HZ BOOKS
华章科技

资深Web开发专家根据Node.js的最新版撰写，对Node.js的所有功能、特性、使用方法和开发技巧进行了全面而深入的讲解，是系统学习Node.js的权威参考书

以实践为导向，不仅为每个知识点配备了精巧的小案例，而且还设计了两个可操作性极强的综合案例

實戰

陆凌牛 著

Node.js: The Definitive Guide

Node.js 权威指南

机械工业出版社
China Machine Press

Node.js: The Definitive Guide

Node.js 权威指南

陆凌牛 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Node.js 权威指南 / 陆凌牛著 . —北京：机械工业出版社，2014.4
（实战）

ISBN 978-7-111-46078-7

I. N… II. 陆… III. JAVA 语言 – 程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2014）第 045238 号

本书旨在成为 Node.js 领域最全面、最系统和最具实战性的一本书，供初学者系统学习和开发者工作时参考。本书作者是资深 Web 开发专家，不仅全面、细致地讲解了 Node.js 的所有功能、特性、使用方法和开发技巧，而且还介绍了与之相关的各种扩展功能和工具的使用。此外，它实战性强，不仅每个知识点都配有精心设计的小案例（具体的实现步骤、完整的实现代码、最终的实现效果，图文并茂），而且还有两个综合性的案例，能让读者迅速获得实战经验。本书所有实例代码都经调试运行成功，读者可以对这些代码进行修改，以便观察各种不同的效果，加深对实例代码的理解。

全书共 16 章，分三个部分：第一部分（第 1 ~ 12 章）从 Node.js 的特性、优势、适用场景、安装配置到各个模块的功能作用，再到开发的具体方法和技巧，非常详尽地讲解了 Node.js 开发者必备的各种基础知识；第二部分（第 13~15 章）介绍了在使用 Node.js 进行开发时极有可能用到的第三方开发包，探讨了如何在 Node.js 应用程序中使用关系型数据库及 NoSQL 型数据库，如何使用 Express 框架开发 Web 应用程序，以及如何使用 Socket.IO 类库实现 WebSocket 通信等；第三部分（第 16 章）讲解了两个综合案例，如何结合使用 Node.js 与 Socket.IO 类库制作一个聊天室应用程序的服务器端及客户端，以及如何结合使用 Node.js 与 Express 框架制作一个 Web 应用程序的服务器端及客户端。

Node.js 权威指南

陆凌牛 著

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：罗词亮
印 刷：北京市荣盛彩色印刷有限公司
开 本：186mm × 240mm 1/16
书 号：ISBN 978-7-111-46078-7
版 次：2014 年 4 月第 1 版第 1 次印刷
印 张：35
定 价：89.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066
购书热线：（010）68326294 88379649 68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

前　言

为何写作本书

近年来，随着智能手机与 HTML 5 的不断普及，JavaScript 脚本语言的重要性也随之不断提升，IT 业界涌现了大量学习与善用 JavaScript 脚本语言的工程师，其中许多工程师对任何服务器端开发语言均一无所知。很多工程师提出，如果能够让服务器端与客户端均使用一种脚本语言，则无疑可以减少服务器端的开发难度，提高服务器端的开发效率。另一方面，由于近几年许多 JavaScript 引擎中均内置了 JIT（Just In Time）编译器，使 JavaScript 引擎的处理速度得到了大幅度提高，JavaScript 脚本语言的运行速度不会逊色于任何服务器端开发语言。

据此现状，2009 年 8 月，IT 业界制定了 CommonJS 标准，用于标准化服务器端 JavaScript 脚本语言，制定服务器端 JavaScript 脚本语言中所需要实现的处理。

同年，美国人 Ryan Dahl 推出了第一个遵循 CommonJS 标准的服务器端 JavaScript 脚本语言开发框架——Node.js。在 Node.js 内部，运行的是 Google 开发的高性能 V8 JavaScript 脚本语言，该语言可以运行在服务器端。Node.js 的一个最重要的特性是通过单线程实现异步处理环境。通常，提及异步处理，开发者们首先会联想到的是服务器端多线程环境，在 Node.js 中，通过事件环与非阻塞型 I/O 机制实现服务器端的异步处理。

为了帮助国内的 Web 开发者更好地学习 Node.js 开发框架，笔者特此推出本书，希望国内的 Web 开发者们能够通过学习本书尽早地运用 Node.js 框架开发出高效的 Web 服务器以及运行于该 Web 服务器中的 Web 应用程序。

读者对象

根据不同使用需要，本书适用于如下读者：

- 对 Web 网站或 Web 应用程序的开发技术感兴趣或者打算从事 Web 网站或 Web 应用程序开发的技术人员。
- Web 网站或 Web 应用程序的开发者（包括 Web 前端开发工程师及后端开发工程师）。
- 有关 Web 网站或 Web 应用程序开发项目的项目管理人员。
- 开设相关课程的大专院校及培训机构。

如何阅读本书

本书内容分三大部分展开。

第一部分： 第 1 章详细阐述什么是 Node.js 框架，为什么要选择 Node.js 框架进行服务器端的开发，使用 Node.js 框架能够解决什么问题，Node.js 框架适合用于开发哪些应用程序，如何下载及使用 Node.js 框架，Node.js 框架的主要特性，使用 Node.js 框架时必须了解的基础知识。第 2 章 ~ 第 12 章针对 Node.js v0.10 版中的各模块进行展开阐述，详细阐述这些模块的作用，如何使用这些模块，这些模块中所提供的各对象、属性、方法及事件。

第二部分： 第 13 章 ~ 第 15 章分别阐述在使用 Node.js 框架进行 Web 服务器端的开发时极有可能利用到的第三方开发包，包括如何在 Node.js 应用程序中使用关系型数据库及 NoSQL 型数据库，如何使用 Express 框架开发 Web 应用程序，如何使用 Socket.io 类库实现 WebSocket 通信。

第三部分： 第 16 章介绍两个综合案例，在第一个案例中，我们讲述如何结合使用 Node.js 与 Socket.io 类库制作一个聊天室应用程序的服务器端及客户端，在第二个综合案例中，我们讲述如何结合使用 Node.js 与 Express 框架制作一个 Web 应用程序的服务器端及客户端。

在本书的每一章中，每一个正在阐述的理论点均使用代码实例进行具体形象地说明，每个实例中所涉及的理论知识都以通俗易懂的语言进行阐述，大部分实例均使用图片来形象说明该实例的运行效果。本书所有实例代码都经笔者亲自测试运行成功，提供给读者学习使用。每个实例的详细代码及其使用到的脚本文件、各种资源文件都可在华章公司网站（www.hzbook.com）的本书页面上下载。读者可以对这些代码进行修改，以便观察各种不同效果，加深对实例代码的理解。

勘误和支持

由于时间及水平方面的原因，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。笔者 QQ 号码为 240824399，联系邮箱为 240824399@qq.com。欢迎读者通过 QQ 或 E-mail 与笔者联系，真诚期待能够听到读者的反馈意见。

致谢

感谢机械工业出版社华章公司的编辑杨福川和姜影，感谢杨福川老师的魄力和远见，感谢姜影老师的细心编辑与校对，感谢二人在这半年多的时间中始终支持我的写作，是他们的鼓励和帮助引导我顺利完成全部书稿。

谨以此书献给众多热爱 Node.js 开发框架的朋友们，以及中国 IT 界从事 Web 网站及 Web 应用程序开发的全体同行。

目　录

第 1 章
Node.js 介绍

最近几年，Web 领域出现了一个全新的 JavaScript 开发框架——Node.js。该框架一经问世，便以其独特的优势得到了广大开发人员的关注。本章先来对 Node.js 框架作一下概要介绍。

本章内容包括：

- 什么是 Node.js 框架，为什么要用 Node.js 框架，使用 Node.js 框架能够解决什么问题，在哪些场合下应该考虑使用 Node.js 框架。
- 如何下载 Node.js 框架。
- 什么是 Node.js 框架中的模块，Node.js v0.10 版中内置了哪些模块以及这些模块的作用。
- 如何开发一个最简单的 Node.js 示例应用程序，以及如何运行这个示例应用程序。

1.1 Node.js 概述

1.1.1 使用 Node.js 能够解决什么问题

Node.js 的首要目标是提供一种简单的、用于创建高性能服务器及可在该服务器中运行的各种应用程序的开发工具。首先让我们来看一下现在的服务器端语言中存在着什么问题。在 Java、PHP 或 ASP.NET 等服务器端语言中，为每一个客户端连接创建一个新的线程，而每个线程需要耗费大约 2MB 的内存，也就是说，理论上，具有 8GB 内存的服务器可以同时连接的最大用户数为 4000 个左右。要让 Web 应用程序支持更多的用户，就需要增加服务器的数量，而 Web 应用程序的硬件成本也就随之增加了。不仅如此，在技术层面也会因此产生一些潜在的问题。例如，由于同一个用户的不同客户端请求可能会被不同的服务器处理，因此必须在所有的服务器之间共享所有的资源。由此可见，在一个 Web 应用程序中，一个主要的瓶颈是服务器所支持的最大同时连接用户量。

Node.js 修改了客户端到服务器端的连接方法，解决了这个问题。因为它并不为每个客户端连接创建一个新的线程，而是为每个客户端连接触发一个在 Node.js 内部进行处理的事件。因此，如果使用 Node.js，可以同时处理多达几万个用户的客户端连接。因此，当需要使 Web 应用程序能够支持大量用户的并发连接的时候，我们应该考虑使用 Node.js。

1.1.2 实现高性能服务器

严格地说，Node.js 是一个用于开发各种 Web 服务器的开发工具。在 Node.js 服务器中，运行的是高性能 V8 JavaScript 脚本语言，该语言是一种可以运行在服务器端的 JavaScript 脚本语言。

那么，什么是 V8 JavaScript 脚本语言呢？该语言是一种被 V8 JavaScript 引擎所解析并执行的脚本语言。V8 JavaScript 引擎是由 Google 公司使用 C++ 语言开发的一种高性能 JavaScript 引擎，该引擎并不局限于在浏览器中运行。Node.js 将其转用在了服务器中，并且为其提供了许多附加的具有各种不同用途的 API。例如，在一个服务器中，经常需要处理各种二进制数据。在 JavaScript 脚本语言中，只具有非常有限的对二进制数据的处理能力，而 Node.js 所提供的 Buffer 类则提供了丰富的对二进制数据的处理能力。

另外，在 V8 JavaScript 引擎内部使用一种全新的编译技术。这意味着开发者编写的高端的 JavaScript 脚本代码与开发者编写的低端的 C 语言具有非常相近的执行效率，这也是 Node.js 服务器可以提供的一个重要特性。

1.1.3 非阻塞型 I/O 及事件环机制

为了实现高性能，Node.js 中采用了以下两种机制：

- 非阻塞型 I/O
- 事件环

JavaScript 脚本语言的一个特征是它只支持单线程。V8 JavaScript 脚本语言也是如此，

因此不需要担心它会造成死锁现象。但是与客户端脚本语言不同的是，Node.js 中为 V8 JavaScript 脚本语言提供了非阻塞型 I/O 机制。例如，当在访问数据库取得搜索结果的时候，在开始访问数据库之后、数据库返回结果之前，存在一段等待时间。在传统的单线程处理机制中，在执行了访问数据库的代码之后，整个线程都将暂停下来，等待数据库返回查询结果之后才能继续执行其后面的代码。也就是说，I/O 操作阻塞了代码的执行，极大地降低了程序的执行效率。由于 Node.js 中采用了非阻塞型 I/O 机制，因此在执行了访问数据库的代码之后将立即转而执行其后面的代码，把数据库返回结果的处理代码放在回调函数中执行，从而提高了程序的执行效率。

另外，在 Node.js 中，虽然不具有因为用户鼠标单击按钮或输入文字等操作而触发的事件，但是具有由于客户端请求建立连接、通过此连接而接收客户端提交数据、停止客户端提交数据的接收等行为而触发的事件。在 Node.js 中，在一个时刻只能执行一个事件回调函数，但是在执行一个事件回调函数的中途可以转而处理其他事件（包括触发新的事件、声明该事件的回调函数等），然后返回继续执行原事件回调函数，这种处理机制称为事件环机制。

1.1.4 Node.js 适合开发的应用程序

到目前为止，我们对 Node.js 进行了相关的概要介绍，那么，Node.js 适合用来开发何种应用程序呢？回答是：当应用程序需要处理大量并发的输入 / 输出，而在向客户端发出响应之前，应用程序内部并不需要进行非常复杂的处理的时候，我们应该考虑使用 Node.js 来进行该应用程序的开发。例如，我们可以开发如下应用程序：

- 聊天服务器：在一个具有很高人气的聊天应用程序中，在同一时刻通常可能存在着大量用户与聊天服务器之间的并发连接，而服务器端本身并不存在非常复杂的处理。
- 综合服务类网站或电子商务网站的服务器：在这类网站的服务器端，往往可能在每秒钟内接收到多达上千条数据并且需要将这些数据书写到数据库中，Node.js 是解决这类问题的关键。Node.js 将通过其队列机制将这些数据迅速书写在缓存区中，然后再通过每一个单独的处理从缓存区中取出这些数据并将其书写到数据库中。如果使用其他服务器（如 Apache 服务器或 Tomcat 服务器）的话，由于这些服务器采用的均为阻塞型 I/O 机制，因此需要为每条数据（到数据库中）的写入等待一段时间。如果使用 Node.js 服务器，由于其采用的是非阻塞型 I/O 机制，因此可以同时实现这些数据到数据库中的写入，而不必再为每条数据的写入等待一段时间。

1.2 安装 Node.js

Node.js 官网（网址为 http://nodejs.org/）上提供了 Windows 与 Mac 版本的安装程序，以及 Linux 版本的源代码。执行安装程序即可将 Node.js 安装在我们的计算机中，在 Windows 操作系统中的默认安装路径为 C:\Program Files\nodejs，执行文件为 node.exe；在 Mac 操作系统中的默认安装路径为 /usr/local/bin，执行文件为 node。

在 Linux 操作系统中，需要对源代码进行编译。在安装了 GCC 之类的编译环境后，可

以通过如下所示的步骤来安装 Node.js。

```
$ wget http:// nodejs.org/dist/v0.10.15/node-v0.10.15.tar.gz
$ tar zxvf node-v0.10.15.tar.gz
$ cd node-v0.10.15
$ ./configure
$ make
# make install
```

最后一行中的“make install”需要通过使用“root”用户来运行，默认安装路径为 /usr/local/bin，执行文件为 node。

目前（2014 年 2 月），Node.js 的最新稳定版本为 0.10.24。

1.3 Node.js 中的模块

接下来，我们将要介绍如何制作一个最简单的 Node.js 应用程序，在介绍该应用程序之前，首先概要介绍一下什么是 Node.js 中的模块。

我们知道，在客户端可以将所有 JavaScript 代码分割为几个 JS 文件，然后在浏览器中将这些 JS 文件合并运行。但是在 Node.js 中是通过以模块为单位来划分所有功能的。每一个模块为一个 JS 文件。每一个模块中定义的全局变量或函数的作用范围也被限定在这个模块之内，只有使用 exports 对象才能将其传递到外部。代码如下所示：

```
exports.printFoo = function(){ return "foo" }
```

在上面这行代码中，我们定义了一个 printFoo 函数，函数内部返回“foo”字符串。同时，我们通过使用 exports 对象使模块外部可以访问这个 printFoo 函数。

在引用模块时，我们需要使用 require 函数。例如，将上面这个具有 printFoo 函数的模块文件保存为 foo.js 文件之后，可以通过以下代码来访问模块中的 printFoo 函数：

```
var foo = require('./foo.js');          // 通过 foo.js 文件路径加载 foo.js 模块
console.log(foo. printFoo());           // 访问 foo.js 模块内的 printFoo 函数
```

在第一行代码中，我们读取 foo.js 模块并通过 foo 对象来引用该模块，在第二行代码中，我们通过 foo 对象来访问模块内的 printFoo 函数，控制台窗口中将输出“foo”字符串。

在 Node.js 中，提供了一些核心模块，其中 Node.js v0.10 版中的模块及其作用见表 1-1。

表 1-1 Node.js v0.10 版中的核心模块

模 块 名	模块功能	Stability（稳定度）
assert	为应用程序的单元测试添加断言处理	5-Locked（今后不会被修改）
buffer	用于实现二进制数据的存储与转换	3-stable（稳定）
child_process	用于实现子进程的创建与管理	3-stable（稳定）
cluster	用于实现多进程	1-Experimental（实验性阶段）
console	用于为控制台输出信息	4-API Frozen（今后 API 部分不会被修改）

（续）

模 块 名	模 块 功 能	Stability（稳定度）
crypto	用于实现数据的加密解密处理	2-Unstable（不稳定）
debugger	用于实现一个内置调试器来帮助开发者调试应用程序	3-stable（稳定）
dns	用于实现与 DNS 相关的处理	3-stable（稳定）
domain	用于实现多个 I/O 之间的协作处理	2-Unstable（不稳定）
events	用于为事件处理提供一个基础类	4-API Frozen（今后 API 部分不会被修改）
fs	用于操作文件及文件系统	3-stable（稳定）
http	用于实现 HTTP 服务器端及客户端	3-stable（稳定）
https	用于实现 HTTPS 服务器端及客户端	3-stable（稳定）
net	用于创建 TCP 服务器与客户端	3-stable（稳定）
os	用于获取操作系统信息	4-API Frozen（今后 API 部分不会被修改）
path	用于处理文件路径	3-stable（稳定）
punycode	用于实现 Punycode 字符串的编码及解码	2-Unstable（不稳定）
querystring	用于处理 HTTP 请求中使用的查询字符串	3-stable（稳定）
readline	用于读取一行标准输入流	2-Unstable（不稳定）
repl	用于实现 REPL（Read-Eval-Print-Loop）交互式运行环境	（无）
stream	用于为流的输入 / 输出处理提供一个基础类	2-Unstable（不稳定）
string_decoder	用于实现从二进制数据到字符串数据之间的转换	3-stable（稳定）
tls	用于使用 OpenSSL 来实现 TLS/SSL 通信处理	3-stable（稳定）
tty	用于获取来自于 TTY 终端的信息	2-Unstable（不稳定）
url	用于实现 URL 字符串的解析与格式化	3-stable（稳定）
util	用于实现各种实用函数	5-Locked（今后不会被修改）
vm	用于为 JavaScript 脚本代码提供一个独立的运行环境	2-Unstable（不稳定）
zlib	内部使用 zlib 类库来实现数据的压缩及解压处理	3-stable（稳定）

在 Node.js 中，可以直接使用 require 函数并将模块名设置为 require 函数的参数值的方法来引用这些模块。例如，可以使用如下代码来引用 http 模块：

```
var http = require('http');
```

在这段代码中，使用 require 函数来引用 http 模块，该函数返回 http 模块对象，将该对象赋值给 http 变量之后即可以通过 http 变量来访问 http 模块中的属性及方法。

在 Node.js 框架中，除了可以使用 V8 JavaScript 引擎中所支持的、ECMAScript 5 中定义的函数与类之外，追加了一些类、函数与对象。可以在不引用任何模块的情况下直接使用这些类、函数与对象。Node.js 中追加的类、函数与对象见表 1-2。

表 1-2 Node.js 中追加的类、函数与对象

类、函数及对象名	描 述
Buffer 类	用于为二进制数据的存储提供一个缓存区
setTimeout 函数	用于在指定时间到达时执行一个指定函数，指定方法为从当前时刻之后多少毫秒
clearTimeout 函数	用于取消在 setTimeout 函数内指定的函数的执行