第一篇

Node.js概述和开发环境的搭建

第 1 章

◂Node.js介绍▸

Node.js 是一个基于 JavaScript 的跨平台开发语言。随着全栈开发技术的不断推广和日益盛行，Node.js 逐渐成为一种非常流行的开发语言。本章主要对 Node.js 进行整体介绍，并对其发展历史和相关版本进行详细的说明，同时会介绍在今后开发中所涉及的基础知识。

通过本章的学习可以掌握以下内容：

- Node.js 的发展历史和特点。
- V8 引擎的介绍及其与 Node.js 之间的关系。
- Node.js 的一些应用场景。
- Node.js 在中国的发展及相关资源。

1.1 Node.js 简介

Node.js 是一个基于 Google 所开发的浏览器 Chrome V8 引擎的 JavaScript 运行环境。Node.js 使用多种先进的技术，其中包括事件驱动和非阻塞式 I/O 模型，使其轻量又高效，受到众多开发者的追捧。

简单来说，Node.js 就是运行在服务端的 JavaScript，可以稳定地在各种平台下运行，包括 Linux、Windows、Mac OS X、SunOS 和 FreeBSD 等众多平台。

作为 Web 前端最重要的语言之一，JavaScript 一直是前端工程师的专利。不过，Node.js 是一个后端的 JavaScript 运行环境（支持的系统包括 Linux、Windows），这意味着我们可以编写系统级或者服务器端的 JavaScript 代码，交给 Node.js 来解释执行。

简单的 Node.js 命令类似于：

```
#node  helloworld.js
```

由于采用 V8 引擎执行 JavaScript 的速度非常快，因此 Node.js 开发出来的应用程序性能非常好。Node.js 已经成为全栈开发的首选语言之一，并且从它衍生出众多出色的全栈开发框架。Node.js 已经在全球被众多公司使用，包括创业公司 Voxer、Uber 以及沃尔玛、微软这样的知名公司。它们每天通过 Node 处理的请求数以亿计，可以说在要求苛刻的服务器系统，Node.js 也可以轻松胜任。

Node.js 还包括一个完善的社区。在 Node.js 的官方网站（http://nodejs.org/）可以找到大量的文档和示例程序，并且 Node.js 还有一个强大的包管理器 NPM。渐渐地，越来越多的人参与到本项目中来，可用的第三方模块和扩展增长迅猛，而且质量也在不断提升，Node.js 已经是全球较大的开源库生态系统之一。

Node.js 不是一个 JavaScript 应用，而是一个 JavaScript 的运行环境，由 C++语言编写而成。

1.2 Node.js 的发展历史和特点

任何语言或框架都不是一天形成的，而是经过漫长的测试、发布、再测试、再发布的迭代过程。本节就来介绍一下 Node.js 的发展过程。

1.2.1 Node.js 发展历史

Node.js 的创始人是大名鼎鼎的 Ryan Dahl。他本来是学数学的，2008 年年末一个偶然的机会让他了解到 Google 推出了一个新的浏览器 Chrome 和崭新的 JavaScript 引擎 V8。他听说这是一个为了更快的 Web 体验而专门制作的更快的 JavaScript 引擎，V8 能够让 Web 应用大大提速。当时，他正在寻找一个新的编程平台来做网站，他非常希望能找到一种语言提供先进的推送功能并集成到网站中，而不是采用传统的方式——不断轮询拉取数据。

Ryan Dahl 对 C/C++和系统调用非常熟悉，他使用系统调用（用 C）实现消息推送这样的功能。如果只使用非阻塞式 Socket，每个连接的开销都会非常小。在小规模测试中，它能同时处理几千个闲置连接，并可以实现相当大的吞吐量。但是，他并不想使用 C，他希望采用另一种漂亮、灵活的动态语言。他最初也希望采用 Ruby 来写 Node.js，但是后来发现 Ruby 虚拟机的性能不能满足要求，后来便尝试采用 V8 引擎，所以选择了 C++语言。

2009 年 2 月，Ryan Dahl 首次在自己的博客上宣布准备基于 V8 创建一个轻量级的 Web 服务器并提供一套库，并在 2009 年 5 月正式在 GitHub 上发布最初版本的部分 Node.js 包。随后几个月里，有人开始使用 Node.js 开发应用。实践证明，JavaScript 与非阻塞 Socket 配合得相当完美，只需要简单的几行 JavaScript 代码就可以构建出非常复杂的非阻塞服务器。

2010 年年底，Node.js 获得云计算服务商 Joyent 资助。创始人 Ryan Dahl 加入 Joyent，全职负责 Node.js 的发展。Node.js 从此以后迅猛发展，并成为一种流行的开发语言。

在官方网站上，Node.js 的版本号是从 0.1.14 开始的，每个发布版本对应不同的 V8 引擎版本和 NPM 包管理器版本，截至作者写作本书时，最新的版本为 V10.9.0，其各个主要版本的具体发布时间参见表 1-1（2014 年以后）。

表 1-1　Node.js 版本的发布时间（2014 年以后）

版本	日期
Node.js V10.9.0	2018-08-15
Node.js v9.11.2	2018-06-12
Node.js v8.10.0	2018-03-06
Node.js v7.10.1	2017-07-11
Node.js v6.11.0	2017-06-06
Node.js v6.0.0	2016-04-26
Node.js v5.0.0	2015-10-29
Node.js v4.2.0	2015-10-12
Node.js v4.0.0	2015-09-08

从 1.x 到 3.x 版本，Node.js 的名称曾经被修改为 io.js，从 Node.js 4.0.0 开始，io.js 的全部代码就合并到 Node.js 的主干发布版本之中了。

Node.js 的发展大致可以分为以下 4 个阶段。

（1）发展初期。创始人 Ryan Dahl 带着他的团队开发出以 Web 为中心的 Web.js，一切都非常混乱，API 大多处于研究阶段。

（2）快速发展时期。Node.js 的核心用户 Isaac Z. Schlueter 开发出奠定 Node.js 如今地位的重要工具——NPM，同时也为他后来成为 Ryan 的接班人奠定了基础。之后 Connect、Express、Socket.io 等库的出现吸引了一大波爱好者加入 Node.js 开发者的阵营中来。CoffeeScript 的出现更是让不少 Ruby 和 Python 开发者找到了学习的理由。其间一大波以 Node.js 作为运行环境的 CLI 工具涌现，其中不乏用于加速前端开发的优秀工具，如 Less、UglifyJS、Browserify、Grunt 等。这个阶段 Node.js 的发展势如破竹。

（3）不稳定时期。经过一大批一线工程师的探索实践后，Node.js 开始进入时代的更迭期，新模式代替旧模式，新技术代替旧技术，好实践代替旧实践。ES 6 也开始出现在 Node.js 世界中。ES 6 的发展越来越快，V8 也对 ES 6 中的部分特性实现了支持，如 Generator 等。

（4）稳步发展时期。随着 ES 2015 的发展和最终定稿，一大批利用 ES 2015 特性开发的新模块出现，如原 Express 核心团队所开发的 Koa。Node.js 之父 Ryan Dahl 退出 Node.js 的核心开发，转而做其他的研究项目。Ryan Dahl 的接任者 Isaac Schlueter（也是 Node.js 的核心构建者）将 Node.js 一直开发下去并不断完善。

1.2.2　Node.js 未来版本规划

Node.js 的核心团队已经为 Node.js 的长远发展做好了详细计划。图 1.1 所示是 Node.js 到 2019 年 10 月为止的所有版本计划及发布时间表。

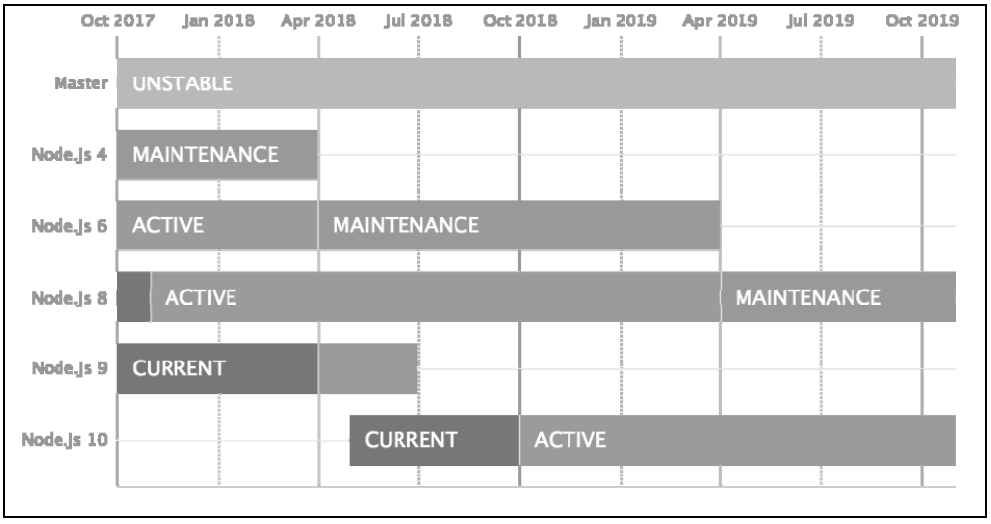

图 1.1　计划及发布时间表

1.2.3　Node.js 的结构

前面介绍了 Node.js 是一个完整的 JavaScript 开发环境，并且是基于 Google 的 Chrome V8 引擎进行代码解释的。它在设计之初就已经定位用来解决传统 Web 开发语言所遇到的诸多问题，所以 Node.js 有很多其他开发语言所不具备的优点，包括事件驱动、异步编程等。下面我们先介绍一下 Node.js 的结构，然后详细分析 Node.js 的一些主要特点。

图 1.2 中，浅色部分是由 JavaScript 编写的，深色部分是由 C/C++完成的。从图 1.2 中可以看出，Node.js 的结构大致分为以下 3 个层次。

- Node.js 标准库。这部分是由 JavaScript 编写的，即我们使用过程中能直接调用的 API。在源码中的 lib 目录下可以看到。
- Node bindings。这一层是 JavaScript 与底层 C/C++能够沟通的关键，前者通过 bindings 调用后者，相互交换数据。
- 支撑 Node.js 运行的基础构件。这层内容比较多，下面详细介绍。

支撑 Node.js 运行的基础构件是由 C/C++实现的，其中包括四大部分。

- V8: Google 推出的 JavaScript VM，也是 Node.js 使用 JavaScript 的关键，它为 JavaScript 提供了在非浏览器端运行的环境，它的高效是 Node.js 之所以高效的原因之一。
- libuv：为 Node.js 提供了跨平台、线程池、事件池、异步 I/O 等能力，是 Node.js 如此强大的关键。
- C-ares：提供了异步处理 DNS 相关的能力。
- http_parser、OpenSSL、zlib 等：提供包括 HTTP 解析、SSL、数据压缩等能力。

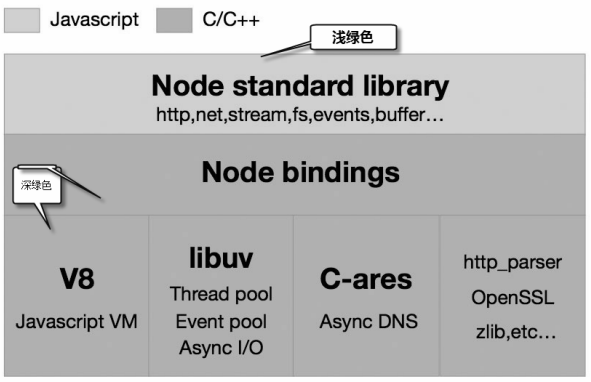

图 1.2　Node.js 的结构

1.2.4　Node.js v10 的特点及新变化

都说 Node.js 强大，这种强大体现在很多方面，如事件驱动、异步处理、非阻塞 I/O 等。这里将介绍 Node.js 具备的不同于其他框架的特点。

1. 事件驱动

在某些传统语言的网络编程中，我们会用到回调函数，比如当 Socket 资源达到某种状态时，注册的回调函数就会执行。Node.js 的设计思想以事件驱动为核心，它提供的绝大多数 API 都是基于事件的、异步的风格。以 Net 模块为例，其中的 net.Socket 对象的事件有 connect、data、end、timeout、drain、error、close 等。使用 Node.js 的开发人员需要根据自己的业务逻辑注册相应的回调函数。这些回调函数都是异步执行的。这意味着虽然在代码结构中这些函数看似是依次注册的，但是它们并不依赖于自身出现的顺序，而是等待相应的事件触发。

事件驱动的优势在于充分利用了系统资源，执行代码无须阻塞等待某种操作完成，有限的资源可以用于其他的任务。此类设计非常适合后端的网络服务编程，Node.js 的目标也在于此。在服务器开发中，并发的请求处理是一个大问题，阻塞式的函数会导致资源浪费和时间延迟。通过事件注册、异步函数，开发人员可以提高资源的利用率，性能也会改善。

2. 异步、非阻塞 I/O

从 Node.js 提供的支持模块中，我们可以看到包括文件操作在内的许多函数都是异步执行的。这和传统语言存在区别。为了方便服务器开发，Node.js 的网络模块特别多，包括 HTTP、DNS、NET、UDP、HTTPS、TLS 等。开发人员可以在此基础上快速构建 Web 服务器。

一个异步 I/O 的大致流程如图 1.3 所示，讲解如下。

（1）发起 I/O 调用

① 用户通过 JavaScript 代码调用 Node 核心模块，将参数和回调函数传入核心模块。

② Node 核心模块会将传入的参数和回调函数封装成一个请求对象。

③ 将这个请求对象推入 I/O 线程池等待执行。

④ JavaScript 发起的异步调用结束，JavaScript 线程继续执行后续操作。

（2）执行回调

① I/O 操作完成后会将结果存储到请求对象的 result 属性上，并发出操作完成的通知。

② 每次事件循环时会检查是否有完成的 I/O 操作，如果有就将请求对象加入 I/O 观察者队列中，之后当作事件处理。

③ 处理 I/O 观察者事件时会取出之前封装在请求对象中的回调函数，执行这个回调函数，并将 result 当作参数，以完成 JavaScript 回调的目的。

Node.js 的网络编程非常方便，提供的模块（在这里是 HTTP）开放了容易上手的 API 接口，短短几行代码就可以构建服务器。

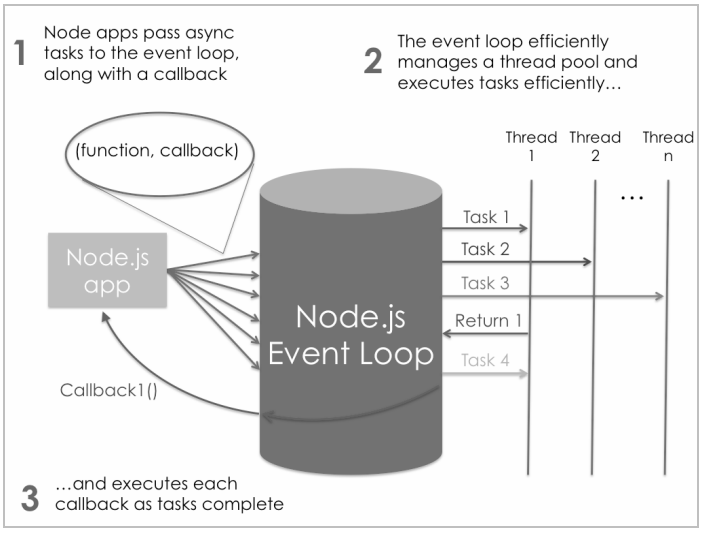

图 1.3　异步 I/O 的流程

3. 性能出众

创始人 Ryan Dahl 在设计的时候就考虑了性能方面的问题，选择 C++和 V8，而不是 Ruby 或者其他的虚拟机。Node.js 在设计上以单进程、单线程模式运行。事件驱动机制是 Node.js 通过内部单线程高效率地维护事件循环队列来实现的，没有多线程的资源占用和上下文切换。这意味着面对大规模的 HTTP 请求，Node.js 是凭借事件驱动来完成的。从大量的测试结果分析来看，Node.js 的处理性能是非常出色的，在 QPS 达到 16700 次时，内存仅占用 30MB（测试环境：RHEL 5.2、CPU 2.2GHz、内存 4GB）。

4. 单线程

Node.js 和大名鼎鼎的 Nginx 一样，都是以单线程为基础的。这正是 Node.js 保持轻量级和高性能的关键，也是 Ryan Dahl 设计 Node.js 的初衷。这里的单线程是指主线程为“单线程”，所有阻塞的部分交给一个线程池处理，然后这个主线程通过一个队列跟线程池协作。我们写的

JS 代码部分不用再关心线程问题，代码也主要由一堆 callback 回调构成，然后主线程在循环过程中适时调用这些代码。

单线程除了保证 Node.js 的高性能之外，还保证了绝对的线程安全，使开发者不用担心同一变量同时被多个线程读写而造成的程序崩溃。

5. 新变化

Node.js V10 版本的正式发布标志着 Node.js Foundation 自诞生以来走到了第 7 个主要版本，并且该版本在 2018 年 10 月成为下一个 LTS 分支。关于 Node.js V10 版本的新功能和新变化基本概括如下。

（1）最新的 Node.js V10 版本自带了定制化的 Node-ChakraCore 引擎，其新增的功能具体包括：

① 全面支持 N-API。
② 可轻松通过新的 Visual Studio Code Extension 进行 Time-Travel 调试。
③ 支持 TTD 的生成器和异步函数。
④ 支持 Inspector 协议。
⑤ 增强稳定性和其他各种改进。

（2）最新的 Node.js V10 版本做了如下重要的更新：

① N-API native addons API 已结束实验状态。
② Async_hooks 过时的实验性 async_hooks API 已被删除。
③ Child Process 忽略了未定义的 env 属性。
④ Console 新增了 console.table()方法。
⑤ 关于 Crypto 功能：

- crypto.createCipher()方法和 crypto.createDecipher()方法已经被弃用，同时被 crypto.createCipheriv()方法和 crypto.createDecipheriv()方法所代替。
- decipher.finaltol()方法已被弃用。
- crypto.DEFAULT_ENCODING 属性已被弃用。
- 新增 ECDH.convertKey()方法。
- crypto.fips 属性已经被弃用。

⑥ 依赖更新：

- Google V8 已升级至 6.6 版本。
- OpenSSL 已升级至 1.1.0h 版本。

1.2.5　Node.js 的应用场景

Node.js 可以应用到很多方面，可以说从 Node.js 开始，程序员可以使用 JavaScript 来开发

服务器端的程序了。Node.js 为前端开发程序员提供了便利，并在各大网站中承担重要角色，成为开发高并发大型网络应用的关键技术。Web 站点早已不局限于内容的呈现，很多交互型和协作型环境也逐渐被搬到了网站上，而且这种需求还在不断地增长。这就是所谓的数据密集型实时（data-intensive real-time）应用程序，比如在线协作的白板、多人在线游戏等，这种 Web 应用程序需要一个能够实时响应大量并发用户请求的平台支撑它们，这正是 Node.js 擅长的领域。此外，Node.js 的跨平台特性是使用 Node.js 语言开发流行的另一大原因。

Node.js 的主要应用场景如下：

- JSON APIs——构建一个 Rest/JSON API 服务，Node.js 可以充分发挥其非阻塞 IO 模型以及 JavaScript 对 JSON 的功能支持（如 JSON.stringfy 函数）。
- 单页面、多 Ajax 请求应用——如 Gmail，前端有大量的异步请求，需要服务后端有极高的响应速度。
- 基于 Node.js 开发 UNIX 命令行工具——Node.js 可以大量生产子进程，并以流的方式输出，这使得它非常适合做 UNIX 命令行工具。
- 流式数据——传统的 Web 应用通常会将 HTTP 请求和响应看成原子事件，而 Node.js 会充分利用流式数据这个特点构建非常酷的应用，如实时文件上传系统 transloadit。
- 准实时应用系统——如聊天系统、微博系统，但 JavaScript 是有垃圾回收机制的，这就意味着系统的响应时间是不平滑的（GC 垃圾回收会导致系统这一时刻停止工作）。如果想要构建硬实时应用系统，Erlang 是一个不错的选择。

例如，实时互动交互比较多的社交网站（像 Twitter 这样的公司）必须接收 tweets 并将其写入数据库。实际上，每秒几乎有数千条 tweet 到达，数据库不可能及时处理高峰时段所需的写入数量。Node 成为这个问题解决方案的重要一环。Node 能处理数万条入站 tweet。它能快速而又轻松地将 tweets 写入一个内存排队机制（例如 memcached），另一个单独进程可以从那里将 tweets 写入数据库。Node 能处理每个连接而不会阻塞通道，从而能够捕获尽可能多的 tweets。

虽然看起来 Node.js 可以做很多事情，并且拥有很高的性能，但是 Node.js 并不是万能的，有一些类型的应用，Node.js 可能处理起来会比较吃力。例如，CPU 密集型的应用、模板渲染、压缩/解压缩、加/解密等操作都是 Node.js 的软肋。

1.3 Node.js 在中国的发展

Node.js 在发展初期，国内就有大量的开发者开始持续关注了。随着 Node.js 的不断成熟，很多国内的公司都开始采用这一新技术。2013 年、2014 年、2015 年的 JS 中国开发者大会都将 Node.js 作为主要的宣讲内容，Node.js 也受国内的开发爱好者追捧。Node.js 开发者在国内的数量不断增加，并涌现出很多组织和机构来自发地进行推广和技术分享。

国内的各大视频培训网站上都有 Node.js 开发的培训教程，各大门户网站也都或多或少地采用了 Node.js 的开发技术，淘宝、网易、百度等有很多项目都运行在 Node.js 之上。阿里云是在这方面比较靠前的公司，其云平台率先支持 Node.js 的开发。淘宝也为 Node.js 搭建了国内的 NPM 镜像网站，方便国内的开发者下载各种开发包。

1.3.1　Node.js 中文资源汇总

（1）有关于 Node.js 最新版本的下载和新闻、丰富的文档资料，非常值得认真学习，是 Node.js 开发爱好者不容错过的网站，网址为 http:// http://nodejs.cn/。

（2）CNode 社区，由一批热爱 Node.js 技术的工程师发起，已经吸引了各个互联网公司的专业技术人员加入，是目前国内非常具有影响力的 Node.js 开源技术社区，致力于 Node.js 的技术研究，并有论坛会定期组织一些技术交流活动，网址为 https://cnodejs.org/。

（3）全栈技术社区，是一个专业的 Node.js 中文知识分享社区，致力于普及 Node.js 知识，分享 Node.js 研究成果，努力推进 Node.js 在中国的应用和发展，网站有大量的技术博客和文章，各个级别的开发者都能找到适合自己学习的资料，网址为 https://www.nodejsnet.com/。

（4）淘宝 NPM 镜像，是一个完整 npmjs.org 镜像，可以用此代替官方版本，同步频率为每 10 分钟一次，以保证尽量与官方服务同步，网址为 https://npm.taobao.org/。

每年的 JS 中国开发者大会和各种 Node.js 分享沙龙都是很好的学习 Node.js 开发技术和交流的机会。一个开发者要时刻保持谦虚的心态，并不断学习最新的技术。这对开发者来说是一种基本能力和素养。

1.3.2　Node.js 的发展和未来

Node.js 在创建之初是为了开发即时通信的 Web 应用，当然现在的 Node.js 已绝不只是一套简单的 Web 堆栈——作为一项技术，它在多个层面焕发出勃勃生机，价值已经远远超出了常见 Web 服务器的范畴。

例如，一款由 Jacob Groundwater 打造的项目 NodeOS，其创始人希望围绕 Linux 核心建立一套新型环境。其中，Node.js 作为“shell”，而 Node 的 NPM 则被用于系统包管理器。截至目前，NodeOS 的首个版本已经创建完成。Node.js 还被用来作为硬件控制的工具代替 C/C++，Noduino 允许大家经由 WebSocket 或者串连接实现 Arduino 访问。该项目虽然尚处于起步阶段，但是驱动主板上的 LED 模块、捕捉来自 Arduino 的事件（例如按下按钮）等常见功能都已经可以正常支持，以后就可以通过网页直接控制 Arduino 硬件和其他物联网设备了。

2016 年，Node.js 发布了两个重量级的版本：v4.4.0 LTS（长期支持版本）和 v5.9.0 Stable（稳定版本），并且成立了 Node.js 基金会，能够让 Node.js 在未来有更好的开源社区支持。Node.js 基金会的创始成员包括 Joyent、IBM、Paypal、微软、Fidelity 和 Linux 基金会。Node.js 的 core（核心）已经非常稳定并逐步被广大开发者认可，进而进行大规模使用。著名的 Node.js 包管理工具 NPM 在 2014 年成为软件开发世界中包管理工具的龙头老大，现在 NPM 包含的模块数是 Java 以及 Ruby 的包管理工具模块数的两倍。图 1.4 反映了 NPM 包管理工具的增长情况。

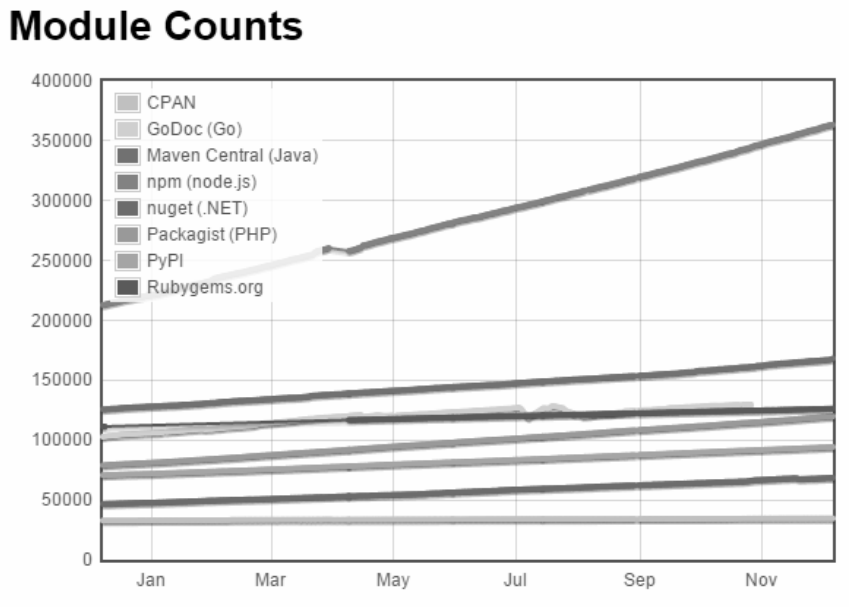

图 1.4　NPM 包管理工具的增长情况

数据来源于 http://www.modulecounts.com/。

2018 年上半年，Node.js 和 NPM 的普及率进一步提升。大公司对 Node.js 的应用持续提升，并推出更多对企业级友好功能的长期计划，可能预示着 Node.js 在企业中会持续增长，替换一部分像 Java 和.NET 这样的典型解决方案。Node.js 诞生于 2009 年，其年龄远不如 Python、Ruby、PHP 等老牌开发语言，但是它成为有史以来发展最快的开发工具。可以预见，在未来的几年 Node.js 技术会不断发展，成为 Web 开发的核心技术，并从现有的 Java、PHP 等语言中争夺到更多的份额。

1.4 温故知新

学完本章内容，读者需要回答：

1. Node.js 有哪几大主要特性？
2. Node.js V10 有哪些新变化？
3. 在选择 Node.js 开发的时候有哪些注意事项？

第 2 章

◂ 部署Node.js开发环境 ▸

部署 Node.js 开发环境是正式接触 Node.js 的第一步。Node.js 可以在多个不同的平台稳定运行，并且具有良好的兼容性。但是 Node.js 部署在不同操作系统下的方法并不完全一致，本章主要介绍如何在各个操作系统平台下进行 Node.js 的部署。

通过本章的学习，可以掌握以下内容：

- Windows 部署：学会如何在 Windows 操作系统上安装 Node.js。
- Linux 部署：学会在各种 Linux 发行版本上部署 Node.js 开发环境。
- Mac OS X 部署：学会在 Mac 的 OS X 系统上部署 Node.js 开发环境。
- 树莓派 3 部署：学习如何使用 nvm 在树莓派 3 上部署 Node.js 开发环境。
- 开发工具介绍：介绍 Sublime Text 3 的使用方法和 Node.js 插件的安装。

2.1 在 Windows 10 下部署 Node.js 开发环境

Node.js 可以在 Windows 系统下稳定运行，本节主要介绍 Windows 10 下的 Node.js 环境部署。在 Windows 中进行 Node.js 环境部署是相对比较简单的，首先从 Node.js 的官方网站（https://nodejs.org/en/download/）上下载最新的 Windows 安装包，国内用户可以通过 Node.js 官方中文站点（http://nodejs.cn/download/）进行下载。中文站点的下载页面和英文站点的布局略有不同，中文站点只提供最新发布版本的下载链接，而英文站点同时提供最新稳定版本和最新版本两个版本的下载链接。

Node.js 的其他发布版本可以从 https://nodejs.org/dist/找到，本书以 v10.9.0 在 Windows 10 进行安装为例进行介绍。

Node.js 的安装包在 Windows 平台分为 installer 和 Binary 两个版本。installer 是通常的安装包发布版本（.msi）。Binary 为二进制版本，可以下载后直接运行（.exe）。这里建议使用后缀为.msi 的安装版本。此外，Node.js 的安装包分为 32 位和 64 位，在下载的时候请查看系统的具体信息，并选择正确的安装包进行下载和安装。

打开 Node.js 官方网站下载页面，如图 2.1 所示。读者可根据自己的系统选择，比如笔者的电脑是 64 位的 Windows 系统，下载的是 node-v10.9.0-x64.msi。

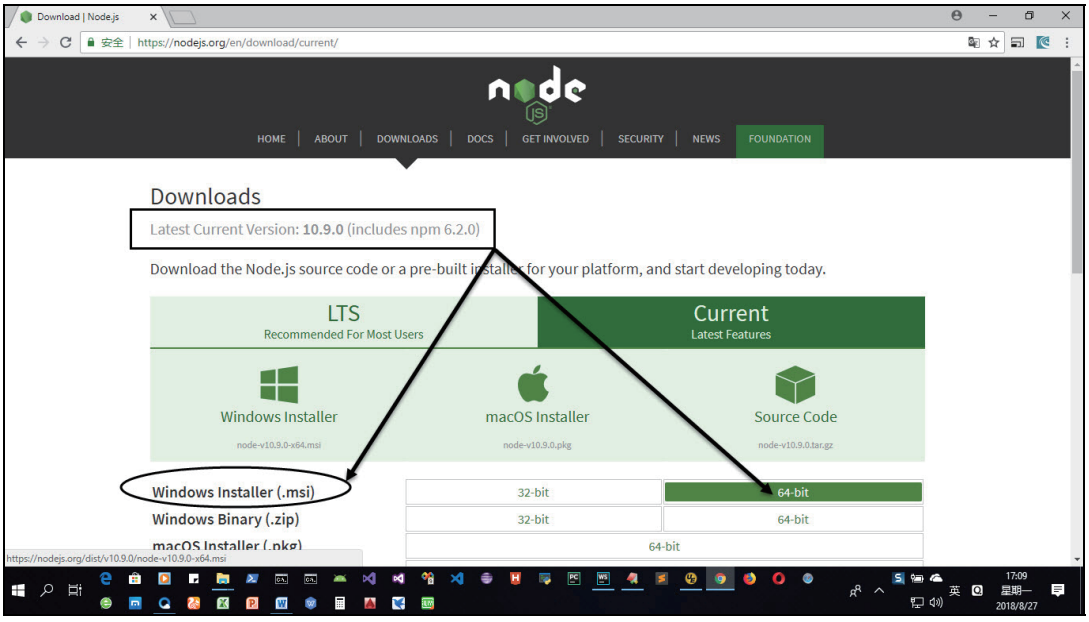

图 2.1　Node.js 官方网站下载页面

如果选择 Node.js 之前的版本，可以在官方网站中找到 Previous Releases 链接，然后查找需要的版本号。

2.1.1　使用安装包安装 Node.js

（1）安装包下载完之后是一个 16MB 大小的后缀名为 msi 的安装文件，双击下载后的安装包文件，首先弹出安装向导的欢迎界面，如图 2.2 所示。

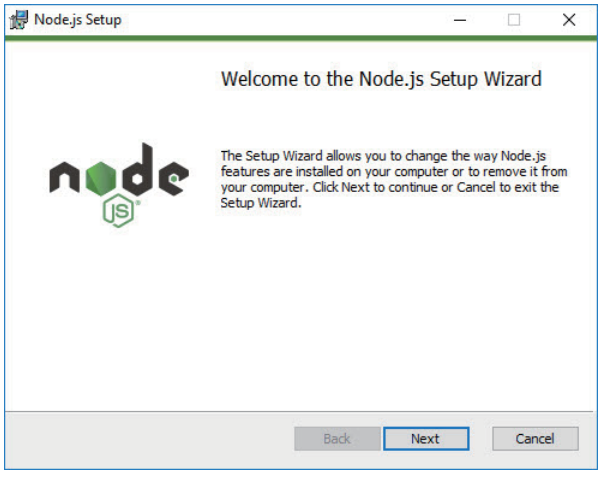

图 2.2　欢迎界面

（2）单击图 2.2 中的 Next（下一步）按钮会出现最终用户授权协议界面，如图 2.3 所示。

勾选接受协议选项后 Next 按钮会变为可用状态，单击 Next 按钮进入下一步。

图 2.3　最终用户授权协议界面

（3）此时打开 Node.js，默认安装目录为 C:\Program Files\nodejs\，如图 2.4 所示。我们既可以通过 Change 按钮修改目录，又可以直接单击 Next 按钮。

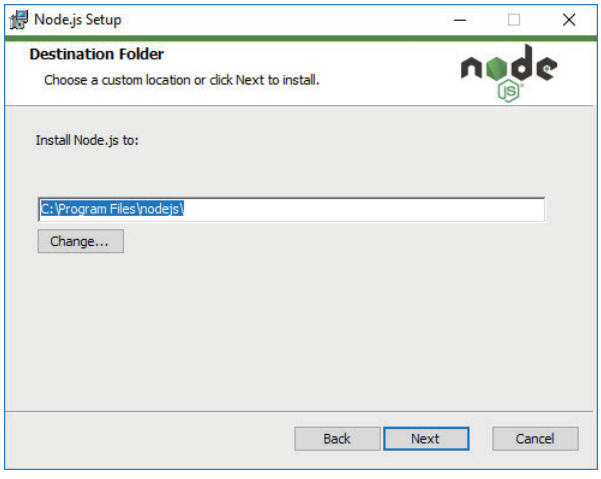

图 2.4　安装目录

（4）单击 Next 按钮后出现如图 2.5 所示的自定义安装界面，我们选择默认设置即可。单击 Next 按钮后会出现一个准备安装界面，单击界面中的 Install 按钮。

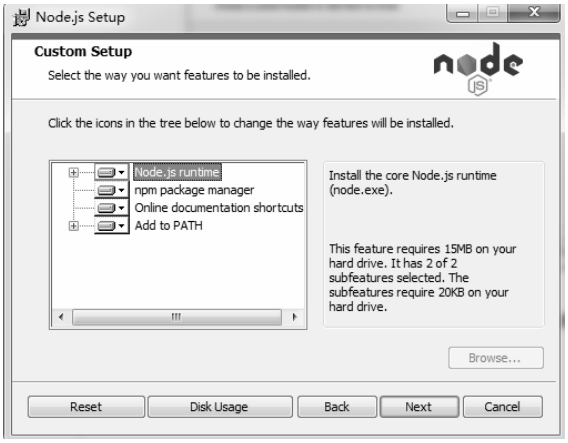

图 2.5　自定义安装界面

（5）开始安装的界面如图 2.6 所示。安装需要等待 1 分钟左右，安装完成后出现完成界面，单击 Finish 按钮就完成安装了。

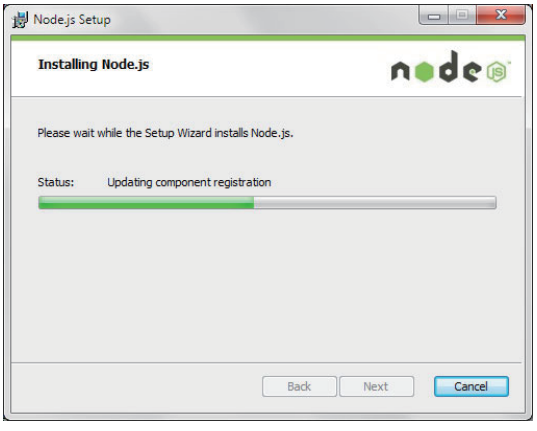

图 2.6　开始安装

（6）安装后，系统默认的环境变量 PATH 是 C:\Documents and Settings\Administrator\Application Data\npm，也可以根据需要手动修改本地的安装目录，并将全局目录设置为与本地初始默认安装目录一致。在完成安装 Node.js 的时候，默认也安装了 NPM。NPM 是 Node.js 的包管理工具，在后面的章节进行介绍。

要查看 PATH 变量，需要右击计算机，选择“属性”|“高级系统属性”选项，打开“系统属性”对话框。然后单击“高级”|“环境变量” 选项，在“用户变量”列表项中找到 PATH 变量，单击下方的“编辑”按钮就可以看到所有变量在这里的设置，主要是设置路径。

2.1.2　测试 Node.js 开发环境

安装 Node.js 开发环境成功之后，创建一个简单的 App 来测试 Node.js 是否能够正常运行。

首先，为 App 创建一个目录，目录里面创建一个名为 hello_world.js 的 JS 文件，然后在 hello_world.js 中写入如下代码。

【示例 2-1】

```
var http = require('http');
http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(3000);
console.log('Server running at http://localhost:3000/');
```

【代码说明】

上面的代码是一个简单的 Node.js Web 服务举例，会在电脑上创建一个 HTTP Web 服务，并在网页上打印出 Hello World 字符串。通过 Windows 开始菜单丨Node.js丨Node.js command prompt 来运行 Node.js。Node.js command prompt 是一个命令行界面，用来启动 Node.js 编译环境。如果在开始菜单里找不到 Node.js command prompt，可以在搜索框中输入 node，然后找到它并运行。

因为默认打开的路径不是我们创建项目的路径，所以这里可以通过 cd 命令来切换路径：

```
E:\                                  #切换到 E 盘
cd E:\WebstormProjects\NodejsDev\chpater02      #切换到项目路径
```

接下来运行 hello_world.js，简单地输入如下 node 命令：

```
node hello_world.js
```

如果一切都顺利，那么我们将会在 command prompt 中看到：

```
Server running at http://localhost:3000/
```

界面内容如图 2.7 所示。

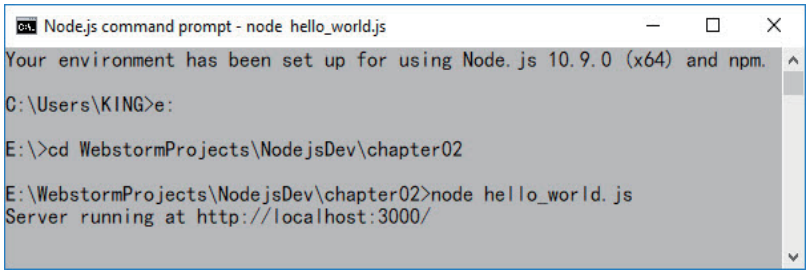

图 2.7　Node.js 命令行运行界面

然后打开浏览器输入如下 URL：

```
http://localhost:3000/
```

接着会在浏览器中看到“Hello World”，如图 2.8 所示。这说明 Node 平台安装成功，并且能成功运行 Node.js 程序。

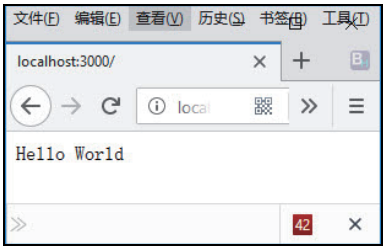

图 2.8　Hello World 程序运行结果

2.2 在 Linux 下部署 Node.js 开发环境

在 Linux 下安装 Node.js 有很多种方法，常见的方法有通过包管理器安装和源码安装。下面针对各种版本的 Linux 和安装方法进行介绍。

2.2.1 通过源码安装 Node.js

下面以 Ubuntu 16.04 为例说明如何使用源码安装 Node.js。

（1）通过下面的命令安装版本工具。

```
apt-get install make g++ libssl-dev git
```

（2）新建一个目录，并使用 wget 命令下载 Node.js 源码。

```
cd /tmp
wget http://nodejs.org/dist/v0.10.32/node-v10.9.0-linux-x64.tar.gz
tar -xvf node-v10.9.0-linux-x64.tar.gz
cd node-v10.9.0
```

或者通过 git 命令直接从 GitHub 上复制。

```
git clone https://github.com/joyent/node
cd node
```

（3）配置安装选项并进行编译安装，其中 X 代表服务器的 CPU 数量。

```
./configure
make -jX
make install
```

安装成功后，可以使用 node -v 命令来检查 Node.js 的版本以及是否安装成功。

Node.js 选择下载源码进行编译安装之前，要确保系统安装了 Python 2.6 或 3.5（或更高的版本）。

2.2.2　通过包管理器安装 Node.js

在 Linux 的不同版本下可以使用 NPM 安装 Node.js。下面仅列举几种常见的安装 Linux 发布版的方法。

1. Arch Linux

在 Arch Linux 中，Node.js 和 NPM 包是全面支持的，可以通过一条指令进行安装：

```
pacman -S nodejs npm
```

2.基于 Debian 和 Ubuntu 的 Linux 发布版

基于 Debian 和 Ubuntu 的 Linux 发布版主要包括 Linux Mint、Linux Mint Debian Edition（LMDE）和 elementaryOS，可以通过 Debian 和 Ubuntu 的社区 NodeSource 来下载和安装，在 GitHub 上的链接地址是 https://github.com/nodesource/distributions。需要注意的是，通过 nodesource 安装的版本可能并不是最新的。

- 安装 Node.js 10.x 版本的方法如下:

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
sudo apt-get install -y nodejs
```

3.Red Hat Enterprise Linux/RHEL、CentOS 和 Fedora

在 Red Hat、CentOS 和 Fedora 上使用 NPM 安装 Node.js 的时候需要修改对应的地址，具体版本的链接可能略有不同。要使用 root 用户登录，并执行下面的命令：

- 4.x 版本

```
curl --silent --location https://rpm.nodesource.com/setup_4.x | bash -
```

- 6.x 版本

```
curl --silent --location https://rpm.nodesource.com/setup_6.x | bash -
```

- 10.x 版本

```
curl --silent --location https://rpm.nodesource.com/setup_10.x | bash -
```

然后使用 yum 命令来安装 Node.js：

```
yum -y install nodejs
```

在 Fedora 18 之后的版本，Node.js 和 NPM 是默认支持的，只需要通过下面的一条命令就可以安装：

```
sudo yum install nodejs npm
```

2.3 在 Mac OS X 下部署 Node.js 开发环境

在 Mac OS X 上安装有 3 种方式，即使用源码安装、使用 NPM 包管理器安装和使用安装包安装。本节主要介绍如何使用.dmg 安装包和 NPM 包管理器进行安装。

2.3.1 使用.dmg 安装包进行安装

首先从 Node.js 官方网站上下载最新的 OS X 安装包，并按照安装向导进行安装，界面如图 2.9 所示。

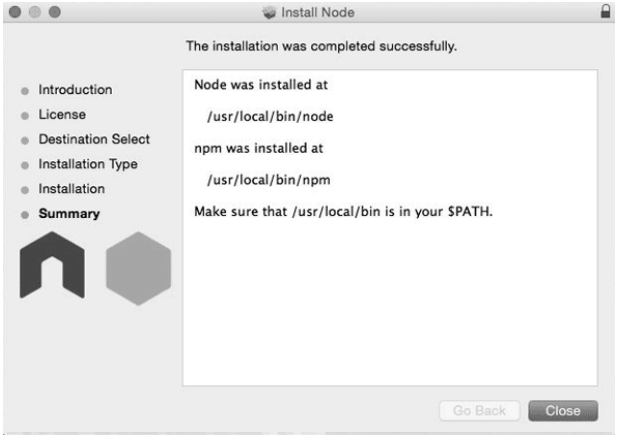

图 2.9 在 OS X 下安装 Node.js

安装之后，/usr/local/bin 在环境变量中已经定义，如果没有，就在当前用户 home 目录下的.bash_profile 或者.bashrc 中添加。

2.3.2 使用 NPM 包管理器安装

使用 NMP 包管理器安装 Node.js 时要从 Node.js 的官方网站上下载最新的 Macintosh Installer。可以通过 curl 执行下面的命令来安装：

```
curl "https://nodejs.org/dist/latest/node-${VERSION:-$(wget -qO-
https://nodejs.org/dist/latest/ | sed -nE 's|.*>node-(.*)\.pkg</a>.*|\1|p')}.pkg"
> "$HOME/Downloads/node-latest.pkg" && sudo installer -store -pkg
"$HOME/Downloads/node-latest.pkg" -target "/"
```

或者使用 MacPorts 执行下面的代码：

```
port install nodejs
```

2.4 在树莓派 3 下使用 NVM 安装 Node.js

树莓派是一款卡片式的学习电脑，由英国剑桥大学在 2012 年 3 月首发。本节主要介绍如何在树莓派 3 的官方操作系统 Raspbian 上使用 NVM 安装 Node.js。

（1）下载和安装 NVM（https://github.com/creationix/nvm）。NVM 的全称是 Node Version Manager（Node.js 版本管理器），它可以让我们在各个 Node.js 版本之间进行灵活的切换。

```
git clone https://github.com/creationix/nvm.git ~/.nvm && cd ~/.nvm && git checkout v0.25.4
```

（2）使用 nano 命令编辑.bashrc 和.profile，在文件末尾增加 source ~/.nvm/nvm.sh，并重新启动树莓派，具体命令如下：

```
sudo nano ~/.bashrc
sudo nano ~/.profile
sudo reboot
```

（3）启动完成后，在 shell 里面执行 nvm 命令。当看到输出时，表示 NVM 安装成功。

```
nvm --version
```

（4）开始安装 Node.js。使用 nvm 命令安装 Node.js 稳定版，如 v8.11.4。

```
$ nvm install 8.11.4
```

（5）安装完成后，可以使用下面的代码进行查看。

```
$ nvm ls
```

这时可以看到自己安装的所有 Node.js 版本。

2.5 使用 NPM 进行 Node 包的安装

前面我们已经使用很多次 NPM 命令了，这其实使用的是 Node.js 默认的包管理器 NPM。当 Node.js 安装完成后，NPM 也默认安装完成。安装包模块使得 Node.js 变成了一个更加强大的 Web App 开发平台。它能预先为 Node.js App 提供所需要的功能。NPM 官方网站号称有 250 000 个不同的包可供开发者下载使用，网址为 https://www.npmjs.com/。

例如，我们希望在 Node.js 上扩展一个 MySQL 接口，可以让我们在 App 中使用 MySQL 数据库，只需要在 Node.js command prompt 中输入如下命令即可：

```
npm  install  mysql
```

上面的命令会通过 NPM 下载和安装 MySQL Node 包。当选择的包安装完成时会看到如图

2.10 所示的信息。

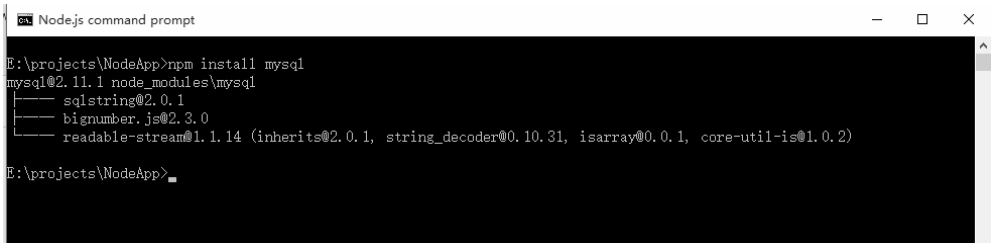

图 2.10　使用 NPM 在 Windows 下安装 MySQL 包

NPM 的常用命令介绍如下。

（1）查看帮助

```
npm help 或 npm h
```

（2）安装模块

```
npm intstall <Module Name>
```

（3）在全局环境中安装模块（-g：启用 global 模式）

```
npm install -g <Module Name>
```

更多的内容可参考 https://npmjs.org/doc/install.html。

（4）卸载模块

```
npm uninstall  <Moudle Name>
```

（5）显示当前目录下安装的模块

```
npm list
```

Node.js 安装成功后，系统会自动在 PATH 用户环境变量和系统环境中分别添加 NPM 和 Node.js 路径。

2.6 开发工具介绍

为了更高效地编写 Node.js 代码，需要使用一个好的编辑器，这里推荐大家使用 Sublime Text。Sublime Text 是一款具有代码高亮、语法提示、自动完成且反应快速的编辑器软件。它主要有以下两大优点。

- 跨平台：Sublime Text 3 为跨平台编辑器，可以在 Windows、Mac OS X 和 Linux 下安装。开发人员在开发和测试的时候切换系统是常有的事情，为了减少重复学习，使用

一个跨平台的编辑器是很有必要的。Sublime Text 3 可以使你的开发工作在各个系统之间进行无缝切换。

- 可扩展：Sublime Text 3 包含大量实用插件，可以通过安装自己领域的插件来成倍提高工作效率。

Sublime Text 有 Sublime Text 2 和 Sublime Text 3 两个版本。它们的界面大致相同，但是 Sublime Text 3 的启动速度更快，而且支持更多的功能，所以本书以 Sublime Text 3 为例进行介绍。

2.6.1　下载安装 Sublime Text 3

Sublime Text 3 beta 版本已经非常稳定了，官方下载网址为 http://www.sublimetext.com/3。需要注意的是 Sublime Text 3 是付费软件，虽然可以无限期地进行试用，但是如果是长期使用，建议购买正版的序列号激活。

（1）打开下载页面，如图 2.11 所示。64 位的 Windows 系统请选择“Windows 64 bit”安装包，即下载文件为“Sublime Text Build 3176x64 Setup.exe”的安装程序。“portable version”下载下来为“Sublime Text Build 3176 x64.zip”编辑器的包，解压后无须安装就能运行。

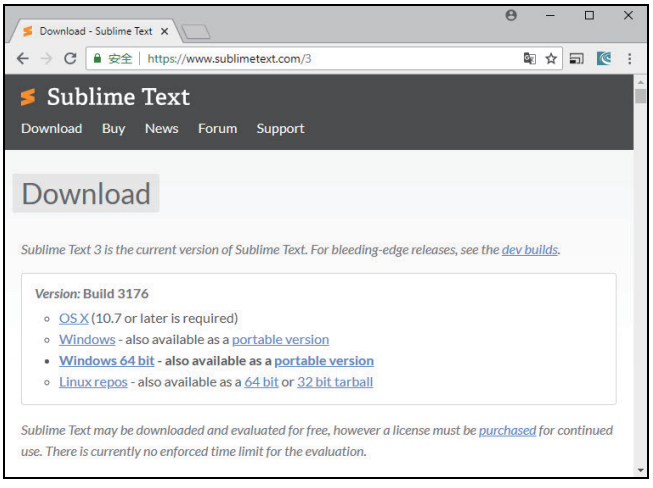

图 2.11　Sublime Text 3 官方下载页面

（2）双击下载下来的安装包，并按照提示进行安装，如图 2.12 所示。

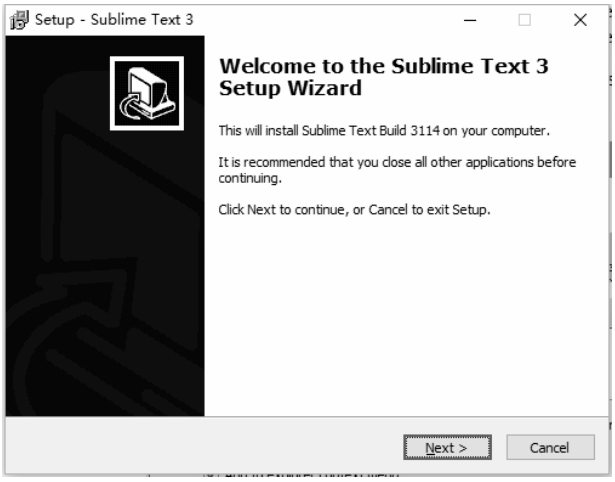

图 2.12　Sublime Text 3 的安装过程

（3）安装成功后，双击桌面上的“Sublime Text 3”快捷图标，就可以打开 Sublime Text 3 程序了。

如果是 Mac OS X 或者 Ubuntu 就下载相应的安装包，并参照安装说明进行操作。如果是在 Linux 下安装，就先使用 uname －m 命令查看操作系统的类型，再选择合适的安装包。

2.6.2　Sublime Text 操作界面

Sublime Text 3 的操作界面如图 2.13 所示。

图 2.13　Sublime Text 3 的操作界面

界面中的各种操作选项说明如下。

- 标签（Tab）：分别显示每个打开的文件。
- 编辑区（Editing Area）：编辑文本内容的区域，位于界面的中心位置。
- 侧栏（Side Bar）：包含当前打开的文件以及文件夹视图。
- 缩略图（Minimap）：当前打开文件的缩略图。
- 命令板（Command Palette）：Sublime Text 的操作中心，使我们基本可以脱离鼠标和菜单栏进行操作。
- 控制台（Console）：使用 Ctrl + `快捷键可以调出该窗口。它既是一个标准的 Python REPL，又可以直接对 Sublime Text 进行配置。
- 状态栏（Status Bar）：显示当前行号、当前语言和 Tab 格式等信息。

Sublime Text 3 的相关操作文档和使用说明在 https://www.sublimetext.com/docs/3/上。

2.6.3　安装 Sublime Text 3 插件

Sublime Text 3 最强大的功能是针对各种开发语言的编辑插件。为了安装和管理这些插件，我们首先需要安装包管理器（Package Control），官方首页链接为 https://packagecontrol.io。通过 Ctrl+`快捷键或者在菜单中选择 View | Show Console 来打开控制台，然后将下面的代码粘贴到控制台中运行。

```
import urllib.request,os,hashlib; h = '2915d1851351e5ee549c20394736b442' +
'8bc59f460fa1548d1514676163dafc88'; pf = 'Package Control.sublime-package'; ipp
= sublime.installed_packages_path();
urllib.request.install_opener( urllib.request.build_opener( urllib.request.Pro
xyHandler()) ); by = urllib.request.urlopen( 'http://packagecontrol.io/' +
pf.replace(' ', '%20')).read(); dh = hashlib.sha256(by).hexdigest(); print('Error
validating download (got %s instead of %s), please try manual install' % (dh, h))
if dh != h else open(os.path.join( ipp, pf), 'wb' ).write(by)
```

这段代码将创建一个安装包的目录，并将包控制器 Package Control.sublime-package 下载到这个目录中。安装完毕后，需要重新启动 Sublime Text 3。

2.6.4　安装 Node.js 插件

在 Package Control 首页的搜索框中输入 NODE，就可以查找到所有和 Node.js 相关的包，如图 2.14 所示。可以看到由 tanepiper 创建的 Node.js 包是当下最热门的 Node.js 插件，下载量为 156 000 次。

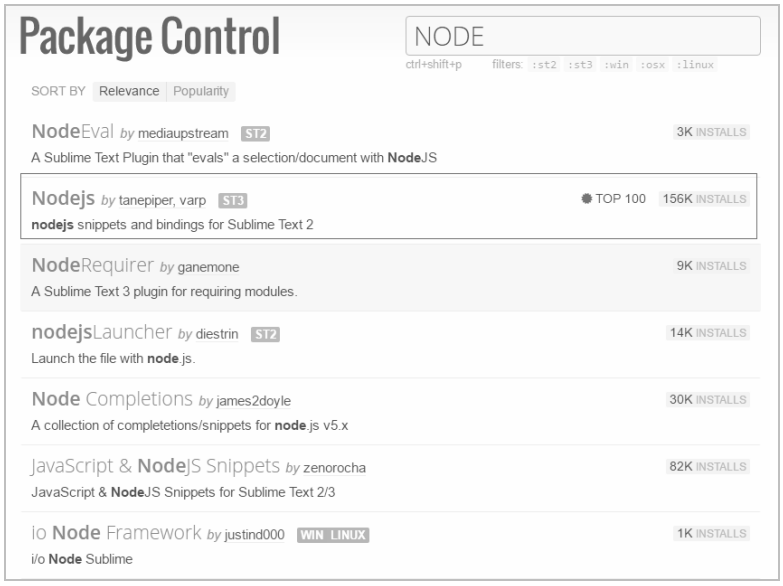

图 2.14　Package Control 包搜索和下载页面

打开 Node.js 包的链接，我们可以看到这个包的详细介绍和使用方法，并且可以查看到这个包每月的下载量。

- 在 MaC OS X 下的安装命令:

```
`git clone https://github.com/tanepiper/SublimeText-Node.js.git
~/Library/Application\ Support/Sublime\ Text\ 3/Packages/Node.js`
```

- 在 Windows 下的安装命令:

```
`git clone https://github.com/tanepiper/SublimeText-Node.js "%APPDATA%\Sublime
Text 3\Packages\Node.js"`
```

- 在 Linux 下的安装命令:

```
`git clone https://github.com/tanepiper/SublimeText-Node.js
$HOME/.config/sublime-text-3/Packages/Node.js`
```

2.6.5　Sublime Text 3 快捷键

按照类型可以把快捷键分为编辑、选择、查找和替换、跳转、窗口、屏幕。这里分别对常用的快捷键做一个简单介绍。

1. 编辑

- Ctrl+Enter: 在当前行下面新增一行，然后跳至该行。
- Ctrl+Shift+Enter: 在当前行上面增加一行并跳至该行。
- Ctrl+←/→: 进行逐词移动。
- Ctrl+Shift+←/→: 进行逐词选择。

- Ctrl+↑/↓：移动当前显示区域。
- Ctrl+Shift+↑/↓：移动当前行。

2. 选择

- Ctrl+D：选择当前光标所在的词并高亮显示该词所有出现的位置，再次按 Ctrl+D 快捷键选择该词出现的下一个位置。在多重选词的过程中，使用 Ctrl+K 快捷键进行跳过，使用 Ctrl+U 快捷键进行回退，使用 Esc 键退出多重编辑。
- Ctrl+Shift+L：将当前选中区域打散。
- Ctrl+J：把当前选中区域合并为一行。
- Ctrl+M：在起始括号和结尾括号间切换。
- Ctrl+Shift+M：快速选择括号间的内容。
- Ctrl+Shift+J：快速选择同缩进的内容。
- Ctrl+Shift+Space：快速选择当前作用域（Scope）的内容。

3. 查找和替换

- F3：跳至当前关键字下一个位置。
- Shift+F3：跳到当前关键字上一个位置。
- Alt+F3：选中当前关键字出现的所有位置。
- Ctrl+F/H：进行标准查找/替换，之后：
  - Alt+C：切换大小写敏感（Case-Sensitive）模式。
  - Alt+W：切换整字匹配（Whole Matching）模式。
  - Alt+R：切换正则匹配（Regex Matching）模式。
- Ctrl+Shift+H：替换当前关键字。
- Ctrl+Alt+Enter：替换所有关键字匹配。
- Ctrl+Shift+F：多文件搜索和替换。

4. 跳转

- Ctrl+P：跳转到指定文件，输入文件名后可以再输入以下内容。
  - @符号：跳转输入，如@symbol 跳转到 symbol 符号所在的位置。
  - #关键字：跳转输入，如#keyword 跳转到 keyword 所在的位置。
  - :行号：跳转输入，如:12 跳转到文件的第 12 行。
- Ctrl+R：跳转到指定符号。
- Ctrl+G：跳转到指定行号。

5. 窗口

- Ctrl+Shift+N：创建一个新窗口。
- Ctrl+N：在当前窗口创建一个新标签。
- Ctrl+W：关闭当前标签，当窗口内没有标签时会关闭该窗口。

- Ctrl+Shift+T：恢复刚刚关闭的标签。

6. 屏幕

- F11：切换至普通全屏。
- Shift+F11：切换至无干扰全屏。
- Alt+Shift+1Single：切换至独屏。
- Alt+Shift+2Columns:2：切换至纵向二栏分屏。
- Alt+Shift+3Columns:3：切换至纵向三栏分屏。
- Alt+Shift+4Columns:4：切换至纵向四栏分屏。
- Alt+Shift+8Rows:2：切换至横向二栏分屏。
- Alt+Shift+9Rows:3：切换至横向三栏分屏。
- Alt+Shift+5：Grid 切换至四格式分屏。

2.7 温故知新

学完本章后，读者需要回答：

1. 在 Windows 10 下安装 Node.js 有哪几种方式？
2. 如何使用 NPM 下载特定的 Node.js 开发包？
3. Node.js 在 Windows 中的默认安装路径是什么？
4. 在 Sublime Text 3 中使用什么快捷键可以新建一个文档？
5. Node.js 支持哪些操作系统？

第二篇

Node.js编程基础

第 3 章

Node.js开发基础

Node.js 是建立在 V8 引擎之上的，也就意味着 Node.js 的语法几乎与 JavaScript 一致。这也是 Node.js 大受前端开发人员欢迎的原因，即通过一门语言便可打通前后端开发。在这一章中将介绍 JavaScript 的基本使用，为之后 Node.js 的学习提供必要的基础。

通过本章的学习可以掌握以下内容：

- JavaScript 的基础语法与使用。
- 了解简单的 JavaScript 编程风格。
- 了解基本的 Node.js 控制台的使用。

3.1 JavaScript 语法

JavaScript 是一门直译式、弱类型的脚本语言，也是 Web 开发最重要的语言之一。JavaScript 由 ECMAScript、DOM（文档对象模型）、BOM（浏览器对象模型）三部分组成。ECMAScript 规定了 JavaScript 的语法核心，这也是本节重点介绍的内容。

3.1.1 变量

1. 交互式运行环境——REPL

Node.js 提供了一个交互式运行环境——REPL。在这个交互式环境中可以运行简单的应用程序。在控制台直接输入 node 命令即可进入这个环境，此时控制台会显示一个“>”符号，表明我们已经进入这个环境，然后就可以直接在命令行输入 node 代码并执行了，如图 3.1 所示。

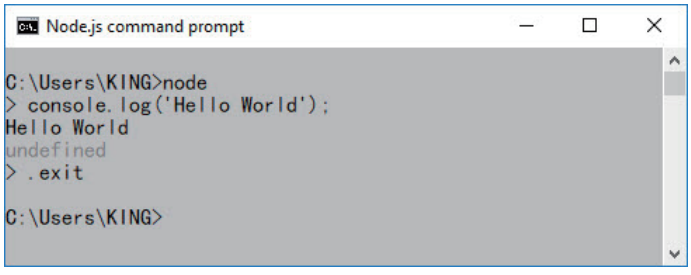

图 3.1　进入 REPL 运行环境

如果要退出该运行环境，连续按两次 Ctrl+C 快捷键，或者输入.exit。Node.js 的命令需要在前面加点。例如，可用.help 查看所有命令。

本节中的所有代码都会在这个环境中使用和运行。

2. 浏览器环境——Chrome & FireFox

当然，读者也可以在浏览器（Chrome & FireFox）的控制台运行。以 Chrome 浏览器为例（FireFox 浏览器与之大同小异），通过使用 F12 键或者 Ctrl + Shift + I 快捷键打开开发者工具，在开发者工具栏中选择 Console 面板，在 Console 中也是显示一个“>”符号，和 REPL 的使用方法一致，如图 3.2 所示。

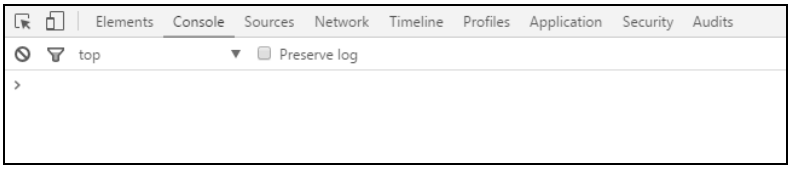

图 3.2　浏览器的 Console 面板

3. 关键字 var

JavaScript 的变量通过关键字 var 来声明。前面说过 JavaScript 是一门弱类型的编程语言，JavaScript 的所有数据类型都可以用 var 关键字来声明，通过“var 变量名=值”的形式就可以对变量同时进行声明和赋值。和许多语言一样，JavaScript 通过分号“;”来分隔不同的语句，以下这段代码就声明了两个变量：

```
var a = "node.js";
var b = 10;
```

4. 变量的命名

JavaScript 规定变量名必须以字母、美元符（$）、下划线（_）三者之一开头，同时 JavaScript 对大小写敏感，大小写不同也就意味着是不同的两个变量。同时，JavaScript 不区分单引号与双引号，因此上一个例子与用单引号表示的效果一致：

```
var a = 'node.js';
var b = 10;
```

5. 变量提升机制

JavaScript 中存在变量提升机制，也就是所有的变量声明在运行时都会提升到代码的最前方。例如，上个例子在运行时实际上会先声明两个变量再赋值：

```
var a;
var b;
```

```
a = "node.js";
b = 10;
```

通过一个更直观的例子或许会让读者更容易理解变量提升。在 REPL 中试图使用一个未声明的变量时会出现 is not defined 错误，如图 3.3 所示。

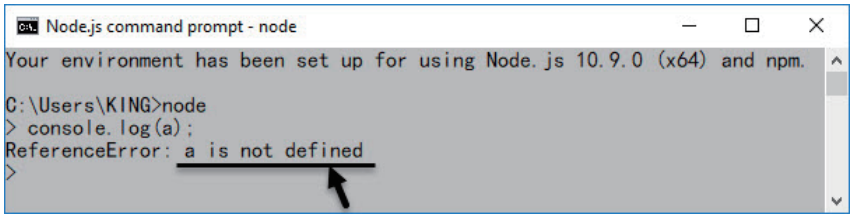

图 3.3　变量未声明错误

如果试图使用一个已经声明却未赋值的变量，那么这个变量所代表的是 undefined，如图 3.4 所示。

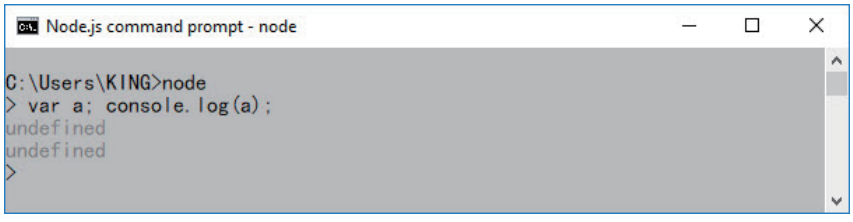

图 3.4　已经声明却未赋值

图 3.4 中有两行 undefined，在执行 node 语句时，在没有任何返回值的时候总是会输出一个 undefined，读者不必介意，这不是错误，而是正常输出的内容。

使用一个在后来定义赋值的变量时会返回 undefined，如图 3.5 所示。

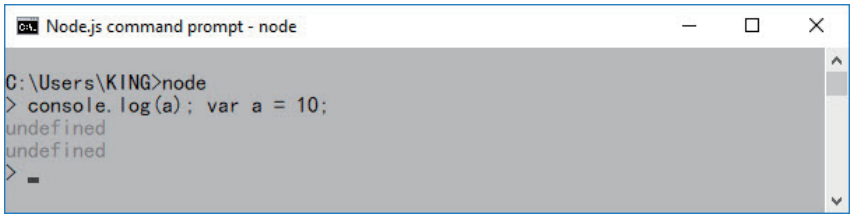

图 3.5　先使用后声明赋值

可以发现这段代码返回 undefined 正是因为变量提升，实际运行的代码如下：

```
var a;
console.log(a);
a = 10;
```

3.1.2 注释

JavaScript 中的注释和很多其他编程语言类似，以双斜杠（//）代表单行注释，以“/*注释内容*/”形式代表多行注释。

```
// 这是单行注释
/* 这是
   多行
   注释
*/
```

3.1.3 数据类型

JavaScript 中的数据类型可以分为简单数据类型和复杂数据类型。简单数据类型有 undefined、boolean、number、string、null，复杂数据类型只有 object。object 由一组无序的键值对组成。

1. 利用 typeof 区分数据类型

利用操作符 typeof 可以区分数据类型。typeof 返回的值有 undefined、boolean、number、string、object 和 function。下面举一个例子。

【示例 3-1】

```
var a;
var b = 12;
var c = 'node.js';
var d = true;
var e = function() {

}
var f = null;
var g = {
    num: 12
}
var arr = [a,b,c,d,e,f,g];
for(var i = 0, max = arr.length; i < max; i++) {
    console.log(typeof arr[i]);
}

// undefined
// number
// string
// boolean
// function
// object
```

```
// object
```

【代码解析】

可以看到 null 和 object 都返回了 object，这是因为 null 实际上是一个空对象指针，当一个变量只声明未赋值时返回 undefined。

number 和 string 数据类型分别指数字类型和字符串类型；boolean 类型和其他语言一样，仅有 true 和 false 两个值；null 仅有一个值 null。

2. 利用 Boolean()转化数据类型

JavaScript 中可以利用 Boolean()方法将其他数据类型转化为布尔值。需要注意的是，空字符串、0、null、undefined、NaN 都将转化为 false，其他值则会转化为 true。下面举一个例子。

【示例 3-2】

```
var a;
var b = null;
var c = 0;
var d = '';
var e = NaN;
var arr = [a,b,c,d,e];
for(var i = 0, max = arr.length; i < max; i++) {
    console.log(Boolean(arr[i]));
}

// false
// false
// false
// false
// false
```

3.1.4　函数

在 JavaScript 中，声明一个函数只需要使用 function 关键字即可，如声明一个求和的函数，代码如下：

```
function add(num1, num2) {
    return num1 + num2;
}
```

当然，函数同样可以作为一个值传递给一个变量，例如：

```
var add = function(num1, num2) {
    return num1 + num2;
}
```

调用一个函数同样很简单，只需要在函数声明之后使用“函数名(参数)”的形式调用即可，

如调用上面的函数：

```
function add(num1, num2) {
    return num1 + num2;
}
add(1, 2);
// 3

add(3,5);
//8
```

函数中默认带有一个 arguments 对象，这是一个类数组对象。arguments 记录了传递给函数的参数信息，因为 JavaScript 中的函数调用时，参数个数并不需要和定义函数时的个数一致。在上面的 add()方法中多添加几个参数，函数仍然会正常执行，例如：

```
function add(num1, num2) {
   return num1 + num2;
}
add(1,2,4,5,5);
// 3

add(3,5,2,3);
//8
```

利用好这一点和 arguments 类数组的特性可以对上述的 add 方法拓展一下，让这个函数无论接收多少个参数，总能返回这些数值的和：

```
function add() {
   var sum = 0;
   for(var i = 0, max = arguments.length; i < max; i++) {
      sum += arguments[i];
   }
   return sum;
}

add(2,3,4);
// 9

add(2,4,5);
// 11
```

还可以利用 JavaScript 中的 arguments 类数组对象模拟函数重载。当然，实际上 JavaScript 并不支持函数重载，比如通过检测 arguments 对象的 length 属性做出不同的反应来模拟重载。下面给出一个完整的例子。

【示例 3-3】

```
function operate() {
    if(arguments.length == 2) {
        return arguments[0] * arguments[1];
    } else {
        var sum = 0;
        for(var i = 0, max = arguments.length; i < max; i++) {
            sum += arguments[i];
        }
        return sum;
    }
}
operate(3, 4);
// 12
operate(3, 4, 5);
// 12
```

以上代码的运行效果如图 3.6 所示。

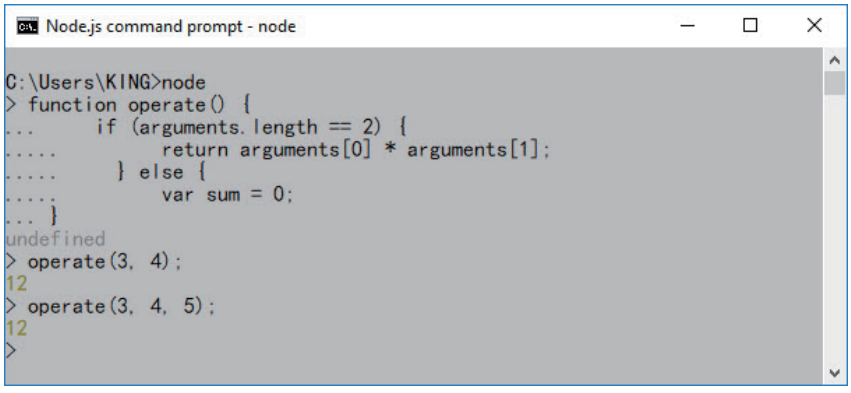

图 3.6　运行效果

【代码解析】

arguments 对象是一个类数组对象。通过数组的 slice()方法可以把 arguments 对象转化为一个真正的数组，这样就可以使用数组的所有方法，而不用担心出现其他问题了。

```
function funName() {
    var arguments = [].slice.call(arguments);
    // the code of function
}
```

3.1.5　闭包

JavaScript 中的变量可以分为全局变量和局部变量。JavaScript 中的函数自然可以读取到全

局变量，而函数外部并不能读取到函数内部定义的变量，例如：

```
var str = 'node.js';

function copy () {
    var str2 = str;
    console.log(str2);
}

copy();
// node.js

console.log(str2);
// str2 is not defined
```

当然，这需要在定义变量的时候使用 var 关键字定义。不用 var 关键字定义的话，实际上这个变量会成为全局对象的一个属性。在 Node.js 中，全局对象是 global，如果上面代码中的 str2 变量不使用 var 定义，str2 就会成为一个全局变量，函数外部也是可以读取到这个变量的，例如：

```
var str = 'node.js';

function copy () {
    str2 = str;
    console.log(str2);
}

copy();
// node.js

console.log(str2);
// node.js
```

建议所有的变量都使用 var 关键字进行定义，以避免出现不必要的错误。

JavaScript 中的闭包可以让函数读取到其他函数内部的变量，如下代码就可以让函数外部读取到函数内部定义的变量，这就是最简单的闭包。

【示例 3-4】

```
function a() {
    var str = 'node.js';
    return function() {
        var str2 = str + ' is poserful';
        return str2;
```

```
    }
}

a()();
// node.js is powerful
```

以上就是 JavaScript 的简要介绍。更多关于 JavaScript 的知识，读者可以阅读相关的书籍进行学习和掌握。进行 Node.js 的学习之前，读者应该对 JavaScript 有一定的了解。

3.2 命名规范与编程规范

与其他语言相比，JavaScript 显得相对灵活，对代码的格式要求也相对宽松，因此对 JavaScript 编码制定一定的规范是非常重要的。一个良好的规范不仅能让阅读代码的人感到清晰愉悦，还能让整个项目更加容易维护。

3.2.1 命名规范

JavaScript 作为一种弱类型的语言，命名的规范显得更加重要，因为开发人员并不能直接看出这个变量的作用。

1. var 关键字

在 JavaScript 中，所有的变量都应该通过 var 关键字来定义，而不是缺少 var 关键字，因为缺少 var 的变量声明会使得这个变量成为全局变量，在开发中应尽量减少全局变量：

```
var a = 'node.js';
// 推荐
b = 12
// 不推荐
```

2. 驼峰命名法

在开发中，变量的命名常常是让开发人员头疼的问题，一般来说每个团队都会有自己的命名规范。近些年来更加流行的是驼峰命名法。如它的名字一样，驼峰命名法中第一个单词的开头小写，其他单词的开头字符大写，例如：

```
var myNumber;
var myString;
```

3. 常量

在其他语言中，会有常量这样一个概念。这是一种不允许在声明赋值之后再修改的变量。显然，JavaScript 中有着同样的需求。在开发人员不希望有些变量得到修改时，常量就显得格

外重要了。例如，定义一个圆周率的常量。

在常量的命名中，开发人员往往采用变量名全部大写的方式来表示这是一个常量。当然，实际上这样的变量依旧是可以被修改的。这需要开发人员共同遵守规定，把这样一个变量作为一个常量，而不是普通的 JavaScript 变量：

```
var PI = 3.14159
// 这是一个圆周率的常量
```

4. 内部变量

开发中还有一类就是内部变量。这类变量并不希望局部作用域之外的作用域来获取这些变量。开发人员通常以下划线“_”开头命名作为约定俗成的内部变量。当然，这同样需要协同的开发人员共同遵守这个约定：

```
var obj ={
    _num: 12,
    // 这是一个内部变量
    put: function() {
        return this._num;
    }
}
console.log(obj.put())
```

5. 有意义的名字

命名规范中同样需要遵守的是，在命名中不应该使用一些无意义的变量名。这些无意义的变量名往往会使开发人员摸不着头脑。变量的命名应该是有意义的、能够表示变量作用的，而不是无意义的、混淆视听的。

3.2.2 编程规范

在 JavaScript 中，遵守编程规范会使得整个项目得到更快的开发和更好的维护。同时，也可以让代码看起来更加优雅易读。

1. 以分号结尾

在 JavaScript 代码中，所有的语句都应该以分号结尾，虽然 JavaScript 中并没有强制要求：

```
var n = 12;
// 以分号结尾，推荐
var n = 12
// 结尾缺少分号，不推荐
```

2. 大括号

JavaScript 4E2D 的所有语句块都应该有大括号，比如一个简单的 if 判断语句块里面只有

一行时，不使用大括号并不会出现错误，但是这往往会让开发人员产生疑惑：

```
var n = 12;
if(n < 10) {
    console.log(n);
}
// 即使语句块里只有一行代码，也应该有大括号
```

3. ===

相等判断中应该尽量使用绝对等于“===”，因为等于“==”存在类型转化，这在开发中可能会出现意想不到的错误。例如，使用“==”来判断 null 与 undefined 时会出现 true 的情况，而使用“===”则不会：

```
console.log(1 == true);
// true
console.log(1 === true);
// false
console.log(null == undefined);
// true
console.log(null === undefined);
// false
```

4. 空格的使用

关于 JavaScript 中的空格，永远不要吝啬，因为满屏连续的字符串会让人头疼。建议在数值操作符（如+、-、*、/、%等）前后留一个空格，在赋值操作符和相等判断中前后留一个空格。在 json 对象中，键值对的冒号后应该留一个空格，如以下空格会让代码在开发人员的眼中更加优雅：

```
var num = 12;
// 赋值操作符前后留空格
if(num === 12){
  // your coding
}
// 相等判断中前后留出空格
var num2 = num * 2;
// 数值操作符前后留出空格
var obj = {
  name: 'node.js'
}
// 键值对冒号后面留出一个空格
```

关于 JavaScript 中的注释，永远记住一条：所有的注释都应该是有意义的，无意义的注释只会让阅读代码的人员感到更加困惑。

关于 JavaScript 中的编程规范就简单介绍到这里。相比于别人介绍的规范，拥有一套属于自己团队内部的使用规范并且让开发人员遵守这个规范更加重要。

3.3 Node.js 的控制台

利用好 Node.js 提供的控制台和 Debug 可以有效地辅助开发和定位 Bug。在 Node.js 中，Console 代表控制台，可以通过 console 对象的各种方法向控制台进行标准输出。

3.3.1 console 对象下的各种方法

在 REPL 交互式运行环境中输入 console，可以看到 console 对象下各种方法组成的一个数组，如图 3.7 所示。

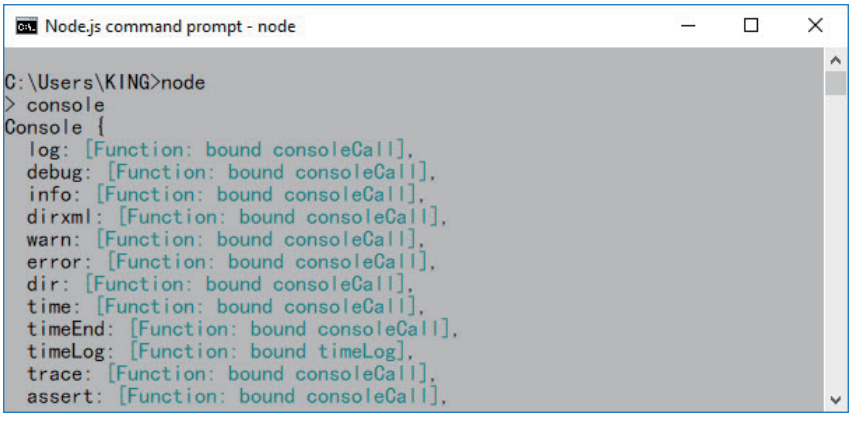

图 3.7 console 对象的方法

3.3.2 console.log()方法

console.log()方法用于标准输出流的输出，也就是在控制台中显示一行信息，例如：

```
console.log('node.js is powerful')
```

无论是在 RPEL 环境中运行这行代码还是作为 Node.js 文件执行这行代码，都可以看到控制台输出了“node.js is powerful”字样。

console.log()方法并没有对参数的个数进行限制，当传递多个参数时，控制台输出时将以空格分隔这些参数，例如：

```
console.log('node.js','is','powerful');
```

运行之后，同样会在控制台输出“node.js is powerful”字样，这三个单词依旧是以空格分隔开来的。

console.log()方法也可以利用占位符来定义输出的格式，如%d 表示数字、%s 表示字符串。

如果需要对后面的多个参数定义格式，就要逐个设置，并且输出时将不会再以空格分隔；如果没有预定义格式，就将正常输出。

示例代码如下：

```
console.log('%s%s', 'node.js', 'is', 'powerful');
// node.jsis powerful

console.log('%s%s%s', 'node.js', 'is', 'powerful');
// node.jsispowerful

console.log('%d', 'node.js');
// NaN

console.log('%d', 'node.js', 'is', 'powerful');
// NaN is powerul
```

在这一段代码中，需要注意的是当使用%d 占位符后，如果对应的参数不是数字，控制台将会输出 NaN。

3.3.3　console.info()、console.warn()和 console.error()方法

console.info()、console.warn()以及 console.error()的使用方法和 console.log()一致，将 3.3.2 小节的代码换成 console.info()、console.warn()、console.error()方法，将得到同样的结果：

```
console.warn('%s%s', 'node.js', 'is', 'powerful');
// node.jsis powerful

console.warn('%s%s%s', 'node.js', 'is', 'powerful');
// node.jsispowerful

console.info('%d', 'node.js');
// NaN

console.info('%d', 'node.js', 'is', 'powerful');
// NaN is powerul

   console.error('%d', 'node.js');
// NaN

console.error('%d', 'node.js', 'is', 'powerful');
// NaN is powerul
```

3.3.4 console.dir()方法

console.dir()方法用于将一个对象的信息输出到控制台。如下代码将定义一个简单的对象。

【示例 3-5】

```
const obj = {
   name: 'node.js',
   get: function() {
     console.log('get');
   },
   set: function() {
     console.log('set');
   }
}
console.dir(obj);
```

在 RPEL 交互运行环境中运行这段代码，可以看到控制台输出了这个对象的信息，如图 3.8 所示。

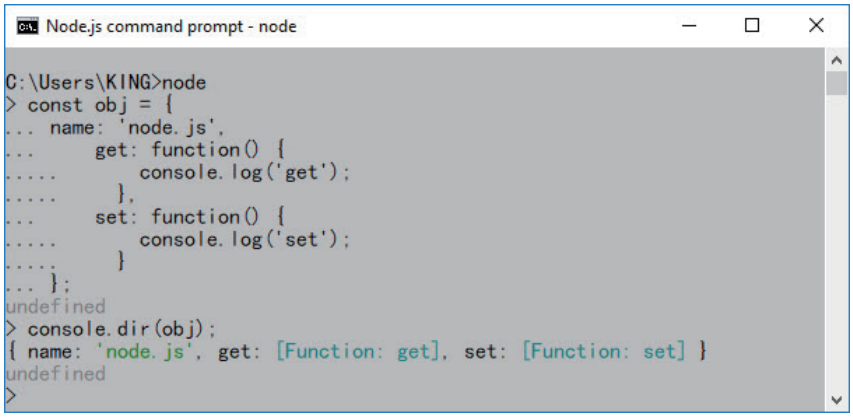

图 3.8 console.dir()方法输出对象信息

3.3.5 console.time()和 console.timeEnd()方法

console.time()和 console.timeEnd()方法主要用于统计一段代码运行的时间。console.time()方法置于代码起始处，console.timeEnd()方法置于代码结尾处。只需要向这两个方法传递同一个参数，就可以看到在控制台中输入了以毫秒计的代码运行时间。如下代码统计了两个循环执行后的时间以及各个循环分别使用的时间。

【示例 3-6】

```
console.time('total time');

console.time('time1');
for(var i =0; i< 10000; i++) {

}
```

```
console.timeEnd('time1');

console.time('time2');
for(var i =0; i< 100000; i++) {

}
console.timeEnd('time2');

console.timeEnd('total time');
```

将这段代码保存为名为 time.js 的文件。利用 node time.js 命令运行这个文件，可以在控制台看到各个循环的使用时间统计，如图 3.9 所示。

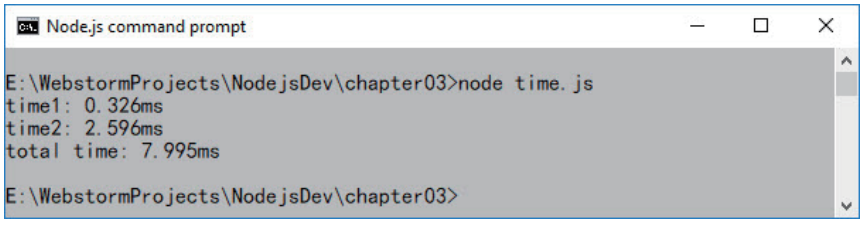

图 3.9　各个循环的使用时间统计

3.3.6　console.trace()方法

console.trace()方法用于输出当前位置的栈信息，可以向 console.trace()方法传递任意字符串作为标志，类似于 console.time()中的参数。在 RPEL 交互运行环境中执行以下代码：

```
console.trace('trace');
```

可以看到此处的栈信息已经在控制台中输出了，如图 3.10 所示。

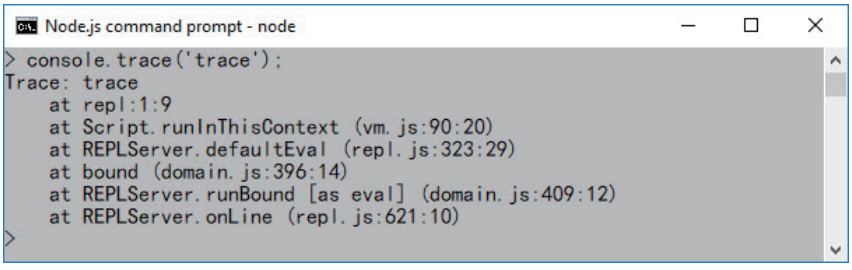

图 3.10　console.trace()输出栈信息

3.3.7　console.table()方法

console.table()用于将数组格式的信息以表格（table）形式进行输出，可以向 console.table()方法传递任意结构形式的数组信息，譬如对象数组等。console.table()方法在 RPEL 交互运行环境中执行以下代码：

```
console.table(tabularData[, properties]);
```

如下代码使用 console.table()方法将一个对象数组以表格的形式进行输出。

【示例 3-7】

```
var arrTable = {
    A: {no : "1", name : "Apple"},
    B: {no : "2", name : "Google"},
    C: {no : "3", name : "Microsoft"}
};

console.table(arrTable);

console.table(arrTable, "name");
```

将这段代码保存为名为 table.js 的文件。利用 node table 命令运行这个文件，可以在控制台看到各个循环的使用时间统计，如图 3.11 所示。

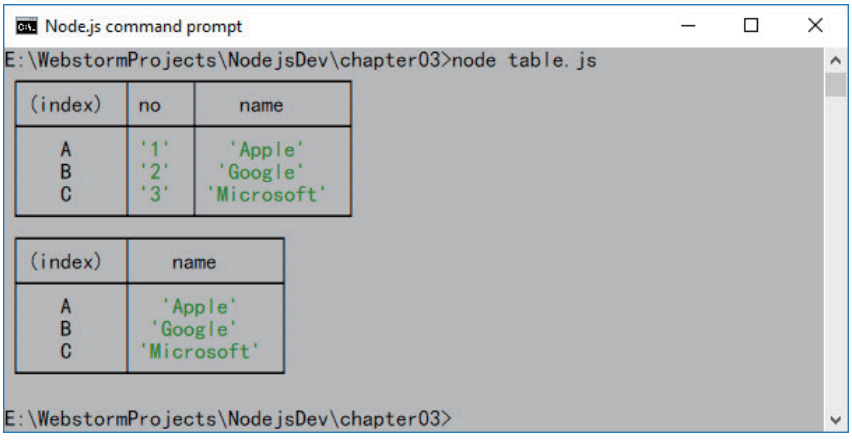

图 3.11　console.table()输出表格信息

3.4 温故知新

学完本章，读者需要回答：

1. 简要介绍 JavaScript 的语法。

2. JavaScript 中将其他数据类型转化为布尔值时需要注意哪些地方？

3. 文中提到 JavaScript 的编程规范有哪些？如果由你来主导一个 JavaScript 项目，你会怎样制定这个项目的规范？

4. Node.js 中的控制台如何统计代码的运行时间？

5. Node.js V10 版本中新增的 console.table()方法如何使用？