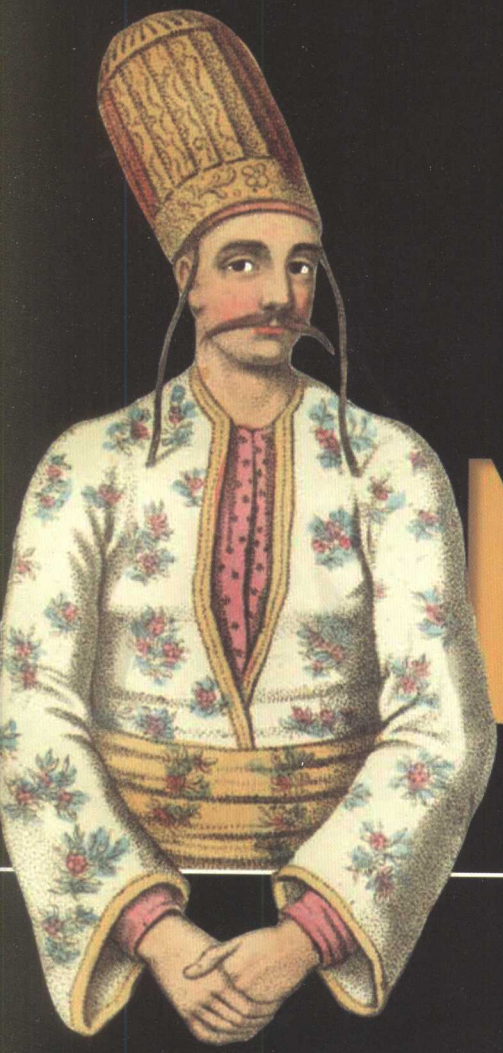
Broadview
www.broadview.com.cn

MANNING

Ben Noordhuis作序



# Node.js 硬实战

## 115个核心技巧

[美] Alex Young Marc Harter 著
承竹 慕陶 邱娟 达峰 译

Node.js in Practice

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硬实战
# 115个核心技巧

[美] Alex Young Marc Harter 著
承竹 慕陶 邱娟 达峰 译

# Node.js in Practice

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

## 内容简介

《Node.js 硬实战：115 个核心技巧》是一本面向实战的 Node.js 开发进阶指南。作为资深专家，本书作者独辟蹊径，将着眼点放在 Node.js 的核心模块和网络应用，通过精心组织的丰富实例，向读者充分展示了 Node.js 强大的并发处理能力，读者从中可真正掌握 Node 的核心基础与高级技巧。本书总共有三部分内容，第一部分是 Node.js 的基础核心，涉及 Buffer、流、网络和进程等相关知识；第二部分是项目实践，涉及测试、Web 开发、调试，生产环境等重要话题；第三部分则完整创建了一个 Node.js 模块。每部分涉及的技术都有详细讲解及注释详尽的示例代码，以帮助读者们更好地理解要点及其应用。

本书适合有一定 JavaScript 基础，追求在 Node.js 上更进一步的开发者。

Original English Language edition published by Manning Publications, USA. Copyright ©2015 by Manning Publications. Simplified Chinese-language edition copyright ©2016 by Publishing House of Electronics Industry. All rights reserved.

本书简体中文版专有出版权由 Manning Publications 授予电子工业出版社。未经许可，不得以任何方式复制或抄袭本书的任何部分。专有出版权受法律保护。

版权贸易合同登记号图字：01-2015-3992

**图书在版编目（CIP）数据**

Node.js 硬实战：115 个核心技巧 /（美）亚历克斯·荣（Alex R. Young），（美）马克·哈特（Marc Harter）著；承竹等译.—北京：电子工业出版社，2017.1
书名原文：Node.js in Practice
ISBN 978-7-121-30402-6
I. ① N…II. ① 亚… ② 马… ③ 承…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（2016）第 277756 号

责任编辑：张春雨
印　　刷：三河市良远印务有限公司
装　　订：三河市良远印务有限公司
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：30　　字数：672 千字
版　　次：2017 年 1 月第 1 版
印　　次：2017 年 1 月第 1 次印刷
定　　价：109.90 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。

# 译者序

JavaScript 从它诞生以来就在浏览器应用中发挥着愈来愈重要的作用，同时，热爱 JavaScript 的人们一直在努力地将其应用在服务器领域。2009 年，Ryan Dahl 带来了 Node.js，从那个时候开始，JavaScript 社区出现了前所未有的繁荣。现在，Node.js 对开发者来说，几乎已经是家喻户晓的了。对于传统的服务器端开发者们而言，Node.js 带来了浏览器端使用已久的异步编程相关的概念，而对于前端开发者们来说，Node.js 则是带来了编写服务端程序方面的挑战。当我们这些对 Node.js 一知半解的开发者野心勃勃想要征服 Node.js 时，我们需要一个经验满满的导师来指引我们前行。

我很幸运地有这样一个机会接触到这本书，并且承担起翻译的任务。我第一次阅读原著时就感觉，它是一本可以胜任 Node.js 导师这个角色的书籍。作者认真负责地介绍了 Node.js 相关的方方面面，并且附带了相当详细的例子来帮助读者快速地理解其中的要点。我相信，对于想要学习 Node.js 的开发者来说，本书会是一个相当好的选择。

翻译书的过程，也是一个学习的过程，原谅我们水平有限，书中翻译内容难免有疏漏。当这本书差不多翻译完成的时候，我发现已经用了好长的时间，Node.js 技术更新得相当快，我相信小部分内容的细节上和现在的应用可能会出现些许差异，还请读者们谅解。

最后，在此对支持本书翻译工作的所有人们表示感谢，尤其是耐心容忍我们一再推迟交稿的编辑，真的非常感谢。

衷心希望本书能够对您有所帮助，请享受 Node.js 给您带来的一切，谢谢。

# 序

在你手中的是一本将带你进入了解 Node 旅程的书。在接下来看到的页面中，Alex Young 和 Mare Harter 会帮助你深刻地把握 Node 的核心模块、网络应用等。

网络应用，是 Node.js 光芒闪耀的地方。亲爱的读者，你可能已经很清楚这一点，我敢说，这就是你购买本书的原因！对于读了序的这少部分人，让我来告诉你们，这一切是如何开始的。

在一开始的时候，有一个 C10K 的问题，这个问题是这样的：如果你要在现代的硬件中处理 10000 个并发连接，你会怎么做？

你可以看到，操作系统处理大量的网络连接在持续很长一段时间里都是很糟糕的。硬件在很多方面很糟糕，软件在另外一些方面也很糟糕，当硬件和软件集成在一起的时候……语言学家暂时还没有合适的词语，单纯用糟糕来形容对这一切并不公平。幸运的是，技术是一个进步的故事，硬件越来越好，软件越来越智能。操作系统，如用户软件等，在管理大量的网络连接上有了很大的进步。

在很久以前，我们征服了 C10K 问题，现在目标转移了，我们已经把眼光投向了 C100K、C500K、C1M 问题。一旦我们轻松地跨越这些界限时，我完全相信，C10M 会是下一个问题。

Node.js 是这个并发性不断增长的故事的一部分，它的未来是光明的。我们生活在一个日益互联的世界，这个世界需要一个强大的工具来连接一切。我相信 Node.js 会是那个强大的工具，我希望，在读完这本书时，你会有同样的感受。

Ben Noordhuis

Cofounder, StrongLoop, Inc.

# 前言

当 Node 在 2009 年出现的时候，我们知道有一些东西不一样了。在服务端 JavaScript 并不是什么新鲜的东西。事实上，服务端的 JavaScript 几乎和客户端的 JavaScript 存在的时间一样长。Node 中，JavaScript 运行时的速度，再加上基于事件的并行，这些很多熟悉 JavaScript 程序员都熟悉的东西，确实是让人感到不可抗拒。不仅仅是像我们这样背景的客户端 JavaScript 开发者——Node 吸引了从系统层面到各种服务端开发、PHP、Ruby 或者 Java 的开发者们。我们都可以发现自己身处于这变化中。

在那个时候，Node 有很多变化，我们受困于它，但是在这个过程中也学习了很多东西。从一开始，Node 关注于一个小的、低级别的核心库，来为大量多样化并且增长的用户提供足够的功能。值得庆幸的是，因为早期的一些设计决定，今天大量多样化的用户空间还存在着。Node 现在更加稳定，并且在许多初创企业和已成功企业生产环境中使用。

当 Manning 联系我们来编写一本中级的、关于 Node 的书时，我们想到了在过去和常见陷阱做斗争时，以及在 Node 社区中获得的经验教训。尽管我们非常喜欢那些提供给开发人员的大量真正优秀的第三方模块，但是，我们留意到越来越少的开发人员接触到 Node 核心基础的教学。所以我们着手编写了《Node.js 硬实战》，来以一种深入和彻底的方式，探索 Node 的根源和基础，并且解决很多我们个人遇见过的，或者看到他人处理过的问题。

# 致谢

我们要感谢很多人，没有他们的帮助和支持，这本书不可能完成。

感谢 Manning Early Access Program（MEAP）的在作者在线论坛发布评论和纠正的读者们。

感谢那些在书的各个阶段提供了宝贵反馈意见的技术评审们：Alex Garrett、Brian Falk、Chris Joakim、Christoph Walcher、Daniel Bretoi、Dominic Pettifer、Dylan Scott、Fernando Monteiro Kobayashi、Gavin Whyte、Gregor Zurowski、Haytham Samad、JT Marshall、Kevin Baister、Luis Gutierrez、Michael Piscatello、Philippe Charrière、Rock Lee、Shiju Varghese 和 Todd Williams。

感谢帮助我们每一步的 Manning 团队，尤其是我们的开发编辑 Cynthia Kane、编辑 Benjamin Berg、校对 Katie Tennant，以及其他在幕后工作的人们。

特别感谢 Ben Noordhuis 为本书写序，还有 Valentin Crettaz 和 Michael Levin 在书出版前仔细的技术校对。

## Alex Young

如果没有 DailyJS 社区的鼓励，我没法完成这样的一本书。感谢近几年来和我分享模块和类库的每一个人：没有你们就无法跟上 Node.js 社区的脚步。同时也要感谢我的同事允许让我在生产环境中使用 Node.js。最后，感谢 Yuka 让我相信我自己可以从事创业和写书这些如此疯狂的事情。

## Marc Harter

我要感谢 Ben Noordhuis、Isaac Schlueter，还有 Timothy Fontaine，为了他们关于 Node 的各种 IRC 讨论：你们了解底层系统，以一种深刻的方式支持和影响着 Node，从你们身上我们获益匪浅。另外，我要感谢和我一起写书的 Alex，似乎很难能有像我这样和 Alex 一起写书的机会，更加有趣的是他是一个在美国中西部的使用英语的人。最后我要感谢的是我的妻子，她让这一切变得可能，我由衷地说一句，你是如此可爱，谢谢你。

# 关于本书

《Node.js 硬实战》提供给读者来深入了解 Node 的核心模块和包系统。我们相信这是成为一个多产和自信的 Node 开发者的基础。不幸的是，因为巨大并且充满活力的第三方生态系统几乎为所有的任务提供了预置好的模块，所以小小的核心很容易被错过。在这本书中，我们在官方文档的基础上来进一步实践和深入。我们想要读者能够仔细分析和研究他们编写的项目以及项目所包含的第三方模块的内部工作。

这本书不是一本 Node 的入门级别的读物。入门的话，我们建议阅读 Manning 的《Node 实战》（*Node.js In Action*）。这本书的受众是那些对 Node 有一定经验，正在寻求更进一步的读者。建议有一定的 JavaScript 基础的读者，最好也熟悉 Windows、OS X 或者 Linux 命令行的读者阅读。

同时，我们注意到很多 Node 开发者有来自客户端的 JavaScript 开发背景。因此，我们会花一些时间来解释一些不大熟悉的概念，如处理二进制数据、底层网络和文件系统的工作，以及和主机操作系统进行交互——所有这些都使用 Node 来作为教学指南。

## 章节路线图

这本书共分为三部分。

第一部分包括了 Node 的核心基础，我们关注于那些可能用到的 Node 核心模块（非第三方模块）。第 1 章简要概述了 Node 的目的和意义。第 2 到第 8 章每一章内容会深入 Node 核心的不同方面，从 Buffers 到流，从网络到子进程。

第二部分内容关注于真实世界的开发技巧。第 9 到第 12 章的内容，将帮助你掌握 4 种

非常实用的技能——测试、Web 开发、调试以及在生产环境运行 Node。除了 Node 核心模块之外，这些章节的内容也会包括许多第三方模块的使用。

第三部分将指引你以一种直接的方式来创建自己的 Node 模块，使用 npm 命令的各种方法来处理打包、运行、测试、基准测试和共享模块。它同时也包括进行有效的项目版本化的有用提示。

整本书有 115 个技巧，每一部分包括了一个特定的 Node.js 主题或者任务，包括了实际问题、解决方案和讨论部分。

## 代码约定和下载

在本书中的所有代码都是像这样的固定宽度的字体，如 `fixed-width font like this`，以和周边其他文本内容区分开来。在很多列表中，代码中的关键概念会有相应的注解，带有编号的符号有时候可以帮助在文本中找到其他额外关于代码的有用信息。

这本书的代码风格基于 Google JavaScript 的编码风格[1]。这意味着我们把 var 声明独立开来，使用驼峰式来命名函数和变量，并且通常都带分号。我们的风格是在 Node 社区使用的多种 JavaScript 代码风格的混合。

大部分书中所示的代码可以以不同的形式在伴随着书的示例源代码中找到。示例源代码可以免费地从博文视点网站上下载：*www.broadview.com.cn*，也可以在 Github 上关注：*https://github.com/alexyoung/nodeinpractice*。

## 读者在线服务

购买《Node.js 硬实战》并注册账号，你可以在上边对本书进行评论、提交勘误或者咨询问题，从编辑或者其他用户那里获得帮助。

[1] *https://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml*

# 关于封面插图

《Node.js 硬实战》的封面插图的标题是“来自 Ayvalik 的年轻人”，Ayvalik 是在土耳其爱琴海岸的一个小镇。这个插图是从奥斯曼帝国在 1802 年 1 月 1 日发布的伦敦 Old Bond 街的 William Miller 创作的一系列服饰中获取的。标题页在整个集合中丢失了，我们已经无法追溯其日期。在书的内容表格中带有英文和法文标识，每个插图都有两位艺术家的名字。两百年之后，你一定会惊讶地发现，他们的艺术作品竟让计算机编程的书变得如此优雅、有魅力。

这一图片系列是由 Manning 的一位编辑在 Manhattan 的西第二十六街 Garage 的一个跳蚤市场购买的。卖家是一个在土耳其 Ankara 的美国人，交易就发生在他正要收摊的时候。Manning 的编辑身上并没有足够的现金，信用卡和支票都被礼貌地拒绝了。当天晚上卖家就要坐飞机回 Ankara，好像没有希望了。结果是怎么解决的？仅仅只是握手，一个老式的 verbal 协议。卖家简单地提出将钱电汇给他就好，编辑带着一张写了银行信息的纸和胳膊下的公事包就走了。当然，第二天我们就把钱汇过去了。我们依旧很感谢这个陌生人对我们的信任。这是很久前的一段回忆。

在 Manning 的我们，为自己的创意和突破，以及基于带回来的两个世纪前的照片集的计算机书籍封面能如此丰富多彩而庆幸。

# 目录

## 第一部分　Node 基础

# 第二部分 实践中的技巧

# 第一部分

# Node 基础

Node 拥有极小的标准库来为模块开发人员提供最底层的 API。虽然找到第三方模块相对容易，但很多任务可以不用它们来完成。在接下来的章节中我们将深入了解一些核心模块并且探索如何在实际中使用它们。

通过强化你对这些模块的理解，你将成为一个更为全面的 Node 开发者。在分析第三方模块时，你也能拥有更多的自信和理解。

# 入门

**本章概要**

- 为什么使用 Node
- Node 的主要特性
- 构建一个 Node 应用

Node 已经迅速成为一个可行并且真正高效的 web 开发平台。在 Node 诞生之前，在服务端运行 JavasScript 是件不可思议的事情，并且对其他的脚本语言来说，要实现非阻塞 I/O 通常需要依赖特殊的类库。但 Node 的出现改变了这一切。

JavaScript 与非阻塞 I/O 的组合极为强大：在 JavaScript 与生俱来的 callback 特性下，我们能在同一进程中异步地操作文件读写、网络 sockets 以及其他的 I/O 操作。

这本书面向的是有一定经验的 Node 开发者，因此本章只是帮助你进行快速的复习。如果你想从 Node 基础开始系统的学习，你可以阅读 *Node.js in Action*（作者 Mike Cantelon、Marc Harter、TJ Holowaychuk，以及 Nathan Rajlich；Manning 出版社 2013 年出版）。

在这一章中，我们将介绍 Node 是什么，它的工作原理，以及为什么它是不可或缺的。在第 2 章你将从 Node 全局对象——那些所有的 Node 进程都能访问的对象与方法开始体验 Node。

**阅读之前**

本书面向的是有一定经验的中高级 Node 开发者，虽然本章内容涵盖了一些入门资料，但后面的章节难度会很大。建议 Node 的入门开发者在阅读本书之前先看一下 *Node.js in Action*。

## 1.1 Node 入门

Node 是一个针对网络应用开发的平台，它基于 Google 的 JavaScript 运行时引擎 V8。但它不仅仅只是 V8。Node 的标准类库是它非常重要的一部分。它涵盖了从 TCP 服务端到同步或者异步的文件管理。这本书将教你如何正确地使用这些模块。

在这之前，让我们从一些场景入手来看看为什么使用 Node 以及何时应该使用它。

### 1.1.1 为什么使用 Node

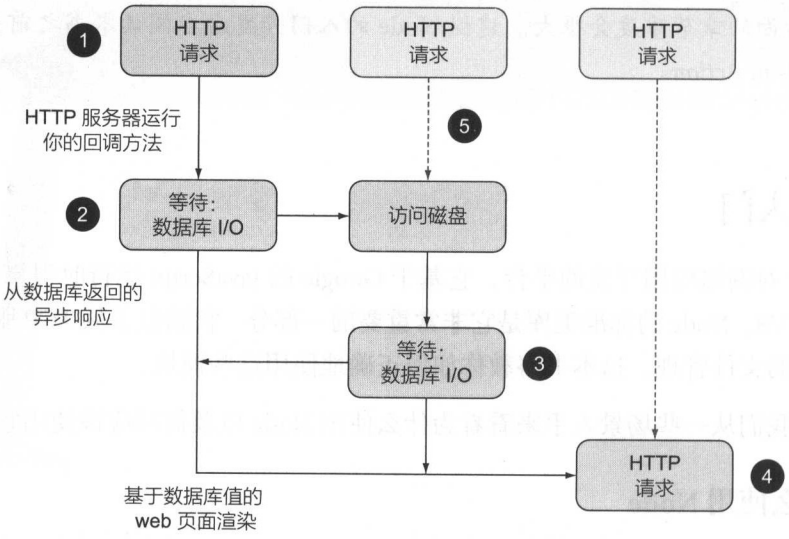
假设你正在开发一个广告服务器，每分钟需要发布几百万条的广告。Node 的非阻塞 I/O 将是一个高效的解决方案，因为服务器能够最大限度地利用到所有的 I/O 资源，而这一切不需要你写特殊的底层代码。并且，假如你已经有一支会写 JavaScript 的开发团队，那么他们应该可以直接参与到 Node 的项目中。传统的 web 平台将无法做到这一点，这也是为什么像微软这样的公司也在积极地推动 Node，尽管他们已经有了像.NET 那么优秀的平台。Visual Studio(.NET IDE) 的用户可以安装一些工具[1]来支持对 Node 的智能提示、性能监测，甚至 npm。微软还开发了 WebMatrix，它不但能直接支持 Node，而且还能部署 Node 项目。

Node 把非阻塞 I/O 作为提高某些类型应用的性能的方式。JavaScript 传统的事件机制意味着在异步编程中，它有着相对方便以及容易理解的语法。在传统的编程语言中，I/O 的操作将阻塞进程直到它完成为止。Node 的异步文件读写以及网络 API 意味着在这些相对较慢的 I/O 操作处理的时候主进程仍然能处理其他请求。图 1.1 展示了如何使用异步的网络和文件 API 同时处理多个任务。

在图 1.1 中，Node 的 `http` 模块接收到并且解析了一个新的 HTTP 请求❶，然后服务端的应用代码使用异步接口，将一个回调函数传入数据库的读取函数中来进行一次数据查询❷。在等待数据返回的同时服务器能够从文件系统中读取网页模板文件❸，这个模板

[1] 参见 *https://nodejstools.codeplex.com/*。

文件被用来展示网页。一旦数据库完成查询，模板内容和数据库的返回数据将被用来渲染页面❹。




❶ 接收一个从浏览器来的 HTTP 请求。

❷ Node 解析请求后，你的代码会运行一个数据库查询。

❸ 当查询回调方法在等待运行时，其他的代码会读取一个 HTML 模板文件。

❹ 然后 Web 页面会基于模板文件和数据库的值来进行渲染。

❺ 同时，其他请求同样可以被处理。

**图 1.1 用 Node 构建的广告服务器**

在服务器处理这个请求的同时，服务器还可以用可用的资源处理其他的请求❺。在不用考虑多线程的情况下开发这个广告服务，你可以仅使用最基本的 JavaScript 编程技术，通过 Node，非常高效地使用服务器 I/O 资源。

其他 Node 适用的场景是 Web API 和网络爬虫，如果你需要下载以及截取网页的内容，那么 Node 将是非常完美的解决方案，因为它能模拟 DOM 操作，并且运行客户端 JavaScript 脚本。而且在这个场景中，Node 有着性能优势，因为网络爬虫主要的消耗在于网络和文件读写的 I/O。

假如你需要调用或者开发一个 JSON API，Node 也是一个非常棒的选择，因为它使得操作 JavaScript 对象变得非常简单。Node 的一些 web 框架（例如 express）能够快速地搭建 JSON API，我们将在第 9 章详细讨论这个。

Node 不仅仅局限于 web 应用，你可以创建任意你想要的 TCP/IP 服务，比如网络游戏服务器，通过 TCP/IP 套接字向各类玩家发送游戏状态，也可以在后台任务中维护游戏数据，将数据发送给玩家。在第 7 章中，我们将探索 Node 的网络 API。

**什么时候使用 Node**

下面的表格有一些 Node 适用的应用例子，来帮你像一个真正的 Node 开发者一样来考虑这个问题。

Node 的强项

| 情景 | Node 的强项 |
| --- | --- |
| 广告分布系统 | • 有效地分配小块信息<br>• 处理潜在的网络速度慢的连接<br>• 容易扩展为多个处理器或服务器 |
| 游戏服务器 | • 使用 JavaScript 来构建业务逻辑模型<br>• 不使用 C 语言来开发迎合特定网络的服务器程序 |
| 内容管理系统、博客 | • 对已经有客户端 JavaScript 开发经验的团队来说，可以很轻松地创建 RESTful JSON APIs<br>• 轻量的服务器，和浏览器端 JavaScript 结合 |

## 1.1.2 Node 的主要特性

Node 的主要特性是它的标准类库、模块系统以及 npm（包管理系统），当然还有很多其他的。但本书主要针对教你如何使用 Node 的这些部分。我们将使用被认为是最佳实践的第三方类库，但你会看到很多 Node 的内置特性。

实际上 Node 最强大的特性是它的标准类库，它主要由二进制类库以及核心模块两部分组成，二进制类库包括 `libuv`，它为网络以及文件系统提供了快速的事件轮循以及非阻塞的 I/O。同时它还有 http 类库，所以你可以很快确定你的 http 客户端与服务端。

图 1.2 是对 Node 内部的高层次概述，展示了各个模块是如何组合的。

Node 的核心模块主要由 JavaScript 编写，也就是说，假如你不理解或者你想了解更多细节，你可以直接阅读 Node 的源码。这不但包括像网络、文件操作、模块系统，以及 stream 这些模块，还包括 Node 特有的特性，例如，通过 cluster 模块同时运行多个 Node 进程，以及可以将代码片段封装在事件驱动的异常处理中的 domain 模块。

接下来，我们将从事件开始深入每个核心模块。

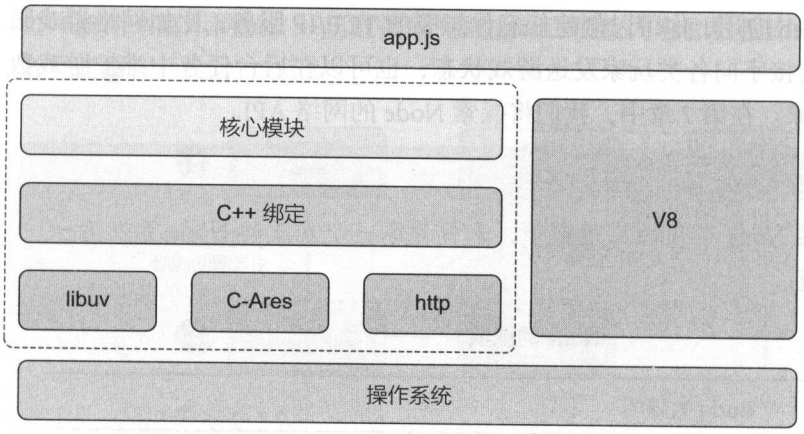



图 1.2 Node 环境中的关键部分

### EventEmitter 事件的接口

每个 Node 开发者迟早会碰到 EventEmitter，一开始，它像是那些只有类库开发者才会使用的东西，但实际上它是大多数 Node 核心模块的基础，Stream、网络、文件系统全部继承于它。

你可以基于 EventEmitter 来创建自己基于事件的 API，例如你要开发一个 paypal 付款处理的模块，你可以让它基于事件，这样 Payment 对象的实例可以触发像 paid 和 refund 这样的事件，通过这样的设计，你可以将这个模块从你的业务逻辑中分离出来，让它可以在其他项目中被重用。

我们在第 4 章有一整章的内容来讲解事件。另外一个有意思的地方是，stream 模块也是基于 EventEmitter 的。

### Stream：高可扩展性 I/O 的基础

Streams 继承于 EventEmitters，能被用来在不可预测的输入下创建数据，比如网络连接，数据传输速度取决于其他用户正在干什么。通过 Node 的 stream API，你可以创建一个对象接收关于连接的事件，在接收到新数据时触发 data 事件，在结束连接时触发 end 事件，在有错误发生时触发 error 事件。

相比较把许多的回调函数传入一个 readable stream 的构造函数，你只订阅你关心的事件要好得多，多个 streams 也可以连接起来，这样你可以用一个 stream 对象从网络读取数据，把读取到的数据输送到另外一个 stream 中加工成另外一个对象，可以把 xml 文件的数据读取出来转换成 JSON 格式，让 JavaScript 操作起来更容易。

我们喜欢 Stream，所以，为它准备了一整个章节。你可以直接跳到第 5 章去深入了解它。你可能觉得 stream 和事件听上去很抽象，没错，它们的确很抽象，但它们是 I/O 模块（例如文件系统和网络）的基础。

### FS：处理文件

Node 的文件模块不但可以通过非阻塞的 I/O 读写文件，而且它也有同步的方法。你可以通过 `fs.stat` 异步获取文件的信息，也可以通过 `fs.statSync` 同步读取。

如果你想通过 Stream 的方式高效地处理文件内容，那么你可以通过 `fs.createReadStream` 来获得一个 `ReadableSream` 对象。在第 6 章，我们会更多地讨论这个方法。

### 网络：创建网络客户端与服务端

网络模块是 `http` 模块的基础，也可以用来创建通用的网络客户端与服务端。尽管 Node 开发通常指的是 web 开发，在第 7 章你会看到如何创建 TCP 和 UDP 的服务，这意味着你并不局限于 http 开发。

### 全局对象与其他模块

假如你有用 Node 开发 web 应用的经验，也许是 Express 框架，那么你也许并不知道你已经使用了 `http`、`net` 以及 `fs` 等核心模块。其他的内置模块也许不那么吸引眼球，但也是至关重要的。

全局对象与方法的设计就是其中一例，比如 `process` 对象，它让你可以把数据传入或者传出标准 I/O 流（stdout、stdin）。就像在 UNIX 或者 Windows 脚本中，你可以把数据通过 `cat` 直接传给 Node 程序。还有无处不见的 `console` 对象，所有的 JavaScript 开发都爱它，也是一个全局对象。

Node 的模块系统也是全局方法。在第 2 章，你将看到如何使用这些特性。

现在你已经了解了一些核心模块，是时候看看怎么使用它们了。下面的例子，我们将会使用 `stream` 模块来生成文件流的数据，你可以用在文件和 http 连接上。如果你想了解更多关于 stream 和 http 的基础，那请参阅 *Node.js in Action*。

## 1.2 构建一个 Node 应用

不多废话，我们将带着你一起构建一个 Node 应用。虽然它不是什么大应用，但它会用到一些 node 的核心功能，比如模块和流，这将是一个快速紧张的过程，那么，打开你的编辑器和终端，开始吧。

接下来的 10 分钟你将学到：

- 如何开始一个新的 Node 项目
- 如何创建你自己的 Stream 类
- 如何编写运行一个简单的测试

Stream 非常擅长处理数据，无论是读、写或者是转换。想象一下，你需要把数据从数据库导入到另外一个格式，比如 csv。你可以创建一个 Stream 类，接收从数据库输入的数据，并将它输出到 csv 的流中，这个新的 csv 流可以传入一个 http 请求，这样，这个 csv 的输出可以直接显示在浏览器上。它还可以接入一个可写的文件流，你甚至可以用这个流来创建一个文件，并发送至浏览器。

在这个例子中，这个 Stream 类可以接受文本输入，记录通过正则表达式匹配的单词，在文件流发送完后把结果通过一个事件发布出去。你可以用它来记录文件中匹配的单词，也可以把网页内容传入这个流，来记录 p 标签的数量，这都取决于你，但首先你需要创建一个新的项目。

### 1.2.1 创建一个新的 Node 项目

你也许在猜专业的 Node 开发如何创建一个新的项目，因为有 npm 在，这是个非常简单的过程。虽然你可以创建一个 JavaScript 文件，并且执行 `node file.js`，我们将使用 `npm init` 来创建一个带有 package.json 文件的新的项目。创建一个新的目录❶，`cd`❷进去，然后运行 `npm init`❸。

```
mkdir first-project          ❶
  cd first-project           ❷
  npm init                   ❸
```

❶ 创建一个新的目录。

❷ 进入目录中。

❸ 创建项目的清单文件。

要习惯敲这些命令，你会经常使用它们！你可以按回车键来接受 npm 提供的默认值。在写 JavaScript 代码之前，你已经领略了 Node 最主要的功能之一——npm 是多么酷。它不但可以安装模块，也可以用来管理项目。

现在是时候写 JavaScript 了。在下一节，你将创建一个新的 JavaScript 文件来实现一个 Stream 类。

**何时使用 package.json 文件**

你可能想写一个小脚本，在犹豫是否需要创建一个 package.json 文件。它并不是必需的，但通常你应该尽可能地创建它们。

Node 开发者倾向于小模块，在 package.json 中写明依赖意味着你的项目无论多小，在未来或者其他开发者的机器上是非常容易安装的。

### 1.2.2 创建一个流的类

创建一个名为 countstream.js 的新文件，使用 `util.inherits` 来继承 `stream.Writable` 并且实现必要的 `_write` 方法。太快了点？我们缓一缓，先看看下边的源码。

例子 1.1 一个用于计数的可写流

```
var Writable = require('stream').Writable;
var util = require('util');

module.exports = CountStream;

util.inherits(CountStream, Writable);                          ❶

function CountStream(matchText, options) {
  Writable.call(this, options);
  this.count = 0;
  this.matcher = new RegExp(matchText, 'ig');                  ❷
}

CountStream.prototype._write = function(chunk, encoding, cb) {
  var matches = chunk.toString().match(this.matcher);          ❸
  if (matches) {
    this.count += matches.length;
  }
  cb();
};

CountStream.prototype.end = function() {
  this.emit('total', this.count);                              ❹
};
```

❶ 继承可写流。

❷ 创建一个全局且忽略大小写的正则对象。

❸ 把当前的输入数据转化为字符串并进行匹配。

❹ 当输入流结束时，触发 total 事件。

这个例子简单地展示了书中后续的例子是怎么样的。我们给出了一个代码片段，并且附加了代码的注释。例如，第一部分代码的类使用了 util.inherits 方法来继承一个 Writable 基类❶。这个例子没有完整地在这里展示，更多的可以参考第 5 章的第 30 个技巧。现在，我们把重点放在正则表达式是如何传递给构造函数❷以及如何在文本流入类的示例时对其计数❸的。Node 的 Writable 类为我们调用 _write 方法，暂时可以不用去关注它。

**Streams 和事件**

在例子 1.1 中有一个事件，total。这是我们自己创建的——当然你也可以自己创建一个。Streams 从 EventEmitter 继承而来，所以它们都拥有同样的 emit 和 on 方法。

当不再有数据时，Node 的 Writable 基类会调用 end 方法❹。该类可以被实例化，并且根据需要来通过管道进行传输数据。下一节你会了解到如何使用管道来连接多个流。

### 1.2.3 使用流

现在你已经知道如何创建一个流的类，很可能你已经实践过了。现在创建一个新的文件，index.js，并加入如下的代码。

例子 1.2 使用 CountStream 类

```
var CountStream = require('./countstream');                    ❶
var countStream = new CountStream('book');                     ❷
var http = require('http');

http.get('http://www.manning.com', function(res) {             ❸
  res.pipe(countStream);                                       ❹
});

countStream.on('total', function(count) {
  console.log('Total matches:', count);
});
```

❶ 加载 countstream.js。

❷ 创建一个 CountStream 的示例用于匹配 book 的文本计数。

❸ 下载 www.manning.com 页面。

❹ 从网站中以管道的方式把数据传给 countStream 用于文本计数。

你可以尝试着使用 `node index.js` 来执行这个例子的代码。它应该会打印 `Total matches: 24` 之类的。你可以尝试着更改一下所要抓取的地址。

这个例子加载了例子 1.1 ❶中的模块并通过字符串 `'book'` 将它实例化❷。同时使用了 Node 的标准模块 `http` 从网站下载文本内容❸以及使用管道把结果传入到我们的 `CountStream` 类中❹。

这里最重要的是 `res.pipe(countStream)`。当你使用管道传输数据时，不用去关心数据有多大或者网络速度多慢：`CountStream` 类会完整进行匹配计数直到数据全被处理完。这个 Node 的程序并不会在一开始下载整个文件！它会把文件一块一块地进行处理。这是很重要的，也是 Node 提供的一个关键的特性。

总结一下，图 1.3 概括了到现在为止，创建一个新的 Node 程序需要做什么。首先创建一个新的目录，执行 `npm init` 命令❶，然后创建一些 JavaScript 代码文件❷，最后执行代码❸。

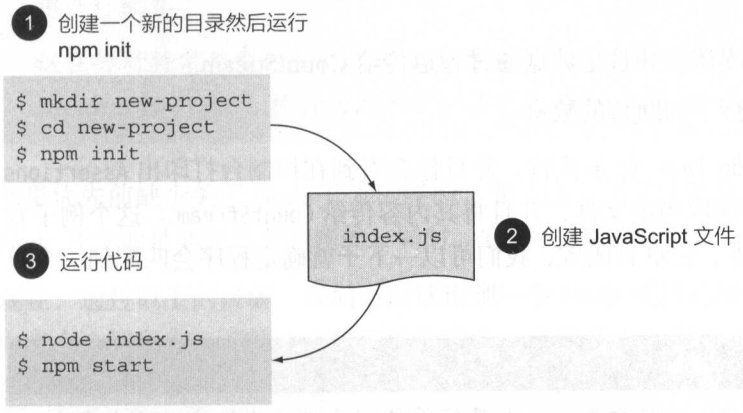



图 1.3 创建新的 Node 项目的 3 个步骤

关于 Node 环境，另外一个很重要的便是测试，下一章将会给出测试 `CountStream` 的例子。

### 1.2.4 编写测试

我们可以不借助任何第三模块来为 `CountStream` 编写一个简单的测试。Node 已经内置了 `assert` 模块用于提供断言，我们可以用它来进行快速的测试。创建 test.js 并且加入如下代码。

例子 1.3 使用 CountStream 类

```
var assert = require('assert');
var CountStream = require('./countstream');
var countStream = new CountStream('example');
var fs = require('fs');
var passed = 0;

countStream.on('total', function(count) {                    ❶
  assert.equal(count, 1);                                    ❷
  passed++;
});

fs.createReadStream(__filename).pipe(countStream);           ❸

process.on('exit', function() {
  console.log('Assertions passed:', passed);                 ❹
});
```

❶ 当流结束时，*total* 事件将会被触发。

❷ 断言所预计的计数。

❸ 为当前文件创建一个可读流，并且把数据通过管道传给 CountStream。

❹ 在程序结束前，展示执行了的断言的数量。

这个测试代码可以使用 `node test.js` 来执行，并且你会看到在控制台打印出 `Assertions passed: 1`。这个测试代码读取当前文件，并且将其内容传给 `CountStream`。这个例子有点问题，但是在这个场景下，它卓有成效，我们可以一下子就确定程序会匹配到一个单词 *example*。

**断言**

Node 内置了一个叫 `assert` 的断言类。一个最简单的测试可以直接使用这个类来构建——`assert(表达式)`。

测试的例子首先做的是监听 `CountStream` 的实例会触发的 `total` 事件❶。这里可以很好地对所期望的匹配数进行断言❷。一个可读流将把关联的当前打开的文件的数据以管道的方式传给我们的类❸。然后，在程序结束前，我们可以打印出多少断言是正确的❹。

有一点非常重要，如果 `total` 事件永远不会被触发，那么 `assert.equal` 也就不会执行。我们便没有办法知道在回调函数中的测试是否执行了，所以这里使用了很常见的计数器来

进行标示。在 Node 程序中也同样可以使用你可能已经很熟悉的其他语言或者平台中常见的编程模式。

如果你有点厌倦了，那么可以休息下。但还有一些好东西来帮助我们更好地实现项目。Node 开发者喜欢在命令行里使用 npm 命令来跑测试或者其他脚本。打开 package.json，把 "test" 属性改成这样子：

```
"scripts": {
    "test": "node test.js"
},
```

现在就可以输入 npm test 来运行我们的测试了。当你有很多测试需要运行或者运行测试变得复杂时，这可以帮助你更加方便地来处理测试。运行测试，测试运行环境，异步测试等相关内容都会在第 10 章讲述。

> npm scripts
>
> npm test 和 npm start 命令可以在 package.json 中来进行配置。你也可以使用 npm run command 来运行自定义的一些命令。这需要你像上述例子一样在 scripts 属性中进行配置。
>
> 这在针对特定类型的测试或者经常要反复执行的程序时特别有用——如 npm run integration-test，执行集成测试，甚至是 npm run seed-data 来生成种子数据。

如果你先前缺少关于 Node 开发的经验，那么这个例子可能有点难，但是它让我们了解 Node 开发者是如何思考问题的，同时，它向我们展示了 Node 强大的优势。

现在，通过以上的讲解，我们可以认识到一个 Node 项目是如何被构建起来的。下一章中将会介绍我们一开始会使用到的技术点，会涉及到在所有 Node 程序中可以使用到的全局特性。

## 1.3 总结

在这一章中，你已经了解了本书涵盖的内容，以及它如何聚焦 Node 中一些强大的内置核心模块，如网络模块、文件系统模块。

你也了解到 Node 为什么那么火，以及如何使用 Node。我们已经提及的有以下多个方面：

- 什么时候适合使用 Node，以及 Node 如何在非阻塞 I/O 的基础上来构建一个高性能的标准 JavaScript 的执行环境。

- Node 的标准库，也称为它的核心模块。
- 核心模块如何处理像网络协议、文件读写或者像流这样更通用的特性中的 I/O。
- 快速开始创建一个 Node 项目以及利用 package.json 来添加依赖和可执行脚本。
- 使用 Node 强大的 `stream` API 来处理数据。
- Streams 类继承于 `EventEmitter`，你可以使用它来触发或者响应你需要使用的各种事件。
- 利用 npm 和 `assert` 模块来编写一个简单的测试，而无须依赖任意第三方模块。

总的来说，我们希望你已经从简单的入门应用学习到一些东西，虽然 Node 无非就是使用基于事件的 API、非阻塞的 I/O，以及 stream，但是利用 Node 提供的独特的工具，像 package.json 配置文件以及 npm 也很重要。

又回到讨论技术的时间了，下一章会介绍不用额外加载就可以使用的全局对象。

# 2 全局变量：Node 环境

**本章概要**

- 使用模块
- 不使用其他模块你能做什么
- process 与 console 对象
- Timers

全局对象能被用在所有的模块中，无论你是写一个网络应用、命令行脚本还是网站应用，你的程序都能访问这些对象。这意味着，你永远可以依赖像 `console.log` 与 `__dirname` 这些特性，我们会在本章详细解释这两个特性。

这一章节的目的是介绍 Node 的全局对象与方法，来帮助你学习那些能被所有 Node 进程所使用的方法。这能帮助你更好地理解 Node、它与操作系统的关系，以及它和其他 JavaScript 运行时的区别，比如浏览器。

Node 提供一些重要的内置方法，甚至不用加载其他的模块。除了那些 ECMAScript 提供的功能，Node 有几个由 Node 提供的 host 对象，来帮助 Node 程序执行。

一个重要的全局对象是 `process`，它可以用来与操作系统通信。UNIX 开发者所熟悉的标准输入输出流，可以通过 Node 的 streaming 接口从 `process` 对象获取。

本PDF仅是样章,书籍版权归著者和出版社所有，未经允许不能在网上传播，如有需要，请尽量购买正版实体书，以表示对知识的尊重！实在有必要获取完整版本PDF，请联系QQ:2856202282,谢谢！

2856202282