐述 Node.js 中异步的历史，以及如何进行异步编程。笔者也录制过一些关于异步的教学视频，在 <https://nodelover.me/course/js-async> 上可以免费获取。

1. 什么是异步

我们用一个故事来简单地概述异步。小雷想要给孩子买奶粉，所以去了经常买奶粉的地方，发现店前的公告板上写着本周日不上班。小雷不可能一直等到星期一奶粉店开门，所以小雷只好去其他地方给孩子买个尿布然后回家，等到星期一再来看看。奶粉店关门就是一个耗时任务，让买奶粉这件事没办法立刻完成，只有等待，在奶粉店面前死等就是同步。而且既然出来了，通常还会考虑家里还缺什么，比如说尿布不多了，所以就去其他的地方买个尿布再回家。这里放下买奶粉这件暂时无法完成的事情，而去做可以立即完成的事情，比如买尿布，这就是异步。

2. 回调实现异步

实现异步的本质是回调，我们从 Node.js 的 API 来看一下异步与同步的区别，让读者有一个直观的感受，代码如下：

```
const fs = require("fs");
```