写作历史

Node.js布道者 朴灵 **文艺型码农**

比其他Node图书的作者，他在淘宝一线的开发经验使这本书更有深度，而他**文艺青年的背景**让这本书读起来极其顺畅，他的钻研精神又让这本书在理论上很有深度。

以上说明了写作的重要性，作家的重要性

本章参考的资源如下：

❑ <http://www.infoq.com/cn/articles/what-is-nodejs>

❑ <https://github.com/popular/watched>

❑http://groups.google.com/group/nodejs/browse\_thread/thread/85f6a3829bc64cb6

❑ <http://groups.google.com/groups/profile?enc_user=dPo6jggAAACthftLMWCfUq8U6obMz179>

❑ <http://search.npmjs.org/>

❑ <http://code.google.com/p/v8/>

**采访Node.JS的创始人Ryan Dahl**

❑ <http://cnodejs.org/topic/4f16442ccae1f4aa27001137>

❑ <http://weibo.com/1744667943/eBszJXcEsX1>

[Why is Node.js named Node.js? [closed]](https://stackoverflow.com/questions/5621812/why-is-node-js-named-node-js)

❑ <http://stackoverflow.com/questions/5621812/why-is-node-js-named-node-js>

看到 Node.js 这个名字，初学者可能会误以为这是一个 Javascript 应用，事实上，Node.js 采用 C++ 语言编写而成，是一个 Javascript 的运行环境。

Qq注册绑定手机号

<https://www.infoq.cn/article/what-is-nodejs/>

# Access denied Error code 1020

You do not have access to bostinnovation.com.

The site owner may have set restrictions that prevent you from accessing the site. Contact the site owner for access or try loading the page again.

访问被拒绝错误码1020您无法访问bostinnovation.com。网站所有者可能设置了阻止您访问网站的限制。请联系网站所有者以获得访问权限或尝试重新加载页面。

序一 CNode社区创始人 空无

序二 谢骋超 网易高级技术专家、架构师 2013年7月8日

Node可以驱动后端复杂的应用程序,在开发高并发、高性能后端服务程序上也有着极大的优势。

此书作者有机会在很多高并发的应用场景中反复实践。这绝对是一本实践性极强的技术书

Node诞生于2009年，天才的屌丝青年Ryan Dahl利用了Google的V8引擎打造了基于事件循环实现的异步I/O框架

2010年,国内外了解Node的人寥寥可数

2011年年初创办CNode开源技术社区（CNodeJS.org）培养熟悉Node的人才。

Node的爱好者大都是些喜欢探索新技术的极客

V8的性能远超其他脚本语言，这成为Node成功的极其重要的因素

淘宝一线的开发经验使这本书更有深度

朴灵自称“一个能搞定回调函数嵌套的男人”

在第4章中，他详细介绍了Node的各种嵌套函数过深的解决方案，例如EventProxy、Promise、async、step、wind.js等各种解决方案都有深入讲解。此外，朴灵还是EventProxy的作者，在这方面有最权威的实践经验。

前言

**究其原因，还是因为人们缺乏历史观层次上的认知，所以会产生一些莫须有的惴惴不安。**

1995年，JavaScript随网景公司发布的Netscape Navigator2.0发布，它最早命名为LiveScript，随后更名为JavaScript。它出自如今的Mozilla公司的CTO——Brendan Eich之手，其产生来源于网景公司发布的Netscape Navigator浏览器需要一种脚本语言来协助浏览器做一些简单的动态操作。当时网景公司与Sun公司合作密切，不懂技术的管理层希望得到一个Java的脚本版语言，以期能像Java一样风靡。Brendan Eich原本进入网景公司是希望做Scheme语言的开发，但是却接到了一个不喜欢的任务，但迫于当时形势，不得不完成此事，于是JavaScript之父在10天的时间里仓促完成了JavaScript的设计，当时的项目代号是Mocha，名字叫LiveScript。

虽然JavaScript的产生与Netscape Navigator浏览器的需求有关系，但它并非只是设计出来用于浏览器前端的。早在1994年，网景公司就公布了其Netscape Enterprise Server中的一种服务器端脚本实现，它的名字叫LiveWire，是最早的服务器端JavaScript，甚至早于浏览器中的JavaScript公布。对于这门图灵完备的语言，网景早就开始尝试将它用在后端。

随后，微软在第一次浏览器大战时，于1996年发布的IE 3.0中也包含了它的脚本语言：JScript。基于商标的原因，它叫JScript，但是与JavaScript兼容。在1997年年初，微软在它的服务器IIS 3.0中也包含了JScript，这就是我们在ASP中能使用的脚本语言。鉴于微软处处与网景针锋相对，出于保护自己的目的，网景公司推进了JavaScript的标准化进程，于1996年11月将JavaScript递交给ECMA国际标准组织，在1997年7月公布了第一个版本，是为ECMA-262号标准，又称ECMAScript。

本书目的：介绍Node自身结构

第2章是从代码组织结构看待Node

第3章是从运行结构看Node，第4章则是从编程结构看Node

第5章则是Node中内存结构的揭示

第6章谈及的是Node中的数据在I/O流中的结构或状态

第7章是Node在网络服务角度的介绍

第8章是Node在HTTP上的展现

第9章讨论了Node的单机集群结构

第10章是从单元测试和性能测试的角度去关注Node

第11章虽然已经脱离了Node编码的范畴，但是站在产品化的角度看待Node，也会颇有收获。

下麵是各章的詳細介紹。

第1章：这一章简要介绍了Node，从中可以了解Node的发展历程及其带来的影响和价值。第2章：这一章介绍了Node的模块机制，从中可以了解到Node是如何实现CommonJS模块和包规范的。在这一章中，我们详细解释了模块在引用过程中的编译、加载规则。另外，我们还能读到更深度的关于Node自身源代码的组织架构。第3章：这一章展示了在Node中我们将异步I/O作为主要设计理念的原因。另外，还会介绍到异步I/O的详细实现过程。第4章：这一章主要介绍异步编程，其中有常见的异步编程问题介绍，也有详细的解决方案。在这一章中，我们可以接触到Promise、事件、高阶函数是如何进行流程控制的。第5章：这一章主要介绍了Node中的内存控制，主要内容有垃圾回收、内存限制、查看内存、内存泄漏、大内存应用等细节。第6章：这一章介绍了前端JavaScript里不能遇到的Buffer。由于Node中会涉及频繁的网络和磁盘I/O，处理字节流数据会是很常见的行为，这部分场景与纯粹的前端开发完全不同。

[Hi Ryan, would you like to introduce yourself?](javascript:;" \o ")

<https://www.infoq.com/interviews/node-ryan-dahl/>

[Hi Ryan, would you like to introduce yourself?](javascript:;" \o ")

Yes. Thank you. I am Ryan Dahl, I work at a company called Joyent which does hosting services and I made Node.js which I assume is why I am being interviewed.

[What was your motivation for beginning to work on Node.js? What problem were you trying to solve and what was the solution you came up with?](javascript:;" \o ")

I was working a lot with Ruby web servers. Originally I am from a math background so I only got into computing a little while ago and I got really interested in Ruby and was fascinated by this problem of uploading files and giving a progress bar to the person uploading the file and it just amazed me that this was such a hard problem because obviously the web browser actually has like the data to display that stuff, but it's just not accessible from the DOM so just what you do is send a request to the web server and ask it how much of this file have you received and it comes back and then you update the DOM with this thing. And there is a nice little module for Mongrel which did this.  
  
I mean traditionally you need to write some sort of module for Apache or something like this and I was just impressed that Mongrel was kind of this functioning web server in a dynamic language and it impressed me how easy it was to kind of get inside of that web server. And so I would say that that is probably how I got into this and fast forward a couple of years and I think combining the kind of Javascript "arms race" that was occurring plus was I've learned about evented programming kind of seamed very natural and it was kind of an experiment but it seemed to turn out well.

2.1 CommonJS规范

CommonJS，顾名思义js的规范化。

为什么要规范

规范化可以让js变得像Python、Ruby和Java具备开发大型应用的基础能力，而不是停留在小脚本程序的阶段

不规范的坏处

规范带来的好处

Js规范化，Node能以一种比较成熟的姿态出现

Node借鉴CommonJS的Modules规范实现了一套非常易用的**模块系统**，NPM对Packages规范的完好支持使得Node应用在开发过程中事半功倍。

**CommonJS对模块module的定义**

主要分为模块引用、模块定义和模块标识3个部分。

const path = require('path')  
const VueLoaderPlugin = require('vue-loader/lib/plugin')  
  
module.exports = {  
 mode: 'production',  
 entry: './src/index.js',  
 output: {  
 path: path.resolve(process.cwd(), './lib'),  
 filename: 'element-ui.common.js',  
 libraryTarget: 'commonjs2',  
 },  
 module: {  
 rules: [  
 {  
 test: /\.vue$/,  
 loader: 'vue-loader',  
 options: {  
 complierOptions: {  
 preserveWhitespace: false  
 }  
 }  
 }  
 ]  
 },  
 plugins: [  
 new VueLoaderPlugin()  
 ]  
}

模块的意义在于将类聚的方法和变量等限定在私有的作用域中，同时支持引入和导出功能以顺畅地连接上下游依赖。如图2-3所示，每个模块具有独立的空间，它们互不干扰，在引用时也显得干净利落。

这套模块导出和引入机制使得用户完全**不必考虑变量污染**，**命名空间等方案与之相比相形见绌。**