

基于v7.6.0，详细介绍Node.js各种新特性，引导读者紧跟技术潮流

- ❖ 从零开始学习开发完整可用的项目，掌握语言细节并获得实际经验
- ❖ 以实际应用为背景，阐述Node.js在实际项目中的定位和潜在的陷阱
- ❖ 适合没有Node.js经验的读者，旨在快速上手到独立开发小型应用



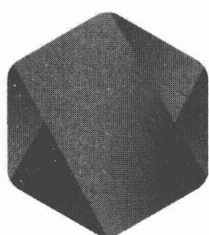
基于 Chrome V8 引擎的 JavaScript 运行环境

新时期的 Node.js入门

李锴 著

清华大学出版社

Web前端技术丛书



新时期的 Node.js入门

李锴 著



清华大学出版社
北京

内 容 简 介

Node.js 是一门开源的、为 Web 而生的语言，具有高并发、异步等特点，并且拥有一个十分活跃的开发者社区。与 Ruby、Python 等语言相比，Node.js 更年轻、更易于没有经验的人上手使用，因此很快在世界各地的开发者中收获了一大批拥趸。在国内，Node.js 在许多企业中获得了广泛应用，并取得了一系列的应用成果。然而，随着技术的不断更新、ECMAScript2015 等新标准纷纷确定，现有的中文书籍就变得有些过时。本书立足于新的技术潮流，介绍了一系列全新的语言特性和标准，以便让读者在学习基础知识的同时紧跟新技术的发展。

本书分为 8 章 6 个附录，讲解了 Node.js 的各种基础特性，使读者快速入门，同时结合语言最新的发展趋势，让读者能够紧跟技术潮流。本书围绕 Node.js 在 Web 站点开发和爬虫系统中的应用展开，对 Node.js 在大型项目中的定位与应用做了详细的说明。

本书可用于 Node.js 入门，适合未接触过 Node 的读者以及在校的学生阅读，也适合作为高等院校和培训学校相关专业的师生教学参考。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目 (CIP) 数据

新时期的 Node.js 入门 / 李锴著. —北京：清华大学出版社，2018

(Web 前端技术丛书)

ISBN 978-7-302-48780-7

I. ①新… II. ①李… III. ①JAVA 语言—程序设计 IV. ①TP312.8

中国版本图书馆 CIP 数据核字 (2017) 第 272827 号

责任编辑：夏毓彦

封面设计：王 翔

责任校对：闫秀华

责任印制：刘祎淼

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>，<http://www.wqbook.com>

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175

邮 购：010-62786544

投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn

质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市铭诚印务有限公司

经 销：全国新华书店

开 本：190mm×260mm

印 张：18.5

字 数：474 千字

版 次：2018 年 1 月第 1 版

印 次：2018 年 1 月第 1 次印刷

印 数：1~3000

定 价：49.00 元

产品编号：075003-01

前言

国内 Node 的开发者很多都读过朴灵写的《深入浅出 Node.js》(以下简称《深入浅出》)一书,笔者也不例外,笔者在 2014 年初第一次接触 Node,最初读的几本书就包含了这本《深入浅出》,该书出版于 2013 年 12 月,距今已经差不多有 4 年的时间了。

对于一门高速发展的语言来说,4 年算得上很长的时间了。4 年前 Node 的版本号还在 0.10.x,而时至今日,已经迎来 v8.0.0 的最新版本了。

Node 项目始于 2009 年,2013 年恰好处在当今(2017 年)和 2009 年的中间节点,一门语言在诞生之初的发展总是最快的,到了现在,Node 逐渐地变得稳定下来。

但即使这样,这 4 年中也发生了不少大事件:Node 从分裂又走向了统一,ES2015 标准的推出等。

那么 4 年后的今天,Node 有了哪些改变呢?

一方面,基本的概念几乎没有改变,底层的 libuv 和事件循环还是原来的样子,主要模块的 API 也没什么大的变化。

另一方面,变动最多的大概是语法了,ECMAScript 沉寂数年之后,终于推出了重量级的新版本 ES2015,并且计划每年发布一个新版本。

换个角度来说,如果现在有一份使用 Node 最新版本写的代码,拿给还在使用 0.10 的开发者看,最大可能是对里面各种奇怪的关键字和语法感到疑惑。这本身就说明了 Node 发生了如此大的变动。

在笔者看来,Node 的发展进入了平台期,这意味着在一段时间内,Node 将维持现有的模样,或许会增加或修改一些语法,底层的 V8 也可能做一些大幅度的改进,但代码的基本结构不会有大的变动。

Node 的发展大致分为几个阶段:

- 第一个阶段:从诞生到分裂,大致 5 年的时间。
- 第二个阶段:从与 io.js 合并到 ES2015 标准正式落地,只有不到半年的时间。
- 第三个阶段:从 v6.0.0 全面支持 ES2015 至今,Node 较大的更新都是围绕着新的 ECMA201x 标准展开的。

所有的新与旧都是相对的,虽然在目前来看,本书记述的内容还算是比较新的,无论是 ES201x 的使用,还是 Koa2 框架的介绍,都属于同类书中较少涉及的领域,但要是再过几年的时间,本书的内容也会变得过时。

本书定位

关于本书的定位，笔者着实费了很大的脑筋，出版社老师建议我写一本入门书，当我知道的时候其实内心是很犹豫的：一方面，Node 的入门书籍市面上已经很多了，随便翻开一本，不论是里面的内容还是章节编排都大同小异；另一方面，入门书反而是最难写出水平的，因为作者们大多已经有了相关领域的编程经验，那样就很难站在入门者的角度来进行写作。

经过考虑之后，我决定写一本《新时期的 Node.js 入门》，一方面能够让本书立足于最新的技术潮流，另一方面对于 ES201X 又或者是 Koa2 来说，它们诞生的时间不长，笔者也不敢说自己对这些新的技术和标准已经有了丰富的编程经验（之前使用最多的还是 ES5 语法和 Express），正好也能站在一个入门者的角度来审视这些新技术。

本书的定位是一本新时期的 Node 入门书，关于“新”在哪里？一方面是内容的“新”，新的标准已经出现，怎么能够停滞不前！ES2015 带来了不少新的语言特性，它让之前需要花不少工夫才能解决的问题变得轻而易举。另一方面，本书不想重复介绍同类书籍已经反复介绍的内容，例如 Express 框架的使用，在市面上翻开任意一本 Node 的入门书籍，都会不厌其烦地向读者教授 Express 的各种用法，作为代替，本书推荐了 Koa 这一框架作为入门介绍，它更贴近新标准。

除了“如何使用”之外，本书还会兼顾 Node 底层的原理，读者大可在第一遍时跳过这些内容，当对 Node 有了一个大致的掌握后，会自然而然地想去了解其背后的原理。

关于本书内容

本书不是一本 ES2015 或者更新标准的说明书，也不想让内容停留在介绍各种模块的 API 罗列上。

第 1 章讲解 Node 的运行机制，主要是底层的一些实现和通用的原理，可能需要一些操作系统的知识。

第 2 章介绍了通用的模块和使用方法，是真正的入门章节。

第 3 章专注于新的 ECMAScript 标准以及 Node 对其的支持。

第 4 章主要介绍如何更好地组织和书写异步代码，采用循序渐进的方式介绍了各种解决方案的优劣之处。

第 5 章使用 Koa2 框架来开发一个 Web 应用，涉及 Web 开发的各方面，例如路由处理、Session、页面渲染、数据存储等。

第 6 章介绍了如何使用 Node 来开发一个爬虫系统。

第 7 章介绍常用的测试方法以及调试 Node 应用的技巧。

第 8 章介绍了 Node 中错误处理的相关知识。

附录 A 介绍进程、线程和协程的概念，属于拓展的背景知识。

附录 B Lua 语言简介，Lua 几乎可以认为是某些概念的最佳实现，例如协程。

附录 C 从零开发了一个玩具式的 Web 框架，可以认为是对 Koa 的一个简单模仿，对于初学者来说，这应该是一个理解 Node Web 原理的好方法。

附录 D MongoDB 和 Redis 简介。

附录 E 使用 Docker 来实现虚拟化。

附录 F 介绍了使用 npm 来进行包管理的一些小技巧。

本书的每个章节都是环环相扣的，每一章讲述的内容都多多少少地被其他章节使用到，建议读者循序渐进地阅读（第 1 章可以放到最后）。

循序渐进不仅仅体现在章节安排上，在系统的实现上也是如此，以第 6 章爬虫的开发为例，从糟糕的代码设计开始，一步步地进行改进，最后达到比较完善的状态。

那些糟糕的代码大部分都是在笔者还是初学者时写的，笔者很庆幸自己还能留着这些代码，它们不仅见证了笔者本人对 Node 的认识过程，也见证了 Node 的发展历史。

关于本书中的代码

所有的源代码都可以在 <https://github.com/Yuki-Minakami/BookExample> 上找到，它们都经过了充分的测试。

本书中，除了 Node 编写的示例代码外，还会穿插一些其他语言（例如 Java）的代码，这是为了通过和其他语言的对比让读者更好地理解 Node 中的特性。

笔者使用的电脑环境为 Mac OSX 10.11，本书中出现的代码绝大多数都是平台无关的，有一些代码在 Windows 环境下运行会出现问题或者不能在 Windows 下运行，本书也做了相应的标注和提示。读者在阅读本书的时候，也建议打开自己的电脑，第一时间把代码写在文本编辑器或者 IDE 中。

本书面向的读者

就像标题所说的，本书是一本入门书，适合在校的学生以及未接触过 Node 的读者阅读。如果读者有其他语言（例如 Java）的编程经验，那么读起来可能会轻松一些，如果有使用 JavaScript 的经验就再好不过了。

如何学习 Node

学习一门编程语言的最好办法，就是将其运用在实际的项目当中，但对于大多数开发者，尤其是自己目前的工作与 Node 无关时，想找到合适的项目并不容易，大多数人做的还只是非常简单的个人项目，例如一个 TODO List，复杂一点的比如一个博客网站，这些都算不上什么复杂的项目，从里面得到的经验也少得可怜。

那些能处理高并发、拥有各种多进程架构的项目不是每个人都有机会做，那么到哪里去找有一些难度的 Node 项目练手呢？

那就只能把目光投向 GitHub 了，使用 GitHub 的搜索功能来寻找一些企业级的 Node 应用，如果感兴趣的话就试着提交代码，为开源项目贡献代码通常是一个不错的加分项。

虽然有点王婆卖瓜，但笔者认为本书第 7 章的项目还是有一定的复杂度的，针对多进程和分布式的扩展还有很大的想象空间，读者可以借助 GitHub 参与到共同开发上来。

名称约定

为了便于区分，JavaScript 在本书中特指对 ECMAScript 的实现，除非特别注明，那么它代表了 ES5 的标准，并且同时适用于浏览器 JavaScript 和 Node。当有些代码和概念特指在浏览器端运行的 JavaScript 时，我们一律使用“前端 JavaScript”来称呼。

当使用 ECMAScript 这一称谓（例如 ES2017）时，大多数是谈论标准内容，不涉及具体的实现。

纠错

笔者毕竟能力有限，在出版本书的时候可能有没有注意到的错误，例如代码运行出错、概念上的不正确等，如果读者有相关的发现，请以“××章××小结代码/内容错误”为标题发邮件至笔者的邮箱 likaiboy_2012@126.com。

致谢

首先要特别感谢的是出版社的夏毓彦编辑，是他让我有机会梳理迄今为止对 Node 的心得，然后得以出版。

另一方面还要感谢我的母亲，当我第一次将自己准备写书的想法告诉她时，不出意外地，她开始怀疑起我的水平来，“就你还想出书？可不要误人子弟啊”。

正是这句话，不断提醒我对内容进行反复修改和验证。因为我意识到这和平时的博客文章不同，是更加严肃，并且对错误的容忍更低的作品。虽然她没有编程相关的经验，但我还是准备将第一本样书送给她，希望她能够阅读。

著者

2017 年 12 月

目 录

第 1 章 基础知识	1
1.1 Node 是什么	1
1.1.1 Node 与 JavaScript	1
1.1.2 runtime 和 VM	2
1.2 Node 的内部机制	3
1.2.1 何为回调	3
1.2.2 同步/异步和阻塞/非阻塞	4
1.2.3 单线程和多线程	6
1.2.4 并行和并发	7
1.3 事件循环 (Event loop)	8
1.3.1 事件与循环	8
1.3.2 Node 中的事件循环	9
1.3.3 process.nextTick	13
1.4 总结	16
1.5 参考资源	16
第 2 章 常用模块	17
2.1 Module	17
2.1.1 JavaScript 的模块规范	17
2.1.2 require 及其运行机制	18
2.1.3 require 的隐患	20
2.1.4 模块化与作用域	20
2.2 Buffer	22
2.2.1 Buffer 的构建与转换	23
2.2.2 Buffer 的拼接	24
2.3 File System	26

2.4	HTTP 服务	30
2.4.1	创建 HTTP 服务器	30
2.4.2	处理 HTTP 请求	32
2.4.3	Response 对象	34
2.4.4	上传数据	35
2.4.5	HTTP 客户端服务	38
2.4.6	创建代理服务器	39
2.5	TCP 服务	40
2.5.1	TCP 和 Socket	41
2.5.2	创建 TCP 服务器	41
2.6	更安全的传输方式——SSL	42
2.6.1	什么是 SSL	43
2.6.2	SSL 原理	43
2.6.3	对称加密与非对称加密	44
2.6.4	关于 CA	45
2.6.5	创建 HTTPS 服务	45
2.7	WebSocket	48
2.7.1	保持通话	48
2.7.2	为什么要有 WebSocket	49
2.7.3	WebSocket 与 Node	50
2.8	Stream	50
2.8.1	Stream 的种类	50
2.8.2	ReadLine	53
2.8.3	自定义 Stream	54
2.9	Events	55
2.9.1	事件和监听器	55
2.9.2	处理 error 事件	56
2.9.3	继承 Events 模块	57
2.10	多进程服务	58
2.10.1	child_process 模块	58
2.10.2	spawn	58
2.10.3	fork	59
2.10.4	exec 和 execFile	60
2.10.5	各方法之间的比较	62

2.10.6	进程间通信	64
2.10.7	Cluster.....	65
2.11	Process 对象.....	66
2.11.1	环境变量	67
2.11.2	方法和事件	67
2.11.3	一个例子：修改所在的时区	68
2.12	Timer	70
2.12.1	常用 API.....	70
2.12.2	定时器中的 this	71
2.13	小结	72
2.14	引用资源	72
第 3 章	用 ES6 来书写 Node	73
3.1	新时代的 ECMAScript.....	73
3.1.1	JavaScript 的缺陷	73
3.1.2	Node 对新标准的支持.....	74
3.1.3	使用 nvm 管理 Node 版本	75
3.2	块级作用域.....	75
3.2.1	ES5 中的作用域	75
3.2.2	let 关键字	77
3.2.3	const 关键字.....	78
3.3	数组	78
3.3.1	find()和 findIndex().....	79
3.3.2	from()方法.....	79
3.3.3	fill()方法.....	81
3.3.4	数组的遍历	81
3.3.5	TypedArray.....	82
3.4	函数	82
3.4.1	参数的默认值	82
3.4.2	Spread 运算符	83
3.4.3	箭头函数	83
3.4.4	箭头函数的陷阱	86
3.5	Set 和 Map.....	87
3.5.1	Set 和 WeakSet.....	87

3.5.2	Map 和 WeakMap	88
3.6	Iterator	89
3.6.1	Java 中的 Iterator	89
3.6.2	ES6 中的 Iterator	89
3.6.3	Iterator 的遍历	90
3.7	对象	91
3.7.1	新的方法	91
3.7.2	对象的遍历	92
3.8	类	93
3.8.1	属性和构造函数	94
3.8.2	类方法	94
3.8.3	__proto__	95
3.8.4	静态方法	96
3.9	类的继承	96
3.9.1	ES5 中的继承	96
3.9.2	ES6 中的继承	98
3.9.3	Node 中的类继承	100
3.10	ES6 的模块化标准	101
3.11	使用 babel 来转换代码	102
3.12	小结	106
3.13	引用资源	106
第 4 章	书写异步代码	107
4.1	异步操作的返回值	108
4.2	组织回调方法	108
4.2.1	回调与 CPS	108
4.2.2	使用 async 模块简化回调	110
4.3	使用 Promise	112
4.3.1	Promise 的历史	112
4.3.2	Promise 是什么	113
4.3.3	ES2015 中的 Promise	114
4.3.4	Promise 的常用 API	116
4.3.5	使用 Promise 组织异步代码	119
4.3.6	第三方模块的 Promise	120

4.4	Generator, 一种过渡方案	122
4.4.1	Generator 的使用	122
4.4.2	Generator 函数的执行	123
4.4.3	Generator 中的错误处理	126
4.4.4	用 Generator 组织异步方法	127
4.4.5	Generator 的自动执行	128
4.5	回调的终点——async/await	131
4.5.1	async 函数的概念	131
4.5.2	await 关键字	133
4.5.3	在循环中使用 async 方法	135
4.5.4	async 和 await 小结	136
4.5.5	async 函数的缺点	137
4.6	总结	138
4.7	引用资源	139
第 5 章	使用 Koa2 构建 Web 站点	140
5.1	Node Web 框架的发展历程	140
5.1.1	Connect	140
5.1.2	Express	141
5.1.3	Koa	141
5.2	内容规划	142
5.2.1	需求分析	142
5.2.2	技术选型	142
5.3	Koa 入门	143
5.3.1	Koa1.x 与 Koa2	143
5.3.2	context 对象	144
5.4	middleware	148
5.4.1	中间件的概念	148
5.4.2	next 方法	150
5.4.3	中间件的串行调用	151
5.4.4	一个例子——如何实现超时响应	152
5.5	常用服务的实现	154
5.5.1	静态文件服务	154
5.5.2	路由服务	155

5.5.3	数据存储	156
5.5.4	文件上传	160
5.5.5	页面渲染	163
5.6	构建健壮的 Web 应用	165
5.6.1	上传文件验证	166
5.6.2	使用 Cookie 进行身份验证	167
5.6.3	使用 Session 记录会话状态	170
5.7	使用 Redis 进行持久化	173
5.7.1	Node 和 Redis 的交互	173
5.7.2	CURD 操作	174
5.7.3	使用 Redis 持久化 session	176
5.7.4	Redis 在 Node 中的应用	179
5.8	Koa 源码剖析	180
5.8.1	Koa 的启动过程	180
5.8.2	中间件的加载	181
5.8.3	listen()方法	184
5.8.4	next()与 return next()	185
5.8.5	关于 Can't set headers after they are sent.	186
5.8.6	Context 对象的实现	187
5.8.7	Koa 的优缺点	189
5.9	网站部署	190
5.9.1	本地部署	190
5.9.2	部署在云服务主机上	191
5.9.3	通过 GitHub pages 来部署	193
5.10	总结	194
5.11	引用资源	194
第 6 章	爬虫系统的开发	195
6.1	爬虫技术概述	196
6.2	技术栈简介	196
6.2.1	request.js	196
6.2.2	cheerio	197
6.2.3	消息队列	199
6.3	构建脚手架	199

6.3.1	选择目标网站	199
6.3.2	分析 URL 结构	200
6.3.3	构建 HTTP 请求	200
6.3.4	解析页面元素	201
6.4	进行批量爬取	203
6.4.1	使用递归和定时器	203
6.4.2	多进程并行	205
6.5	爬虫架构的改进	206
6.5.1	异步流程控制	206
6.5.2	回到最初的目标	206
6.5.3	多进程模型的缺陷	208
6.6	进程架构的改进	208
6.6.1	生产/消费模型	208
6.6.2	生产者的实现	209
6.6.3	消费者的实现	211
6.7	反爬虫处理	213
6.7.1	爬虫的危害	213
6.7.2	识别一个爬虫	213
6.7.3	针对爬虫的处理	214
6.8	总结	216
6.9	引用资源	216
第 7 章	测试与调试	217
7.1	单元测试	218
7.1.1	使用 Assert 模块	218
7.1.2	Jasmine	219
7.1.3	Ava.js——面向未来	224
7.2	测试现有代码	227
7.3	更高维度的测试	228
7.3.1	基准测试	228
7.3.2	集成测试	229
7.3.3	持续集成	229
7.4	调试 Node 应用	231
7.4.1	语言和 IDE	232

7.4.2	使用 node-inspector	233
7.4.3	使用 v8-inspector	234
7.4.4	使用 IDE 进行调试	236
7.4.5	cpu profiling	237
7.5	总结	239
7.6	引用资源	239
第 8 章	Node 中的错误处理	240
8.1	Error 模块	241
8.2	错误处理的几种方式	241
8.3	被抛弃的 Domain	243
8.3.1	Domain 模块简介	243
8.3.2	Domain 原理	247
8.3.3	Domain 中间件	249
8.3.4	Domain 的缺陷	249
8.4	ES6 中的错误处理	250
8.4.1	Promise	250
8.4.2	Generator	250
8.4.3	async 函数	251
8.5	Web 服务中的错误处理	251
8.5.1	针对每个请求的错误处理	251
8.5.2	Express 中的错误处理	252
8.5.3	Koa 中的错误处理	252
8.6	防御式编程与 Let it crash	253
8.7	总结	256
8.8	引用资源	256
附录 A	进程、线程、协程	257
A.1	从操作系统说起	257
A.2	Node 中的协程	258
附录 B	Lua 语言简介	259
B.1	Lua 中的数据类型	259
B.2	定义一个函数	260

B.3 Lua 中的协程	261
附录 C 从零开发一个 Node Web 框架	263
C.1 框架的雏形	263
C.2 框架的完善	264
C.3 总结	268
附录 D MongoDB 和 Redis 简介	269
D.1 NoSQL	269
D.2 MongoDB 简介	269
D.3 Redis 简介	271
附录 E 使用 Docker 来实现虚拟化	274
E.1 Docker 的一些常用命令	274
E.2 Redis 服务	275
附录 F npm 与包管理	277
F.1 package.json 常用字段	277
F.2 依赖版本的管理	278

第 1 章

◀ 基础知识 ▶

——太阳底下没有新鲜事

本章主要介绍一些基本概念和 Node 的内部机制，如果读者对这部分暂时不感兴趣（事实上没人一开始就对这些概念感兴趣），可以先跳过这部分直接阅读第 2 章的内容。等对 Node 的使用有了大致了解之后，再回来看本章也不迟。

关于本章的内容，翻开任何一本经典的操作系统的教材都可以找到比本章更加全面和权威的描述（你可能有那么一瞬间后悔在学校没有认真掌握相关的知识，但这没关系，笔者也是这样），本章只负责介绍一些基础的概念，这有助于加深对 Node 的理解。

1.1 Node 是什么

在讨论所有 Node 相关的问题之前，我们必须明确一个问题，Node 是什么？

这看起来是一个再简单不过的问题，但如果不看答案（官网描述）直接回答起来却不是很容易，刚接触的开发者可能会认为 Node 就是 JavaScript（笔者当初也是这么想的），这种看法并不准确。

1.1.1 Node 与 JavaScript

回过头来看官网的定义：

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Node 是一个 JavaScript（严格来说是 ECMAScript）运行时（runtime），所谓的 runtime 直译过来就是运行时组件，读者可以将其想象成一种编程语言的运行环境。这个运行环境包括了运行代码需要的编译器（解释器）以及操作系统的底层支持等。

对一门编程语言来说，相对于语法本身，更重要的是编译器（解释器）将如何对待这些语法。Node 底层使用 C++ 实现，语法则遵循 ECMAScript 规范，如果创始人愿意，完全可以将 Node 创造成一个新的 Ruby 或者 Python 运行时，只不过名字大概就要改成 Node.rb 或者 Node.py 了。

这里有个扩展问题，编程语言是什么？

编程语言是一种抽象的规范，拿 C++ 来说，真正的 C++ 其实是厚厚的一摞文档，上面规