TURING 图灵原创

BYVoid 编著

Node.js 开发指南

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵原创

BYVoid 编著

Node.js 开发指南

人民邮电出版社
北 京

图书在版编目（CIP）数据

Node.js开发指南 / 郭家宝编著. — 北京 : 人民邮电出版社, 2012.7
（图灵原创）
ISBN 978-7-115-28399-3

Ⅰ. ①N… Ⅱ. ①郭… Ⅲ. ①JAVA语言－程序设计－指南 Ⅳ. ①TP312-62

中国版本图书馆CIP数据核字(2012)第110717号

内 容 提 要

本书首先简要介绍 Node.js，然后通过各种示例讲解 Node.js 的基本特性，再用案例式教学的方式讲述如何用 Node.js 进行 Web 开发，接着探讨一些 Node.js 进阶话题，最后展示如何将一个 Node.js 应用部署到生产环境中。

本书面向对 Node.js 感兴趣，但没有基础的读者，也可供已了解 Node.js，并对 Web 前端 / 后端开发有一定经验，同时想尝试新技术的开发者参考。

图灵原创

Node.js开发指南

◆ 著　　BYVoid
责任编辑　王军花
执行编辑　丁晓昀

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京天宇星印刷厂印刷

◆ 开本：800×1000　1/16
印张：11.75
字数：249千字　　2012年7月第1版
印数：1－5 000册　　2012年7月北京第1次印刷

ISBN 978-7-115-28399-3

定价：45.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223
反盗版热线：(010)67171154

站在巨人的肩上
Standing on Shoulders of Giants

www.ituring.com.cn

前　言

这本书讲了什么

本书是一本Node.js的入门教程，写给想了解Node.js的开发人员。我的目标是使读者通过阅读本书，学会使用Node.js进行Web后端开发，同时能熟悉事件驱动的异步式编程风格，以便进一步了解Node.js的许多高级特性，以及它所应用的更多领域。

本书共6章，分别讨论了Node.js的背景、安装和配置方法、基本特性、核心模块以及一些进阶话题。除此之外，还有2个附录，分别介绍了JavaScript的高级特性和Node.js编程规范。下面简要概述各章的主要内容。

第1章“Node.js简介”

这一章概述了什么是Node.js。读过这章后，你将对Node.js有一个基本的认识，同时了解它与JavaScript的深厚渊源。

第2章“安装和配置Node.js”

这一章讲述了如何在各种不同的环境下安装和配置Node.js及其基本运行环境，同时你你可以了解到如何编译Node.js，以及多版本管理工具。

第3章“Node.js快速入门”

这一章讲解Node.js的基础知识，你将会学到如何使用Node.js的基本环境和工具进行开发、运行和调试。同时，还会讲解异步式I/O与事件式编程的一些重要概念，这些概念将会贯穿全书。此外这一章还详细介绍了Node.js的模块和包的系统，这些都是开发中经常会碰到的内容。

第4章“Node.js核心模块”

这一章以全局对象、基本工具、事件发射器、文件系统和HTTP为代表，介绍了Node.js

最常用的核心模块。你将会在后面的章节及以后的开发中经常与这些模块打交道。

第 5 章“使用 Node.js 进行 Web 开发”

这一章是本书的实践性章节，一步一步教你如何从零开始用 Express 框架创建一个网站，实现路由控制、模板解析、会话管理、数据库访问等功能，最终创建一个 Web 2.0 微博网站。

第 6 章“Node.js 进阶话题”

这一章涉及几个进阶话题，包括模块加载机制、控制流分析和优化、生产环境的应用部署等内容，最后还讨论了 Node.js 适用的范围，帮助读者在今后的开发中作出更好的取舍。

附录 A“JavaScript 的高级特性”

这个附录介绍了 JavaScript 的一些高级特性，如函数作用域、闭包和对象的操作等内容。这些特性在浏览器端的JavaScript 开发中并没有受到应有的重视，而在 Node.js 中却十分常见，阅读这个附录可以帮助你更好地理解并运用 JavaScript 进行复杂的网站开发。

附录 B“Node.js 编程规范”

这个附录介绍了 Node.js 代码风格的一些约定，遵守这些约定可以让你的代码更清晰、易懂，同时也有利于接口开发的统一。该附录还分享了一些开发经验，可以让程序避免很多意外错误和性能损失。

谁应该阅读本书

本书的目标读者是想要学习 Node.js，但没有任何系统的经验的开发者。如果你听说过 Node.js，并被它许多神奇的特性吸引，那么这本书就是为你准备的。通过阅读本书，你可以对 Node.js 有全面的认识，学会如何用 Node.js 编程，了解事件驱动、异步式 I/O 的编程模式，同时还可以掌握一些使用 JavaScript 进行函数式编程的方法。

本书假设读者已经学过至少一门编程语言，对基本的程序设计语言概念（如变量、函数、递归、对象）有所了解。如果你是首次学习编程语言，我建议你先学一门常见的且容易入门的语言，如 Java 或 C。

如何阅读本书

熟悉浏览器端 JavaScript 的读者将很容易学会 Node.js 的许多特性，包括事件式编程、闭包、回调函数等，因为这些特性已经在浏览器中被广泛应用。同时，你还可以学到 Node.js

在Web开发中的服务器端与浏览器端的结合方式，这无论是对前端设计还是后端开发都是有利的。你还会对 JavaScript 有一个全新的认识，因为服务端的 JavaScript 中没有 DOM 和 BOM，也不存在浏览器之间的兼容性问题。

不熟悉 JavaScript但是了解C、Java、C++、C#的读者将很容易学会 JavaScript 的语言特性及 Node.js 的基本机制，如模块和包。你需要关注的仅仅是 JavaScript 语言的特别之处，以及服务器端开发中需要注意的一些要点。

已经非常了解 Web 后端开发（如 PHP、ASP.net、Ruby on Rails、Django 等）的读者，本书将通过 Node.js 给你一个不同的视野。你会发现 Node.js 和这些传统的框架有很大的区别，因为它使用了事件式编程和异步 I/O，所以你需要改变一些已有的思维方式。同时，你还能享受到 Web 前后端紧密配合带来的新鲜感，并可能对 Ajax 有全新的认识。

如果是完全没有接触过JavaScript的读者，那么我建议你看完本书的前两章以后，花点时间到http://www.w3school.com.cn/js/网站看看 JavaScript 的入门教程。你只要了解基础知识就行了，本书并不要求你学成一个JavaScript专家。在这之后请阅读本书的附录A，了解一下实际开发中可能会遇到的稍微复杂的语言特性。附录A是为本书量身定做的，你可以从中很快地学会 Node.js 经常使用到的那些特性。如果你想更加深入系统地学习JavaScript，推荐阅读 Mozilla JavaScript指南http://developer.mozilla.org/en/JavaScript/Guide。

本书从第3章开始，将介绍如何用 Node.js 开发，你应该仔细阅读这一章。第4章是一些最基本的模块介绍，涉及Node.js 模块的基本风格，这可能会帮助你理解后面介绍的 API。第5章是一个真枪实弹的实战演练，跟随这一章的每个步骤你就可以用 Node.js 实现一个真正的 Web 应用，体验开发的成就感。第6章则是一些进阶话题，你会在这里接触到 Node.js的一些深层次概念，同时你还将学会如何真正部署 Node.js 应用。

本书的每一章最后都有一个参考资料小节，里面有很多有价值的资料，如果感兴趣不妨继续深入阅读。在阅读本书的过程中，我建议你抽时间看看附录B，在这里你会了解到Node.js开发的一些编程规范，写出符合社区风格的漂亮程序。

如何学习 Node.js

通读本书，你将会学到 Node.js 的很多东西，但如果想完全掌握它，我建议你亲自尝试运行本书中的每一段代码。本书的所有代码可以在http://www.byvoid.com/project/node上找到。[①] 除此之外，你最好自己用 Node.js 做一个项目，因为通过实践你会遇到很多问题，解决这些问题可以大大加深对 Node.js 的理解。

注意，不要忘了互联网网上的资源，比如Node.js 的官方 API 文档http://nodejs.org/api/。我强烈推荐你去 CNodeJS 社区看看http://cnodejs.org/，这里汇集了许许多多中国优秀的

① 读者也可以到图灵社区（iturng.com.cn）本书的页面上下载源代码或提交勘误。——编者注

Node.js 开发者。他们每天都在讨论着大量有关Node.js各个方面的话题，你可以在上面获得很多帮助。同时，CNodeJS 社区的网站也是用 Node.js 写成的，而且是开源的，它是一个非常好的让你了解如何用 Node.js 开发网站的实例。

体例说明

本书正文中出现的代码引用都会以等宽字体标出，例如：`console.log('Node.js')`。代码段会以段落的形式用等宽字体显示，例如：

```
function hello() {
  console.log('Hello, world!');
}
```

在正文之中，偶尔还会穿插一些提示和警告，例如：

这是一个提示。

这是一个警告。

致谢

感谢对这本书提出宝贵意见的朋友们，他们是牟瞳、李垚[①]、周越、钟音、萧骐[②]、杨旭东、孙嘉龙、范泽一、宋文杰、续本达、田劲锋、孟亚兰和李宇亮。他们为本书的结构、内容、语言表述等方面给出了许多有建设性的意见。

感谢 CNodeJS 社区的贾超、田永强和微软亚洲研究院的杨懋，以及VMware公司的柴可夫。他们不仅帮助审阅了本书，还解决了许多技术问题，给这本书提出了许多改进方案。

感谢弓辰开发的 Rime 输入法[③]，我用它完成了本书的创作。

还要感谢我的朋友徐可可，图灵公司的杨海玲、谢工、王军花以及各位编辑，她们给我提供了许多帮助和鼓舞，没有她们的激励，我很难顶着巨大的学业压力坚持写完这本书。

郭家寶

① 李垚是果壳网的作者之一，他的个人网站是http://www.liyaos.com/。

② 萧骐是*Dive into Python*的译者，活跃在 linuxtoy http://linuxtoy.org/。

③ Rime 是一个优秀的开源输入法，它不仅支持繁体和简体的拼音输入，而且是跨平台的，可以在 Windows、Linux、Mac上使用，其网址是：http://code.google.com/p/rimeime/。

目　录

Node.js简介

第 1 章

Node.js，或者 Node，是一个可以让 JavaScript 运行在服务器端的平台。它可以让JavaScript脱离浏览器的束缚运行在一般的服务器环境下，就像运行 Python、Perl、PHP、Ruby程序一样。你可以用 Node.js 轻松地进行服务器端应用开发，Python、Perl、PHP、Ruby 能做的事情 Node.js 几乎都能做，而且可以做得更好。

Node.js 是一个为实时Web（Real-time Web）应用开发而诞生的平台，它从诞生之初就充分考虑了在实时响应、超大规模数据要求下架构的可扩展性。这使得它摒弃了传统平台依靠多线程来实现高并发的设计思路，而采用了单线程、异步式I/O、事件驱动式的程序设计模型。这些特性不仅带来了巨大的性能提升，还减少了多线程程序设计的复杂性，进而提高了开发效率。

Node.js 最初是由 Ryan Dahl 发起的开源项目，后来被 Joyent 公司注意到。Joyent 公司将Ryan Dahl 招入旗下，因此现在的 Node.js 由 Joyent 公司管理并维护。尽管它诞生的时间（2009年）还不长，但它的周围已经形成了一个庞大的生态系统。Node.js 有着强大而灵活的包管理器（node package manager，npm），目前已经有上万个第三方模块，其中有网站开发框架，有 MySQL、PostgreSQL、MongoDB 数据库接口，有模板语言解析、CSS 生成工具、邮件、加密、图形、调试支持，甚至还有图形用户界面和操作系统 API工具。由 VMware 公司建立的云计算平台 Cloud Foundry 率先支持了 Node.js。2011年6月，微软宣布与 Joyent 公司合作，将 Node.js 移植到 Windows，同时 Windows Azure 云计算平台也支持 Node.js。Node.js 目前还处在迅速发展阶段，相信在不久的未来它一定会成为流行的Web应用开发平台。让我们从现在开始，一同探索 Node.js 的美妙世界吧！

1.1　Node.js 是什么

Node.js 不是一种独立的语言，与 PHP、Python、Perl、Ruby 的“既是语言也是平台”不同。Node.js 也不是一个 JavaScript 框架，不同于 CakePHP、Django、Rails。Node.js 更不是浏览器端的库，不能与 jQuery、ExtJS 相提并论。Node.js 是一个让 JavaScript 运行在服务端的开发平台，它让 JavaScript 成为脚本语言世界的一等公民，在服务端堪与 PHP、Python、Perl、Ruby 平起平坐。

Node.js 是一个划时代的技术，它在原有的 Web 前端和后端技术的基础上总结并提炼出了许多新的概念和方法，堪称是十多年来 Web 开发经验的集大成者。Node.js 可以作为服务器向用户提供服务，与 PHP、Python、Ruby on Rails 相比，它跳过了 Apache、Nginx 等 HTTP服务器，直接面向前端开发。Node.js 的许多设计理念与经典架构（如 LAMP）有着很大的不同，可提供强大的伸缩能力，以适应21世纪10年代以后规模越来越庞大的互联网环境。

Node.js 与 JavaScript

说起 JavaScript，不得不让人想到浏览器。传统意义上，JavaScript 是由 ECMAScript、

文档对象模型（DOM）和浏览器对象模型（BOM）组成的，而 Mozilla 则指出 JavaScript 由 Core JavaScript 和 Client JavaScript 组成。之所以会有这种分歧，是因为 JavaScript 和浏览器之间复杂的历史渊源，以及其命途多舛的发展历程所共同造成的，我们会在后面详述。我们可以认为，Node.js 中所谓的 JavaScript 只是 Core JavaScript，或者说是 ECMAScript 的一个实现，不包含 DOM、BOM 或者 Client JavaScript。这是因为 Node.js 不运行在浏览器中，所以不需要使用浏览器中的许多特性。

Node.js 是一个让 JavaScript 运行在浏览器之外的平台。它实现了诸如文件系统、模块、包、操作系统 API、网络通信等 Core JavaScript 没有或者不完善的功能。历史上将 JavaScript 移植到浏览器外的计划不止一个，但Node.js 是最出色的一个。随着 Node.js 的成功，各种浏览器外的 JavaScript 实现逐步兴起，因此产生了 CommonJS 规范。CommonJS 试图拟定一套完整的 JavaScript 规范，以弥补普通应用程序所需的 API，譬如文件系统访问、命令行、模块管理、函数库集成等功能。CommonJS 制定者希望众多服务端 JavaScript 实现遵循 CommonJS 规范，以便相互兼容和代码复用。Node.js 的部份实现遵循了CommonJS规范，但由于两者还都处于诞生之初的快速变化期，也会有不一致的地方。

Node.js 的 JavaScript 引擎是 V8，来自 Google Chrome 项目。V8 号称是目前世界上最快的 JavaScript 引擎，经历了数次引擎革命，它的 JIT（Just-in-time Compilation，即时编译）执行速度已经快到了接近本地代码的执行速度。Node.js 不运行在浏览器中，所以也就不存在 JavaScript 的浏览器兼容性问题，你可以放心地使用 JavaScript 语言的所有特性。

1.2 Node.js 能做什么

正如 JavaScript 为客户端而生，Node.js 为网络而生。Node.js 能做的远不止开发一个网站那么简单，使用 Node.js，你可以轻松地开发：

- 具有复杂逻辑的网站；
- 基于社交网络的大规模 Web 应用；
- Web Socket 服务器；
- TCP/UDP 套接字应用程序；
- 命令行工具；
- 交互式终端程序；
- 带有图形用户界面的本地应用程序；
- 单元测试工具；
- 客户端 JavaScript 编译器。

Node.js 内建了 HTTP 服务器支持，也就是说你可以轻而易举地实现一个网站和服务器的组合。这和 PHP、Perl 不一样，因为在使用 PHP 的时候，必须先搭建一个 Apache 之类的

HTTP服务器，然后通过HTTP服务器的模块加载或CGI调用，才能将PHP脚本的执行结果呈现给用户。而当你使用Node.js时，不用额外搭建一个HTTP服务器，因为Node.js本身就内建了一个。这个服务器不仅可以用来调试代码，而且它本身就可以部署到产品环境，它的性能足以满足要求。

Node.js还可以部署到非网络应用的环境下，比如一个命令行工具。Node.js还可以调用C/C++的代码，这样可以充分利用已有的诸多函数库，也可以将对性能要求非常高的部分用C/C++来实现。

1.3　异步式I/O与事件驱动

Node.js最大的特点就是采用异步式I/O与事件驱动的架构设计。对于高并发的解决方案，传统的架构是多线程模型，也就是为每个业务逻辑提供一个系统线程，通过系统线程切换来弥补同步式I/O调用时的时间开销。Node.js使用的是单线程模型，对于所有I/O都采用异步式的请求方式，避免了频繁的上下文切换。Node.js在执行的过程中会维护一个事件队列，程序在执行时进入事件循环等待下一个事件到来，每个异步式I/O请求完成后会被推送到事件队列，等待程序进程进行处理。

例如，对于简单而常见的数据库查询操作，按照传统方式实现的代码如下：

```
res = db.query('SELECT * from some_table');
res.output();
```

以上代码在执行到第一行的时候，线程会阻塞，等待数据库返回查询结果，然后再继续处理。然而，由于数据库查询可能涉及磁盘读写和网络通信，其延时可能相当大（长达几个到几百毫秒，相比CPU的时钟差了好几个数量级），线程会在这里阻塞等待结果返回。对于高并发的访问，一方面线程长期阻塞等待，另一方面为了应付新请求而不断增加线程，因此会浪费大量系统资源，同时线程的增多也会占用大量的CPU时间来处理内存上下文切换，而且还容易遭受低速连接攻击。

看看Node.js是如何解决这个问题的：

```
db.query('SELECT * from some_table', function(res) {
  res.output();
});
```

这段代码中db.query的第二个参数是一个函数，我们称为*回调函数*。进程在执行到db.query的时候，不会等待结果返回，而是直接继续执行后面的语句，直到进入事件循环。当数据库查询结果返回时，会将事件发送到事件队列，等到线程进入事件循环以后，才会调用之前的回调函数继续执行后面的逻辑。

Node.js的异步机制是基于事件的，所有的磁盘I/O、网络通信、数据库查询都以非阻塞

的方式请求，返回的结果由事件循环来处理。图1-1 描述了这个机制。Node.js 进程在同一时刻只会处理一个事件，完成后立即进入事件循环检查并处理后面的事件。这样做的好处是，CPU 和内存在同一时间集中处理一件事，同时尽可能让耗时的 I/O 操作并行执行。对于低速连接攻击，Node.js 只是在事件队列中增加请求，等待操作系统的回应，因而不会有任何多线程开销，很大程度上可以提高 Web 应用的健壮性，防止恶意攻击。

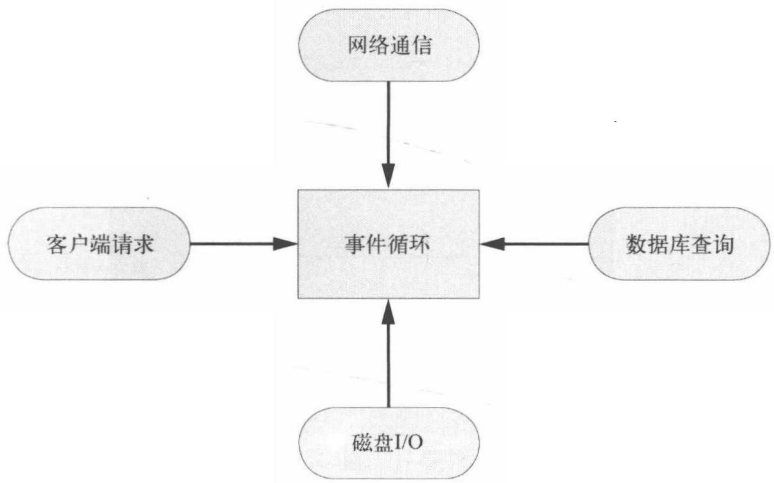

图1-1 事件循环

这种异步事件模式的弊端也是显而易见的，因为它不符合开发者的常规线性思路，往往需要把一个完整的逻辑拆分为一个个事件，增加了开发和调试难度。针对这个问题，Node.js 第三方模块提出了很多解决方案，我们会在第6章中详细讨论。

1.4 Node.js 的性能

1.4.1 Node.js 架构简介

Node.js 用异步式 I/O 和事件驱动代替多线程，带来了可观的性能提升。Node.js 除了使用 V8 作为JavaScript引擎以外，还使用了高效的 libev 和 libeio 库支持事件驱动和异步式 I/O。图1-2 是 Node.js 架构的示意图。

Node.js 的开发者在 libev 和 libeio 的基础上还抽象出了层 libuv。对于 POSIX[①]操作系统，libuv 通过封装 libev 和 libeio 来利用 epoll 或 kqueue。而在 Windows 下，libuv 使用了 Windows

① POSIX（Portable Operating System Interface）是一套操作系统 API 规范。一般而言，遵守 POSIX 规范的操作系统指的是 UNIX、Linux、Mac OS X 等。

的IOCP（Input/Output Completion Port，输入输出完成端口）机制，以在不同平台下实现同样的高性能。

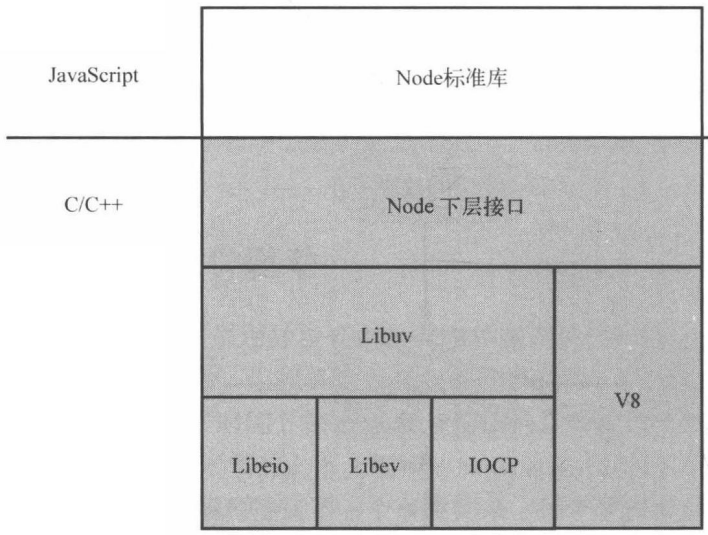

图1-2　Node.js的架构

1.4.2　Node.js与PHP + Nginx

Snoopyxd详细对比了Node.js与PHP+Nginx组合，结果显示在3000并发连接、30秒的测试下，输出“hello world”请求：

- PHP每秒响应请求数为3624，平均每个请求响应时间为0.39秒；
- Node.js每秒响应请求数为7677，平均每个请求响应时间为0.13秒。

而同样的测试，对MySQL查询操作：

- PHP每秒响应请求数为1293，平均每个请求响应时间为0.82秒；
- Node.js每秒响应请求数为2999，平均每个请求响应时间为0.33秒。

关于Node.js的性能优化及生产部署，我们会在第6章详细讨论。

1.5　JavaScript简史

作为Node.js的基础，JavaScript是一个完全为网络而诞生的语言。在今天看来，JavaScript具有其他诸多语言不具备的优势，例如速度快、开销小、容易学习等，但在一开始它却并不是这样。多年以来，JavaScript因为其低效和兼容性差而广受诟病，一直是一个被人嘲笑的“丑小鸭”，它在成熟之前经历了无数困难和坎坷，个中究竟，还要从它的诞生讲起。

1.5.1 Netscape 与 LiveScript

JavaScript 首次出现在1995年，正如现在的 Node.js 一样，当年 JavaScript 的诞生决不是偶然的。在1992年，一个叫 Nombas 的公司开发了“C减减”（C minus minus，Cmm）语言，后来改名为 ScriptEase。ScriptEase 最初的设计是将一种微型脚本语言与一个叫做 Espresso Page 的工具配合，使脚本能够在浏览器中运行，因此 ScriptEase 成为了第一个客户端脚本语言。

网景公司也想独立开发一种与 ScriptEase 相似的客户端脚本语言，Brendan Eich[①]接受了这一任务。起初这个语言的目标是为非专业的开发人员（如网站设计者），提供一个方便的工具。大多数网站设计者没有任何编程背景，因此这个语言应该尽可能简单、易学，最终一个弱类型的动态解释语言 LiveWire 就此诞生。LiveWire 没过多久就改名为 LiveScript 了，直到现在，在一些古老的 Web 页面中还能看到这个名字。

1.5.2 Java 与 Javascript

在JavaScript 诞生之前，Java applet[②]曾经被热炒。之前 Sun 公司一直在不遗余力地推广 Java，宣称 Java applet 将会改变人们浏览网页的方式。然而市场并没有像 Sun 公司预期的那样好，这很大程度上是因为 Java applet 速度慢而且操作不便。网景公司的市场部门抓住了这个机遇，与 Sun 合作完成了 LiveScript 实现，并在网景的Navigator 2.0 发布前，将 LiveScript 更名为 JavaScript。网景公司为了取得 Sun 公司的支持，把 JavaScript 称为 Java applet 和 HTML 的补充工具，目的之一就是为了帮助开发者更好地操纵 Java applet。

Netscape 决不会预料到当年那个市场策略带来的副作用有多大。多年来，到处都有人混淆 Java 和 JavaScript 这两个不相干的语言。两者除了名字相似和历史渊源之外，几乎没有任何关系。现在看来，从论坛到邮件列表，从网站到图书馆，能把 Java 和 JavaScript 区分开的倒是少数[③]。图1-3 是百度知道上的“Java 相关”分类。

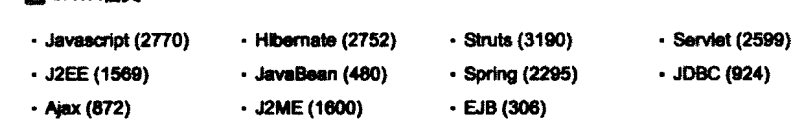

图1-3 百度知道上的“Java 相关”分类

① Brendan Eich 被人称为 JavaScript 之父，他完全没想到自己当年无心设计的一个语言会成为今天最流行的网络脚本语言。

② applet 的意思是“小程序”，它是 Java 的一个客户端组件，需要在“容器”中运行，通常浏览器会充当这个容器。

③ Brendan Eich 为此抱憾不已，他后来在一个名为“JavaScript at Ten Years”（JavaScript 这10年）的演讲稿中写道：“Don't let marketing name your language.”（不要为了营销决定语言名称）。