一线Node.js开发者数年实战**经验总结**，借助源码分析，深入浅出地介绍**Node技术栈全貌**

Node.js 进阶之路

尤嘉◎编著

清華大學出版社

Node.js 进阶之路

尤嘉◎编著

清華大學出版社
北京

内 容 简 介

本书内容涵盖了 Node.js 高并发的原理、源码分析以及使用 Node.js 开发应用所需要的不同层面的技术实践。具体来讲，本书包括 Node.js 异步机制（配以源码分析）、编辑与调试、测试技术、Docker 部署、模块机制、V8 引擎与代码优化、Promise 和 ES6 Generator、LoopBack 开源框架、使用 C++ 编写扩展、JavaScript 严格模式、编码规范等内容。在 LoopBack 章节，本书详细介绍了使用此框架开发企业级 Web 应用的步骤，帮助读者迅速掌握使用这个强大框架的诀窍。最后一章详细介绍了编写不同类型的 C++ 模块的知识，并对堆内存管理等内容做了深入探讨。

本书适合所有前端和后端的开发人员阅读。

图书在版编目(CIP)数据

Node.js进阶之路 / 尤嘉 编著. — 北京：清华大学出版社，2017
ISBN 978-7-302-45693-3

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

中国版本图书馆 CIP 数据核字（2016）第 289152 号

责任编辑： 袁金敏
封面设计： 刘新新
责任校对： 徐俊伟
责任印制： 李红英

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： 清华大学印刷厂
经　　销： 全国新华书店
开　　本： 185mm×230mm　　**印　　张：** 12.75　　**字　　数：** 211 千字
版　　次： 2017 年 1 月第 1 版　　**印　　次：** 2017 年 1 月第 1 次印刷
印　　数： 1～3500
定　　价： 35.00 元

产品编号：071875-01

前 言

本书写给那些打算或者正在使用 Node.js（简称Node，后文均用此简称）创建 Web 应用的开发者。众所周知，JavaScript 的灵活易用以及 V8 引擎的加速，再加上活跃的社区支持，使得用 Node 开发应用的成本低，收益大。2015 年 ES6 标准的确立，为JavaScript成为企业级开发语言扫除了不确定性。这本书的选材契合这个领域最新的技术进展，深浅适宜地介绍了 Node 技术栈的全貌。

本书共分9章。第1章概述，介绍 Node 异步实现的原理，涵盖了 Node 实现异步的两种方式。这部分引用了 Node 源码，以求逻辑清晰与内容翔实。第2章～第7章是站在 JavaScript 的角度，介绍了用 Node 开发应用的方方面面，包括编辑与调试、测试技术、Docker 部署、模块机制、V8 引擎与代码优化、Promise 和 ES6 generator 等内容。第8章介绍了 LoopBack 开源框架的使用。本书没有介绍 Express（可能读者早已熟悉），因为本书希望为读者引荐一个更加强大易用的企业级 Web 框架。第9章则从 C++ 的角度介绍了 Node 扩展模块的编写，这部分适合那些想要了解 V8 引擎的读者。可以说 C++ 是 Node 技术栈的基石。本书希望向读者呈现构成 Node 技术栈的JavaScript和C++ 全貌。

本书不假定读者有 Node 研发经验，但需熟悉 JavaScript。如果读者最近才接触编程，建议选一本更初级的教程，或者先到 W3School（http://www.w3school.com.cn/js/index.asp）上看看。本书每一章都有源码示例，这些示例大部分可以在 Node 支持的任何系统上运行，但也有例外。建议使用本书第3章介绍的容器，在 Linux 环境下运行本书示例。大部分示例代码可以从 https://github.com/classfellow/node-AdProgramming 下载。

饮半盏湖水，当知江河滋味；拾一片落叶，尽享人间秋凉。希望本书成为读者熟练掌握 Node 技术栈的那一盏湖水、一片落叶。

致谢

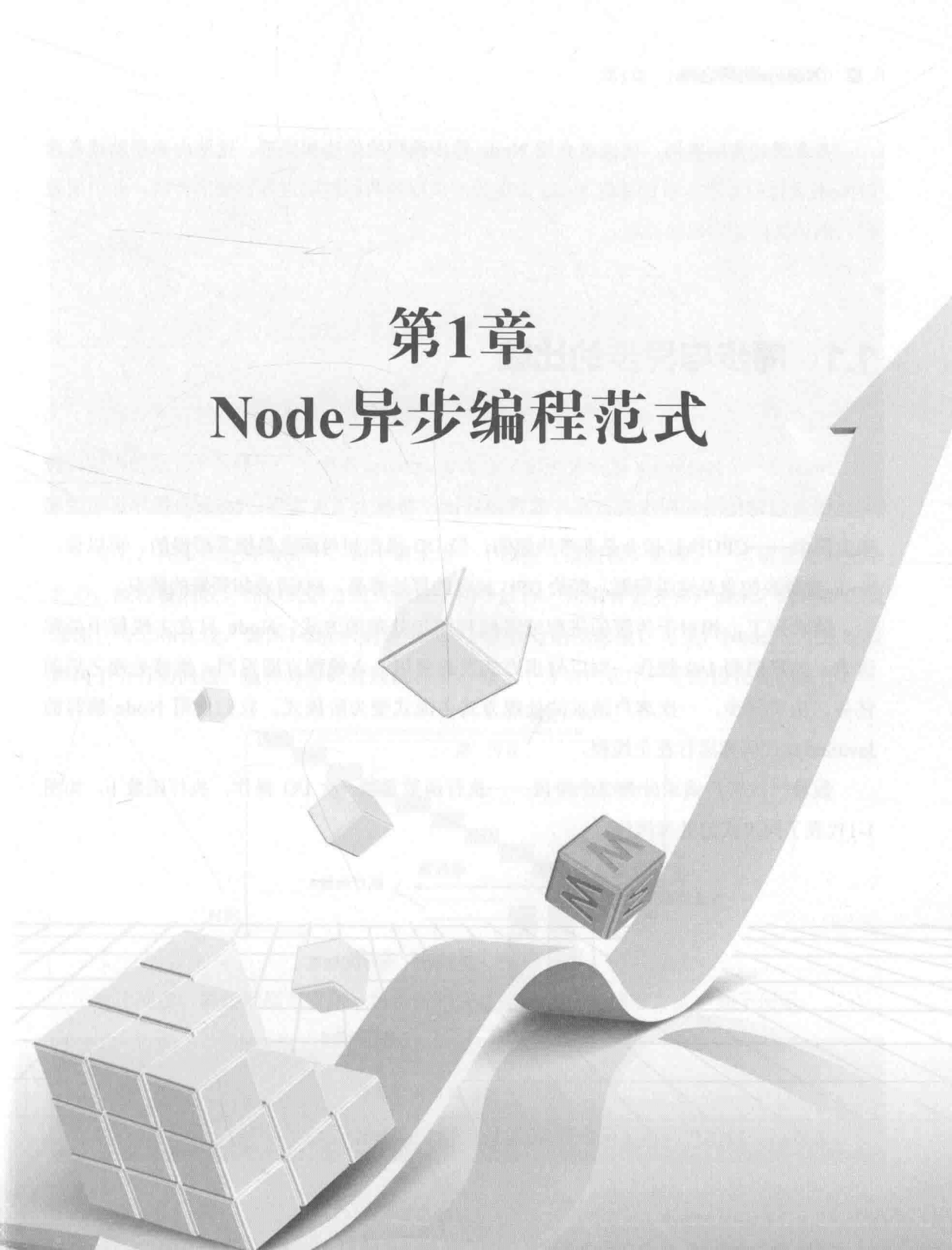
感谢 CNode 社区，它提供了一个非常好的平台，本书前期的一些章节从中得到了积极的反馈，使笔者有了继续写下去的动力。首都师范大学的刘晓莲同学，利用周末时间审阅了本书的稿件，提出的一些见解，使得本书在内容安排上更合理，更容易看懂，在此表示感谢。笔者周围的一些同事部分地阅读了初稿并给出了积极的反馈，在此一并谢过！

作者邮箱

classfellow@qq.com

目 录

第1章
Node异步编程范式



本章通过实际案例，向读者介绍 Node 异步编程的优势和原理，这些内容帮助读者理解Node运行的本质。本章还就 Node 实现异步调用的两种机制进行详细的介绍，并引用源码，剖析其内部实现的流程。

1.1 同步与异步的比较

Node是一个JavaScript 运行时环境，它为 JavaScript 提供了一个异步 I/O 编程框架，较其他语言通常使用的同步式方案，其性能好比“搭载上了火箭”。Node的指导思想说起来也简单——CPU执行指令是非常快速的，但 I/O 操作相对而言是极其缓慢的。可以说，Node 要解决的也是这类问题，即给 CPU 执行的算法容易，I/O请求却频繁的情况。

请求到了，相对于传统的进程或者线程同步处理的方式，Node 只在主线程中处理请求。如果遇到 I/O 操作，则以异步方式发起调用，主线程立即返回，继续处理之后的任务。由于异步，一次客户请求的处理方式由流式变为阶段式。我们使用 Node 编写的 JavaScript 代码都运行在主线程。

假设一次客户请求分为三个阶段——执行函数 a，一次 I/O 操作，执行函数 b。如图1-1代表了同步式的处理流程。

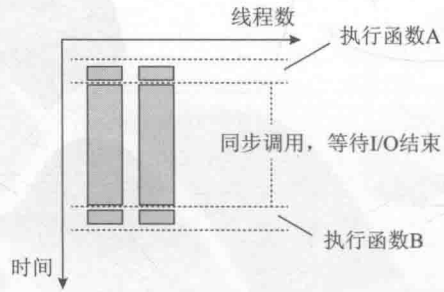


图1-1 同步式的处理流程

可以用一段伪代码描述同步请求的过程，如下所示：

```
// 代码 1-1
function request() {
  //开始执行函数 a
  $rel_a = stage_a();
  //读文件，将文件内容返回到 $data
  $data = readfile();
  //将前两步的结果作为参数，调用函数 b
  stage_b($rel_a, &data);
}
```

可见，同步式处理方式中，每个请求用一个线程（或进程）处理。一次请求处理完毕之后，线程被回收。同步式的方式只画出了两个线程，如果有更多客户请求，线程数还要增加。与之相比较，如图1-2所示则是 Node 异步执行的示意图。可见，Node 一个主线程解决了所有的问题。这种异步式处理流程中，每一个方块代表了一个阶段任务的执行。

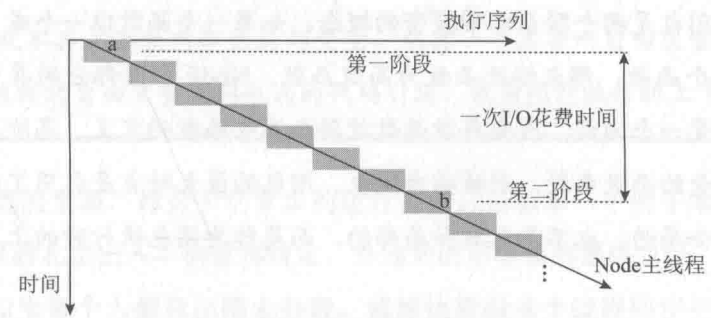


图1-2 Node异步执行示意图

对比同步，异步同样可以用一段伪代码表达Node异步的处理方式，如下所示：

```
// 代码 1-2
var request = function(){
  //开始执行函数 a
```

```
  var rel_a = stage_a();
  //发起异步读取，主线程立即返回，处理之后的任务
  readfileAsync(function(data){
    //在随后的循环中，执行回调函数
    stage_b(rel_a, data);
  });
}
```

Node也“站在巨人的肩上”。这个“巨人”是大名鼎鼎的 V8 引擎，有这样一个强大的“心脏”，再配合基于高阶函数和闭包的异步编码范式，使得用 Node 构建的程序在性能上有着出色的表现。

高阶函数与闭包是两个联系非常紧密的概念。如果一个函数以一个或多个函数作为参数，或者返回一个函数，那么称此函数为高级函数。Node 中大部分的异步函数，参数列表的最后一项接受一个回调，这类异步函数就符合高阶函数的定义。高阶函数执行后返回的函数，或者接受的函数参数，则被称为闭包。闭包的最大特点是引用了它之外的变量，这些变量既不是全局的，也不是参数和局部的，而是作为闭包执行时的上下文环境存在。如下所示：

```
// 代码 1-3
function wrapper(price){
  var freeVal = price;
  function closure (delta){
      return freeVal * delta;
```

```
  }
  return closure;
}
var clo1 = wrapper(100);
var clo2 = wrapper(200);
setTimeout(function(){
  console.log(clo1(1));
  console.log(clo2(1));
}, 500);
```

代码1-3运行后输出的结果如下：

```
100
200
```

函数 wrapper 是一个高阶函数，执行后返回一个闭包，这个闭包将 price 纳入自己的作用域，price 就不再是函数内部的局部变量，它有一个名字叫自由变量，其生命期与闭包绑定。price 这样的自由变量被闭包内的代码引用，成为闭包执行的上下文。

Node高性能的来源，得益于它异步的运行方式。可以举一个例子来理解异步对性能的提升。按照目前北京出入车辆管理规定，外地来的车需要办理进京证，而办进京证需等待一定时间。如果每个人都自己跑去办理，就好比开启多个线程同步处理，办理窗口有限，就得排队。而把这件事委托给第三方，就好比不开启线程或进程，把耗时的I/O请求委托给操作系统。这种情况下人们从办证的任务中解放出来，因而能继续做其他事情。若来了一个任务，交给第三方去处理，则第三方就有一个接单队列，其只需拿着所有的接单，去办理地点逐个办理即可。

如图1-3所示是办理进京证的示意图。假设有20个人，每人开车来回花费2小时，往返油费60元，办理窗口有两个，在窗口办完一个进京证平均需要5分钟。不考

虑排队时间，则20个人办证花费的总时间至少是20×2 + 20×5÷60小时，总油费是20×60元。按照目前北京市最低工资标准推断，假设有车族时薪为100元，则总成本为(20×2+20×5÷60)×100+20×60元，这个数字大概是5367元。下面来对比一下异步办理进京证的花费。

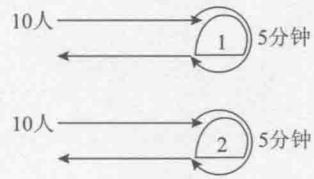


图 1-3　同步办理进京证示意图

客户将资料交给代理者，此人拿着 20 个客户的资料，一个人办理完所有事情。同样假设此代理者的时薪为 100元，则总花费为(20×5÷60)×100+60元，这个数字大概是 227元。还应该注意到，这种方式还省去了一个办理窗口，如图1-4所示。

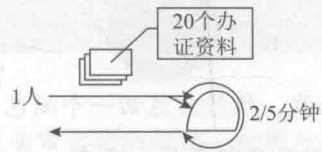


图 1-4　异步办理进京证示意图

在上面的讨论中，将任务委托给第三方，至少涉及两个细节，一是上下文，二是后续动作如何对接。以办证为例，客户需要把自己的一些资料交给代理人。办完之后，代理人需要有一种方式将结果交给客户。在同步办证逻辑中，所谈的这两个细节是不需要特别提出的。人们排队去窗口办理，轮到的人，说明来意，交出资料，耐心等待；办证窗口里面的工作人员办理好后，把新证交给窗口处的等待者即可。但在异步办证的逻辑中，所谈的这两个细节就需要认真考虑了。代理者去一个窗口办理，因为办证处仍然具有同时处理两个请求的能力，所以代理者同时交出两个客户资料请求办理。假设这个代理人把 A 和 B 的资料交给窗口，只过了 4 分钟，就办好了一个，此人需要判断此证属于 A 还是 B，不能搞混了，否则他后续的步骤将会出错。这种因为异步带来的返回结果次序的不确定性，是异步编程框架需要解决的一个问题。

假设有一个异步读取文件函数 readAsync，在主线程中，调用此函数发起了一个异步读取操作。因为执行的是异步函数，所以主线程立即返回，继续处理其他任务。操作系统执行完具体的读取操作后，将数据准备好，通知主线程。这个过程需要解决的一个问题是，当主线程得到通知后，如何识别异步请求是谁发起的，以及得到数据之后下一步该做什么。前面讲到了闭包，这里恰恰就需要闭包的特性对接执行流程。因为闭包保留了异步调用发起时的上下文信息，于是执行闭包，将结果传入，就可沿着发起读取文件时的执行环境继续运行后续逻辑。

设想如果来了1000个并发请求，按照创建进程或线程的同步处理方式，一个服务器运行这么多进程或者线程，使CPU频繁地在上下文切换是多么低效。CPU 和内存资源的内耗挤压了真正花费在处理业务逻辑上的时间，造成服务器性能下降，但假如服务器少开线程，则又会造成请求排队等待。而在异步方式下，不需要开多余的线程，所有请求均由主线程承担。一旦遇到耗时的 I/O 请求，以异步的方式发起调用。一次异步调用同时也会创建一个闭包传进去，操作系统执行具体的 I/O 操作，将结果放入指定缓存，然后将本次 I/O 置为就绪状态。主线程在每一次循环中，收集就绪的请求，取出原先的闭包，然后调用回调函数并将结果传入。这个过程不需要创建任何多余的线程或进程。在1.2节，将以实例从脚本和 C++ 层面描述这个过程。

Node能够最大限度利用硬件资源，其原因即如此。换句话说，CPU 把绝大多数时间花在处理实际业务逻辑上，而不是线程或进程的等待和上下文切换上。在这种处理模式下，假如主线程阻塞，那说明真的是没有任务需要处理，而不是等待 I/O 结束。

1.2 Node异步的实现

本节从一次HTTP请求来跟踪Node内部的执行过程，读者可以从中了解 Node 是如何工作的，然后再将网络 I/O 与本地磁盘 I/O 做对比，指出 Node 实现异步调用的两种机

制。读者在阅读本节时，可以比照着 Node 源码来看，以加深理解。

1.2.1 HTTP请求——完全异步的例子

HTTP是一个应用层协议，在传输层使用TCP协议与服务器进行数据交互。一次HTTP请求分四个阶段，分别是连接、请求、应答、结束。如下代码是一段发起HTTP请求的代码，下面就以这段代码为例，看看 Node 是如何以异步的方式完成上述四个过程的。

```
// 代码 1-4
"use strict";
var http  = require('http');
var options = {
  host: 'cnodejs.org'
  ,port: 80
  ,path: '/'
  ,method: 'GET'
  ,headers: {
    'Content-Type': 'application/json'
  }
};
var req = http.request(options);
req.once('response', (res)=>{
  var result = '';
  res.on('data', function (chunk) {
    result += chunk.toString();
  });
```