Web开发经典丛书

hapi.js实战

[美] Matt Harrison 著

梁 宵 郭美青 翟懿博 译

清华大学出版社

北 京

EISBN:978-1633430211
hapi.js in Action
Matt Harrison

图书在版编目(CIP)数据

hapi.js实战 / (美) 马特 • 哈里森(Matt Harrison) 著；梁宵，郭美青，翟懿博 译. —北京：清华大学出版社，2017
(Web开发经典丛书)
书名原文：hapi.js in Action
ISBN 978-7-302-47977-2

Ⅰ. ①h… Ⅱ. ①马… ②梁… ③郭… ④翟… Ⅲ. ①JAVA语言—程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2017)第 209163 号

责任编辑：王 军 韩宏志
封面设计：牛艳敏
版式设计：方加青
责任校对：曹 阳
责任印制：

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：
装 订 者：
经 销：全国新华书店
开 本：185mm×260mm 印 张：22 字 数：521 千字
版 次：2017 年 9 月第 1 版 印 次：2017 年 9 月第 1 次印刷
印 数：1 ～ 3000
定 价：68.00 元

产品编号：-01

译 者 序

当 Ryan Dahl 在 GitHub 上发布第一个 Node 版本的时候，也许未想到这将给 Web 开发 (尤其是 JavaScript 语言) 带来什么。冲出了浏览器沙箱的牢笼，JavaScript 在后端世界大放异彩。全球开发者都张开双臂迎接 Node。Node 的生态系统呈现爆发式扩张，各类工具、框架层出不穷。这使得不借助第三方 Web 服务器和应用服务器构建 Web 应用成为可能，其间涌现了如 Express、Meteor 和 KOA 等优秀的 Node 开发框架。

和众多老牌的 Node 框架相比，hapi 是后起之秀，然而它成功地构建了自己的生态圈。PayPal、Yahoo!、Mozilla 和 Disney 等公司都使用 hapi 构建了自己的 Web 应用，其影响力可见一斑。模块化、以配置为中心和插件化是 hapi 最大的亮点。它的核心仅由少数几个模块构成，其他功能均通过插件扩展来实现，在丰富的功能和无限制的可定制性之间取得了完美平衡。更令人兴奋的是，hapi 经历了沃尔玛在黑色星期五期间史上最大规模的 Node 部署，可以说已经在生产环境中经历了千锤百炼。

本书可能是迄今为止唯一一本介绍 hapi 的入门书籍，由浅入深地全面介绍 hapi 的设计思想与核心功能，并呈现丰富的案例。希望你能从中受益，并享受使用 hapi 带来的快乐，这也是框架名称的由来。

本书由搜狗营销事业部的梁宵、郭美青和翟懿博协作翻译，并由刘建负责最终审校和润色。翻译过程也是学习和总结的过程，我们深感 Node 生态圈的伟大，并为 hap.js 的设计思想所折服。

感谢我们的家人和伙伴，你们无时无刻不在鼓励着我们。感谢清华大学出版社的李阳老师，没有你的信任和悉心指导，我们不可能在短时间内完成本书的翻译。感谢搜狗营销事业部的高清霞、李剑、李晓清对本书翻译工作的大力支持。

在翻译过程中我们努力做到信达雅，并尽力修正了一些原文的小错误，但由于水平有

限，难免存在疏漏之处，敬请广大读者指正。

谨以此译作献给广大 Node 开发者，相信 Node 的明天会更加美好！

译者

序

任何开发框架的核心思想都是共享模式与代码，我们汲取集体的智慧并依赖彼此的成功。框架是由志同道合的开发者们共享的通用基础设施，用于解决相似的问题。它们也是团队协作和高效沟通的核心组成部分。

大约在 Node.js 概念兴起的同时，从一个小工具集开始，逐渐演化出 hapi.js 框架。从大家的经验中，我们学到了更多东西，构建了更大规模的产品系统，扩充了工程师团队，增加了协作复杂性，这个框架不断进化来反映这些问题，并在内部把它们消化了。

hapi.js 社区的文档是非常好的参考资料，但是它不会 (也不能) 包含大量的知识点和经验，这些需要从多年的实践中获得。这便是《hapi.js 实战》的切入点——提供大量知识点，这是迄今为止提供给那些少数吃螃蟹的人为数不多的参考资料。

作为 hapi 的核心项目维护者，Matt Harrison 促使内部人员的专业技能和社区领导力有机结合。核心维护者的大部分工作包括与新加入的和有经验的开发者互动，解答疑问和调研问题。正是这些覆盖了成百上千个问题和疑问的经验和工作使得本书弥足珍贵。

进入 hapi.js 之门并非难事。你可以只用几个方法在几小时之内搭建起一个可运行的 Web 应用。然而，hapi.js 是一个基于最佳实践不断演进的框架，使用自有术语。从服务器、连接、插件和路由，到领域、扩展和方案，这个框架能从很多方面使你的生活变得更简单，使你的产品变得更出色。但如果你不知道它们的存在，它们也无能为力了。

不管对于哪门技术，从新手成长为专家都需要时间。Matt 用本书将大量非文档化的零散知识组织成一篇容易学习的记叙文，新手和专家都会从中受益。

当开始 (或继续) 体验 hapi.js 框架时，你将加入一个忙碌而庞大的应用开发者社区，他们共同致力于开发一流的工具和模式。这些工具已经支持一系列引人瞩目的产品和服务。从支付服务、在线零售店到云和音乐服务，如今你很可能已经在使用至少一个基于 hapi.js 的应用了。我们或许会在未来使用到你创建的应用。

有本书在手，你已经向成为 hapi.js 的专家迈出了第一步。无论你利用其中的经验去构建创新型产品，创建自己的开源组件去分享给他人，还是作为一名积极的贡献者加入 hapi.js 社区，我都希望你会从框架的使用中找到乐趣并获得力量。它是由数十名开源开发者用爱与奉献精神打造的，期望每天都能让应用开发变得更加美好。

它叫 hapi 是有原因的。

Eran Hammer

Hapi.js 的创建者

自　　序

在我16岁的时候，偷偷地把小伙伴的口头禅录制到闪存里，乐此不疲。我在Photoshop中把花哨的图案切成小块，并在Dreamweaver中用HTML表格整合网站。

我花了很长的时间远离Web，训练自己成为一名建筑师。但我总是对构建网站念念不忘，用自己的代码创造新奇的玩意儿，让它在屏幕上呈现并运行。于是在毕业的时候，我做了一个180度的转变——我决心回归Web。

但情况已经发生了变化！我们有了CSS、Web标准和JavaScript，而PHP是当时炙手可热的服务器技术。我补习了所能学到的一切，包括WordPress、Joomla、Zend、CodeIgniter以及当时所有的大型框架和内容管理系统。我做了一些相当像样的网站，并找到了一份初级开发者的工作。事实上，我不知道自己真正在做什么。我对正在发生的事情的本质所知甚少。这些框架做了很多不可思议的事情……而且它们太大了，我觉得完全迷失了。

当我发现Node.js时，情况开始改变了。我从编写一些小脚本开始，比如启动一个小型HTTP服务器，为浏览器提供访问硬盘文件的服务。没有花哨的东西，没有重量级框架，只有几行简单的JavaScript代码，和我在浏览器中了解的一样。我知道每行代码在做什么。代码和内部运作之间没有距离，这让我更加满意。

此后，我逐渐理解了每一件事情。我过去使用的那些所有重量级框架只不过是在模糊Web开发简单美丽的初心。我发誓如果不了解框架的运行机制，就永远不会在工作中使用它。

与Node世界中的其他人一样，我很自然地开始使用Express开发Web应用。那时Express对于开发小型项目很好用。然而随着项目规模越来越大，越来越正规，我不断地遇到问题。比如在面临扩充开发者队伍、多团队协同和开发大型应用时，使用Express是件痛苦的事情。拼凑几个自己的依赖就能使用Node开发正规应用，我开始怀疑这种想法

是不是太天真了。也许像 Zend 和 ASP.NET 这样的大型重量级框架才适于做那样的事情，它们带有大量的类和抽象，那才是复杂项目所需要的吗？

大约在那个时候我找到了 hapi。这些在沃尔玛的家伙们竟敢在黑色星期五用一个 Node.js 应用去支撑世界最大零售商的移动流量。他们是一个庞大的团队，拥有大量用户，并且他们的系统很复杂。看来他们在使用 Express 时遇到了和我一样的挫折，并开发了自己的框架。最令人振奋的是，他们把它开源了。也许可以使用 Node 开发企业级应用了。我必须试一试！

切换到 hapi 对我来说是快速而无痛的。我喜欢简单的配置驱动的 API 和强大的插件系统。我的代码比以前整洁多了。还有，它即快速又安全。我决定一直使用 hapi 了。我开始在它的 GitHub 项目上贡献代码，改善文档，并撰写博客为 hapi 布道。

我发现社区成员相当热心，乐于助人，响应问题和修复 bug 都很快。这个项目一直处于一个非常健康的状态。只发布测试覆盖率为 100% 的代码，这条原则显示了他们对质量的承诺。

当 Manning 出版社联系我撰写一本关于 hapi 的书籍时，我对前景有一些担忧，但是我知道我会答应的。hapi 需要一本书，而我责无旁贷。很荣幸能与你分享我从 hapi 中学到的一切。在经历了差不多 18 个月的写作后，我比以前更了解这个框架了，而我的观点一如既往。我相信它在丰富的功能和无限制的可定制性之间取得了完美平衡，但依然能够简单地启动和运行，而且不妨碍你去开发复杂的应用。

致　　谢

说来惭愧，只有我的名字出现在本书的封面上。事实上，这是很多人士共同工作和努力的结晶。我一个人无法完全胜任这项任务。每位作者在致谢中都会这样说，这并不是出于谦卑，这是事实。

首先，我要感谢开发编辑 Susanna Kline，我与她在 Skype 上交流了很久。我们无所不谈，从天气到航班，再到牛仔 (她来自得州)，偶尔谈谈本书。Susanna 有无限的能力，没有她的好建议和见解，这会是一本没那么厚的完全不同的书。同时感谢 Cynthia Kane 在第 1 章中作为编辑对我的帮助。

非常感谢 Michael Stephens，他是我在 Manning 出版社的第一位联系人。感谢你 Mike，在我们第一次聊天时对我的鼓励，并相信我能胜任这个项目。起初你给我的信心支撑着我走到了最后。

感谢 Corbin Collins，他的建议非常宝贵，保证了本书具有很高的质量和准确性。感谢 Nickie Buckner 的火眼金睛和对原稿的技术审查，他在文本中挑出了非常细小的错误。同样非常感谢 Elizabeth Martin，他在出版的最后阶段收集了很多问题。下面的审稿人慷慨地付出了他们的时间并改进了本书：Davide Fiorentino lo Regio、Earl Bingham、Gavin Whyte、Gonzalo Huerta-Canepa、Jeff Smith、Jeroen Benckhuijsen、Jerry Tan、Jonathan Whittington、Margriet 和 Nikander Bruggeman、Matt Hernandez、Nick McGinness、Philippe Charrière、Ryan Pulling、Stephen Byrne 以及 Thomas Peklak。

同时感谢所有 hapi 贡献者在最开始阶段创建了 hapi，使用它工作和编程是一件非常愉悦的事情。没有你们卓越的工作，就不会有本书的问世。特别感谢 Eran Hammer 创建了 hapi 项目，以及合作阅读本书的原稿，并为我们撰写了精彩的序言。

感谢我的女友 Yi Ching，一直支持我并相信我的能力，即使我没那么自信。

感谢我的父母 Gail 和 David，在我寻找自我的道路上赐予我关爱、鼓励和资源，即使在看上去无路可走的时候。

前　言

《hapi.js 实战》是为完全的新手准备的入门书籍。开篇以简单实用的示例使你逐步适应，然后在理论知识的基础上，深度覆盖 hapi 中所有的重要功能、各种各样的插件和庞大生态系统中的模块。阅读本书后，你将可以得心应手地构建可维护的、快速的、安全的生产环境 Web 应用。

本书面向的读者

任何对构建网站、API、单页面服务器或用 JavaScript 编写的联网 HTTP 服务感兴趣的人应该阅读本书。无论你是否在职业生涯中使用过 Node.js，还是刚刚开始，本书都将给你呈现一个 360 度视角的 hapi 世界。

即使你熟悉 hapi，毫无疑问在本书中还会找到新的有用资料。有经验的 hapi 开发者可以参考本书来了解如何使用更深奥的功能，这些例子将对你有所帮助。

本书的组织结构

本书被分为 3 个部分，共 11 章。

第 I 部分“入门”不会磨磨蹭蹭。它会很快带你编写代码。

- 第 1 章“hapi 简介”便是如此。你会了解到 hapi 对什么有益，以及一些用于创建应用的关键组成部分。
- 第 2 章“构建 API”收集一些需求，并帮助你构建一个功能完整的 hapi 应用。
- 第 3 章“构建网站”研究一些更加面向前端的 hapi 功能，比如静态内容服务。

第 II 部分“扩展工具箱”让你在第 I 部分的基础知识之上，对关键组成部分有更深入的理解。我还混合了一些内容，教你一些额外的超能力，比如验证和缓存。

- 第 4 章“深入理解路由和 handler”深入探讨 hapi 应用中两个通用的核心组件。
- 第 5 章“理解请求和响应”研究一个请求的生命周期以及如何修改它。

- 第 6 章“使用 Joi 验证”教你如何使用具有表现力的强大 Joi 库来锁定 API，对抗恶意的数据输入。
- 第 7 章“使用插件构建模块化应用”展示如何扩展 hapi 以及如何把应用拆分成小的可维护的插件包。
- 第 8 章“充分利用缓存”教你如何利用浏览器和服务器端的缓存来增加应用负载。

第 III 部分“创建健壮的应用”讲述如何确保你的应用是安全的、经过严格测试的、远离 bug 的。

- 第 9 章“身份验证和安全”研究了验证用户身份的多种方式和一些通用的安全漏洞防范技术。
- 第 10 章“使用 Lab、Code 和 server.inject() 进行测试”教你书写简单强大的测试去探查应用的每个角落。
- 第 11 章“投入生产环境及更多相关内容”帮助将你的应用投入生产环境，并提供一些出错时的建议和技术。

有两个附录。附录 A 提供了补充信息，包括下载安装 Node 和 npm。附录 B 介绍了版本号，包含了对本书用到的包的说明。

关于代码

本书包含了很多源代码示例，有带编号的代码清单，也有行内的普通文本。在两种例子中，源代码用等宽字体格式化以区别于普通文字。有时代码为粗体，用来高亮显示特别重要或与周边讨论有关的代码。

多数情况下，源代码被重新格式化了；我们加入了折行和修正的缩进以适应书中可用的页面空间。在极少数情况下，这样还不够，代码清单包含了续行符 ➥。此外，当代码以文本描述时，源代码中的注释通常会从代码清单中移除。

所有例子的源代码和本书中的代码清单都可以在 GitHub(https://github.com/mtharrison/hapi.js-in-action) 上找到。代码被层次化地组织起来，以匹配书中的章节标题。这样设计是为了在你学习任何章节时都能方便地查找代码。

另外，可以访问 www.tupwk.com.cn/downpage，输入本书的书名或中文 ISBN 下载代码；也可直接扫描封底的二维码下载。

本书和 GitHub 上的代码样例仅工作在 Node.js 4.0.0 以上版本，因为用到了一些 ES2015 的属性，比如 let、const 和箭头函数。

作者在线

购买《hapi.js 实战》时，能够免费访问由 Manning 出版社运营的私有论坛，在这里你可以对本书进行评论，提出技术问题，并获得作者和其他用户的帮助。要访问和订阅这个论坛，请去 https://www.manning.com/books/hapi-js-in-action。这个页面提供了注册后如何进入论坛的信息，指出什么样的帮助有用，介绍论坛的行为规范。

Manning 出版社对读者的承诺是提供一个地点，在这里各位读者之间，读者与作者之间可以碰撞出思想的火花。我们不承诺作者现身的次数，他们对该书论坛的贡献是自愿的 (并且是无酬劳的)。我们建议你尽量问他一些具有挑战性的问题，激发他的兴趣。

只要本书还在销售中，作者在线论坛和前面讨论的存档就会在出版社的网站上开放。

其他在线资源

- 如果发现任何关于 hapi 使用方法或整体项目的问题，可在讨论区仓库 https://github.com/hapijs/discuss 提交 issues。
- 在 Stack Overflow 上也有 hapi.js 的标签 (http://stackoverflow.com/questions/tagged/hapijs)。

作者简介

Matt Harrison 是一位自由职业的 Web 开发者和顾问。他是 hapi.js 的核心贡献者，高产的博客作者，也是一位活跃的 Node.js 社区成员。此前，他曾是一名建筑师。他喜欢吃拉面，喝吉尼斯黑啤。

关于封面插图

《hapi.js 实战》的封面人物是 Sauvage de la Nouvelle Zeelande(一位新西兰原住民)。这幅插画取自一本名为 *Costumes de Différents Pays* 的多国服饰画集，由 Jacques Grasset de Saint-Sauveur(1757–1810) 创作，1797 年在法国出版。每幅插画都经过手工精绘并着色。

Grasset de Saint-Sauveur 丰富生动的画集让我们穿越时空，看到 200 年前文化色彩浓郁的世界各地的城镇和区域。那时的人们彼此隔离，说着不同的方言和语言。在街道或乡村，仅从他们的服饰，就能轻易地分辨出他们住在哪里以及他们在生活中的职业或身份。

自那时起，服饰习俗已经改变了，当时如此丰富多彩的地区多样性已经逐渐消失。如今通常很难区分来自不同大陆的居民。也许，尝试从乐观的角度来看，文化和视觉上的多姿多彩已经蜕变成更加多样化的个人生活——或者说更丰富以及有趣的现代科技生活。曾几何时，很难区分出计算机图书之间的差别，Manning 出版社通过封面推动了计算机出版行业的创新，这些封面来自两个世纪前特色鲜明的地区性生活，它们通过 Grasset de Saint-Sauveur 的画作复活了。

目　录

第 I 部分　入门

第 I 部分

入　门

《hapi.js 实战》的第 I 部分将为后面的章节打下基础，同时介绍 hapi 中的很多核心概念。

第 1 章介绍 hapi，说明了它是什么以及能做什么。你会从宏观上了解看到应用如何被整合在一起。

在第 2 章中，你要全情投入，并使用 hapi 构建一个 JSON API。你将了解如何把业务需求转化为可运行的代码，并将接触到很多关键概念，包括路由、handler、身份验证和数据库操作。

第 3 章专注于前端，为在第 2 章中构建的 API 搭建一个简单的网站。你将全面学习静态文件服务以及如何使用模板语言去组织布局、片段和视图中的标记。

第 1 章

hapi 简介

1

本章内容：

- hapi.js 是什么
- hapi.js 解决的问题
- hapi.js 背后的理念
- hapi.js 的架构

不论使用什么语言或平台，如果你构建过 Web 应用就会清楚，有时一个简单的版本会迅速变得复杂起来。在构建一个 Web 应用时，有很多事情需要考虑，包括如何设计缓存、验证 (Validating) 和身份验证 (authenticating)。还需要确定如何用合理方式组织代码和目录的结构。

哪怕对于有经验的开发者，最开始的决策往往是困难的，并且开弓没有回头箭。这就是我们更愿意使用框架而不是每次都从零开始的原因之一。

使用 Node.js 构建 Web 应用的框架有很多。像 Express.js 这样短小精悍的框架，为了达到灵活性目标，基本不能拿来就用，仍需要做很多基础配置。而其他只关注 API 的框架 (比如 Restify) 则会在应用范围上受到限制。

> **术语** 我有时写 hapi.js，有时只写 hapi。一般来讲，使用 hapi.js 时，我是把项目、生态系统以及社区作为一个整体。而使用 hapi 时，是指核心的 hapi 包。hapi 总以小写字母 h 开头。交替地使用 Node.js 和 Node 也是这个含义。同样，我可能使用 Express.js 或 Express，这是指另一个流行的 Node 框架。

hapi 能让你鱼与熊掌兼得。以配置为核心的开发方式意味着它很灵活，能随需求的变化而调整。但如果只想让业务逻辑轻松地启动并运行起来，使用此框架是一个明智的选择。

hapi 的插件系统和丰富的模块生态系统意味着 hapi 核心团队已经开发的模块通常能够解决你的问题，你可以放心，它是安全的，经过了广泛测试。

1.1 hapi 是什么

hapi 是由沃尔玛实验室移动 Web 团队开发，是使用 Node 构建 Web 应用的开源框架。如果你用过沃尔玛的移动应用或在手机上浏览过沃尔玛的网站，那么你已经在使用 hapi 了，而且很可能并不知道它的存在。

hapi 最常见的应用场景是作为 JSON API 构建 Web 服务，但它也可用来构建 HTTP 代理，或作为单页面应用或网站的服务器组件。图 1.1 展示了这些应用的例子。如果需要构建使用 HTTP 协议的软件，hapi 会极大地简化你的工作。

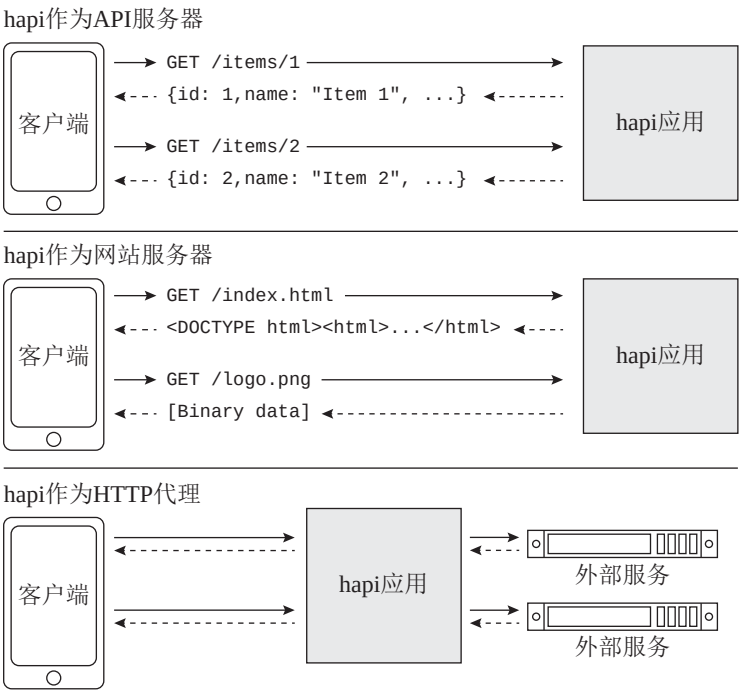
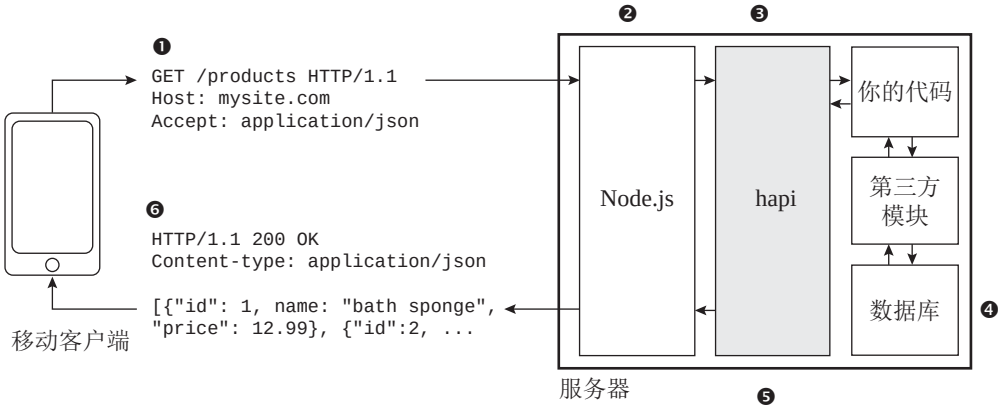


图 1.1 hapi 可用于构建各种类型的网络应用

Node 已经提供了使用内置 http 模块构建 Web 服务器的方法，但只使用 http 构建正规应用是一件复杂而困难的事情。幸运的是，这正是 hapi 框架一展拳脚的地方。

为理解 hapi 用在哪里更适合，请看图 1.2。可以看到，hapi 介于 Node 和应用代码之间，提供操作 HTTP 请求和响应的抽象层。



❶ 移动应用发起 HTTP 请求，获取产品数据。

❷ 请求被 Node 接收并传递给 hapi。

❸ hapi 验证用户身份并把请求路由到代码中相应的函数。

❹ 应用从数据库中获取产品数据。

❺ 产品数据被传递给 hapi 的 reply() 函数。hapi 进行验证，如果配置了缓存，则将输出缓存下来。

❻ HTTP 响应由 Node 发送到移动应用。

图 1.2　hapi 在一个应用实例中所处的位置

如果你从事过 Node 开发，在使用 hapi 相似的功能时会感觉很亲切。Express.js、Restify、Director、Loopback 和 Sails 都是此类用于构建 Web 应用的框架。

本章将介绍 hapi 与其他框架的区别，并讨论 hapi 的主要组成部分；成功地运用 hapi 需要了解这些内容。同时也思考 hapi 适用于哪类应用，不适用于哪类应用。然后介绍寻求帮助的最佳去处。

Node.js

Node 是一个使用 JavaScript 开发应用的平台，可运行在服务端。它由 Google Chrome 浏览器中极快的 V8 JavaScript 引擎提供支持。Node 在 V8 的基础上提供了一组 API，它们是你在开发服务端应用时梦寐以求的，过去在其他语言中才能找到。这些 API 可用于操作 TCP、HTTP、文件系统等。它们通过 CommonJS 模块系统作为内置的 JavaScript 模块对外公开。

由于 Node 中的 http 模块层次很低，每次从头开始构建 Web 应用是一项繁重的工作，并且需要编写很多样板代码。这便是 hapi 这类框架的用武之地，它们提供了更高级别、更便捷的 API，适于构建多种应用。

如果你有不同的服务端编程背景，比如 PHP 或 Ruby，可能不会这样直接在应用中处理 HTTP 请求。你可能更熟悉把 Apache 或 NGINX 作为一个 Web 服务器。在 Node 世界里情况完全不同——Web 服务器实际上是应用的一部分，如图 1.3 高亮显示的部分。

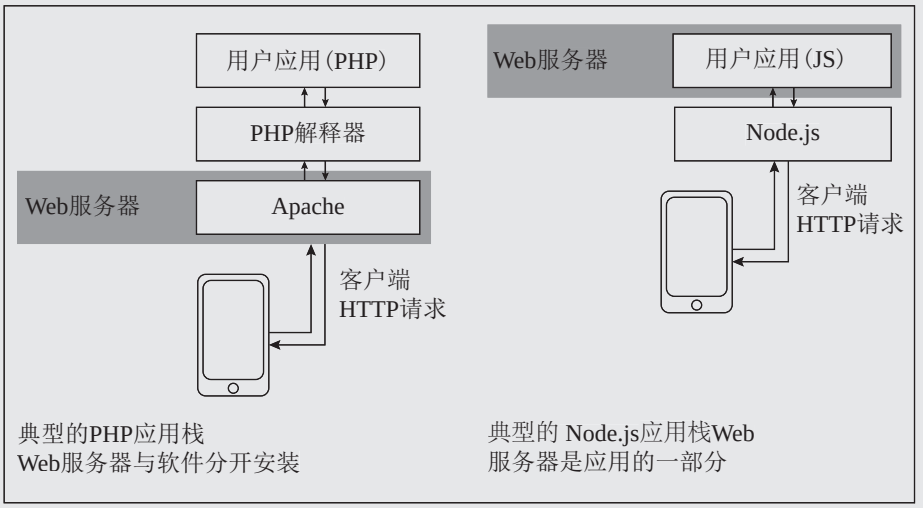


图 1.3 使用 Node，Web 服务器是应用开发者的职责

1.1.1 hapi 的特色

既然有那么多功能相似的框架，为什么还要选择 hapi 呢？hapi 的开发团队从其他现存框架中吸取了经验和教训，沉淀出了一套终极方案。作为负责全球最大零售商沃尔玛的移动网站基础设施的开发者，他们可能有些吹毛求疵，但他们集体的努力为我们奉献了一款可以从中受益匪浅的框架。

随着越来越多的人士在移动设备上购物，沃尔玛的移动流量和销售额逐年倍增，这不断地给工程师们提出了新挑战。由高级架构师 Eran Hammer 领导的沃尔玛实验室的移动 Web 团队选择 Node 作为构建未来基础设施的平台。这个决定基于 Node 高效可扩展的模型，工程师对 JavaScript 语言的熟悉程度，以及围绕 Node 的健康且蓬勃发展的社区。

沃尔玛的开发团队在地理上是分散的。这意味着他们需要这样一个框架，它是高度模块化的，并能以一种有效且安全的方式简化大型项目的多人协作。

开发人员在使用 Express 时遇到了问题，特定的中间件顺序和可能冲突的路由意味着一个团队成员的更新可能会对其他人的工作产生不利影响。团队在 hapi 中解决了这个问题，他们发明了插件架构 (在第 9 章中深入讨论) 和一个确定的路由算法，这为每位开发者的工作创建了一个更加隔离和安全的环境。

在 2013 年的黑色星期五，hapi.js 承担了 100% 的 Walmart.com 的移动流量。那可能是 Node 迄今为止最大规模的部署行动之一，它取得了巨大成功，这让其他许多企业对 Node 和 hapi 构建企业级 Web 服务的潜力刮目相看。

#nodebf

在 2013 年的黑色星期五当天，沃尔玛实验室团队在 Twitter 上以 #nodebf 为题和世界分享了在生产环境中使用 Node 的经验。他们贴出了示例代码，展示基础设施的零部件如何被整合在一起。其中一些推文展示了全天的内存和 CPU 消耗量平稳的折线图，如图 1.4 所示。这些推文引起了人们的关注，他们想知道 Node 为生产环境中的企业级应用到底做好了哪些准备。

图 1.4　#nodebf 话题中平稳的内存消耗图

1. 它是 Node

Node 是构建 API 的极佳选择。JSON 已经成为了事实上的网页数据传输编码标准。在 JavaScript 中使用 JSON 是很自然的选择。Node 运行时的底层实现细节让你不需要购买昂贵的硬件就能让 API 支持数以千计的并发用户。

2. 注重模块化有益于团队协作

对于我和很多其他用户来讲，模块化可能是 hapi 最大的优势。插件系统让你参与到像乐高积木一样相互隔离的应用模块中，并把它们作为单独的应用运行。单个模块或插件可以完全独立地开发、测试和分发 (作为 npm 包)，它们可能来自一个大组织中不同的开发者或团队。插件也能让开发者创建的功能与整个开源社区共享。第 7 章会涉及 hapi 插件。

3. 配置驱动的特性

hapi 配置优于代码的理念意味着不需要记住那么多执行常用任务的方法，比如管理文件的上传。相反，复杂的行为被封装到配置驱动的 API 中。你不需要从头开始学习所有的配置项，因为框架总会选择合理的默认配置。

4. 快速上手

hapi 的目标之一就是减少编写 Node 应用所需的样板代码的数量。使用 hapi 几乎可以马上开始编写应用的业务逻辑。也就是说你可以快速尝试 hapi 并决定它是否适用。

5. 它是开源的

当使用一个开源框架时，它的代码对全世界是开放的，也可以向其贡献代码。如果功能存在缺失，可以提交一个功能申请，甚至自行发起一个添加此功能的 pull request。如果在使用某个特别功能时卡壳了，可以自己去分析代码或在 GitHub 上提交一个 issue。

6. 在生产环境中千锤百炼

沃尔玛使用 hapi.js 运行其所有的移动网站和很多其他服务。没有比这需求更高的场景了，

hapi 证明了它的价值。其他使用 hapi 的大公司还包括 PayPal、Yahoo!、Mozilla 和 Disney。

1.1.2 hapi 是哪类框架

有一些大一统的框架，迫使你去采用可预测且一致性的方式做事，比如总要使用固定的格式命名数据库表。大一统的框架减少了你做决定的次数，它们预先做好了决定并提供有用的行为规范，并不需要写代码。使用大一统的框架，你通常会为了获取便捷性而牺牲灵活性。

大一统框架的一个明显例子是 Ruby on Rails，它做了一些关于应用架构方式和所用数据格式的假设。违背这些假设并按自己的方式去做是很困难的，到最后会发现这几乎是不可能的。

大一统的框架通常是铁板一块，在一个库或包中提供了许多功能。当使用这样一个庞大的框架时，几乎不需要外部软件，它假定你做任何事情都会按照框架规定的方式去做。

另一个极端是微框架，它们做的事情很少而且几乎不为你做决定，它们是非常小的程序包，围绕平台的原生能力提供通用任务的便捷 API。此类框架的典范是 Express.js，一个 Node Web 框架。Express 不提供代码组织、输入验证或数据库操作的解决方案。所有这些事情在开发时都可能遇到，但需要你自行决定如何去做或者使用其他软件的接口。

hapi 在提供丰富功能和保持简洁之间取得了很好的平衡。hapi 的核心库仅提供创建现代 Web 应用时所需的功能。任何附加功能，比如身份验证，都是可用的，但会作为 npm 上的一个插件模块提供。hapi 应用通过插件系统由这些模块，连同你的代码拼合而成。图 1.5 对比了这些方案。

hapi 当然有它的主张，但是你也可以有自己的。当不想以 hapi 提供的方式做事时，只需要加载一个以你想要的方式工作的插件就可以了——或者干脆自己写一个。

1. 模块化的，基于插件的架构

使用 hapi 构建最简单的应用通常也要涉及模块的管理。在 npm 上往往有不止一个可以完成给定任务的流行模块。hapi 团队简化了这个过程，他们发布了与 hapi 无缝集成的一整套模块。

就像通过添加日志功能来扩展框架一样，hapi 应用中的插件用来从逻辑上划分应用的业务。假设我们要为一个电子商务网站构建后台的 API。可能需要一些不同逻辑的组件来完成此事。我们可能有一个用于支付的组件，一方面用来做相关商品推荐 (类似亚马逊的捆绑销售功能)，另一方面用来做分析。使用 hapi，可以把每一个功能做成独立的插件，并在服务器中按需加载。这是一个强大的方案。如果一个特定的 API 需要扩展，只需要在更多的服务器中加载那个插件，而不需要复制整个应用。这个模块化方案也意味着，随着应用的增长，每个团队可以专注于各自的组件并独立工作。

图 1.6 展示了使用 hapi 构建一个应用可能涉及的所有组件。如果一些术语还不熟悉，此刻不必担心。后续章节会详细地解释它们。现在重要的是认识到，hapi 应用一般是由许

多不同的模块组合而成的。

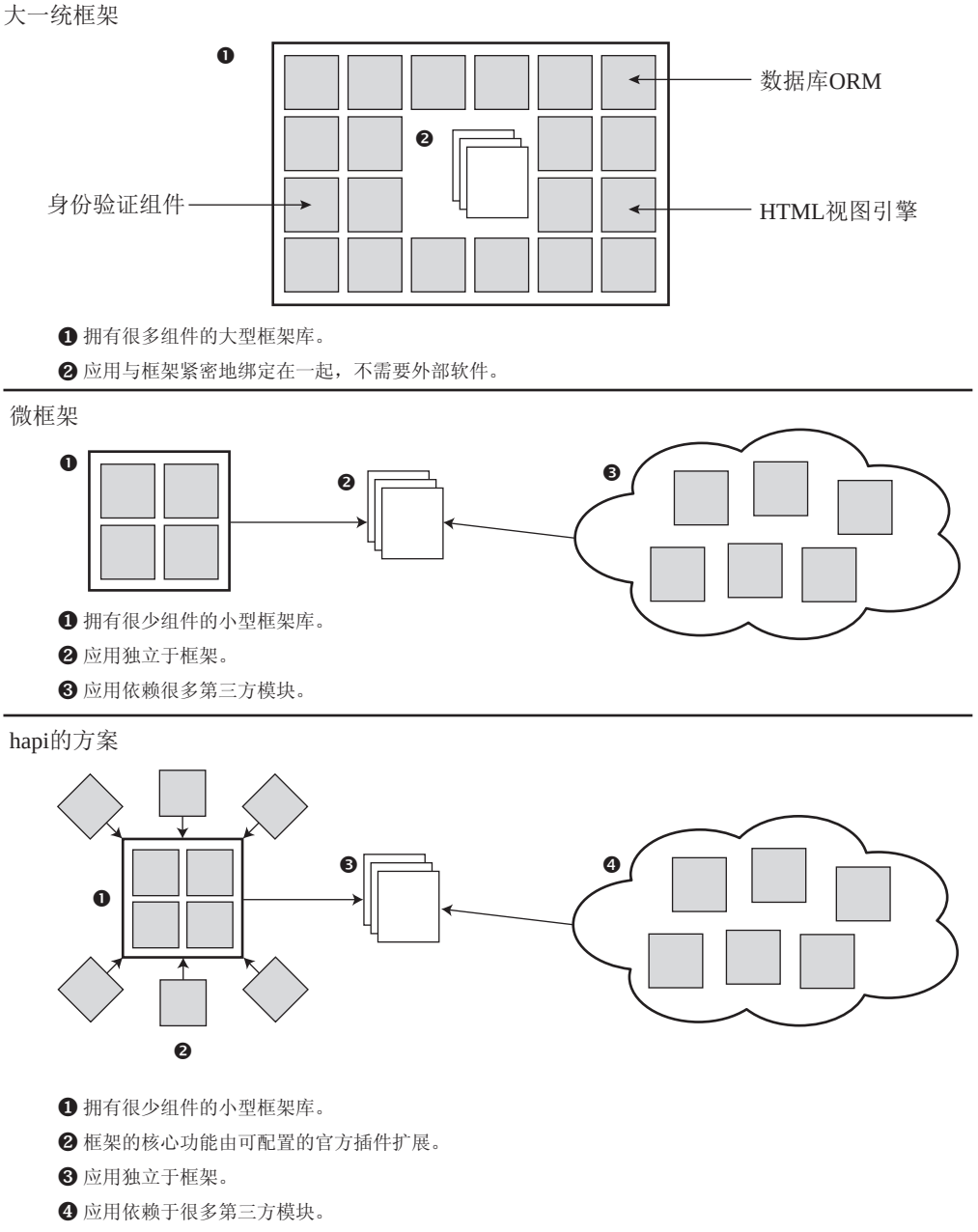


图 1.5　对比不同的框架类型

2. 以配置为核心

hapi 框架的理念是写配置优于写代码。因此，hapi 中的许多功能的实现依赖于一个小型方法集上丰富的配置项 (而不是通过调用方法)。也就是说，与其他框架相比，你可以用更少的代码表达想法。而更少的代码通常意味着一个更易于维护和理解的应用。

但为什么配置要比代码好呢？为说明以配置为核心和以代码为核心的 API 之间的差异，假设我们要编写一个用于糖果自动生产线的 JavaScript 库。一个以代码为核心或提供大量方法的例子可能会像代码清单 1.1 一样。

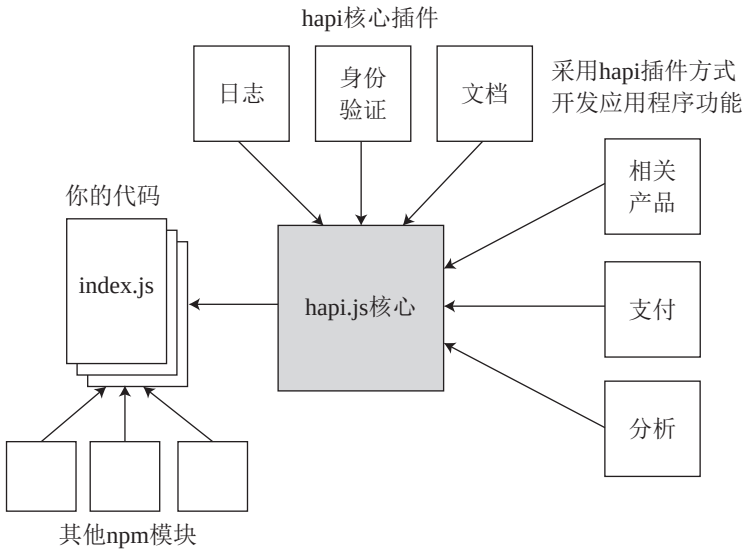


图 1.6 一个电子商务 API 如何被构造为 hapi 应用

代码清单1.1 提供大量方法的jellybean库

```
const Jellybean = require('jellybean');

const bean = new Jellybean.Bean();

bean.setName('Coffee');
bean.setColor('brown');
bean.setSpeckles(false);
bean.setHumanReaction('yuck!');
```

前面的代码非常冗长——为创建 jellybean 调用了 bean 对象的 5 个方法。代码清单 1.2 展示了一个更面向配置的 API。

代码清单1.2 提供大量配置项的jellybean库

```
const Jellybean = require('jellybean');

const options = {
    name: 'Tutti Frutti',
    color: 'mixed',
    speckles: true,
    humanReaction: 'yum!'
};

const bean = new Jellybean.Bean(options);
```

这个例子在创建 jellybean 对象时不需要调用任何方法。取而代之的是构造函数接受一

个包含了所有选项的配置对象。第二个例子更灵活，因为它把配置和代码分离了。

可以把 jellybean 的所有配置放入一个单独的文件中并包括它们。如果以后想更改配置，则不必更改代码——只要更新配置就可以了。

hapi 中的许多方法都在此原则上运行。这个方法将有助于编写整洁的代码和更便于维护的应用。

接下来，我将介绍 hapi 的主要组成部分，开始构建一个应用时需要了解这些内容。

1.2　hapi 的组成部分

hapi 是功能完备的框架，其中的某些部分你可能永远不会用到。但是一些关键组件是每一个 hapi 应用的基础。本节将介绍服务器、连接、路由、handler 和插件，如图 1.7 所示。

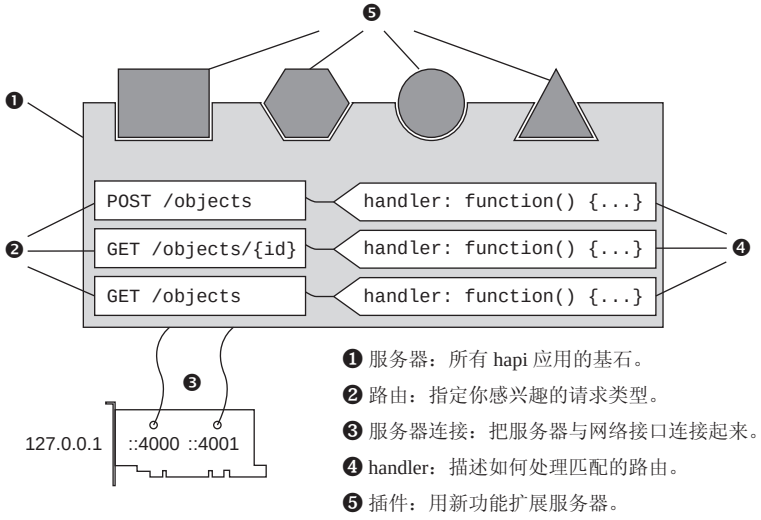


图 1.7　hapi 的组成部分

应用接收到的每一个请求将经历一个生命周期，在此期间会与这些元素发生交互，然后生成一个返回给客户端的响应。图 1.8 展示了这个过程的基本概况。一旦掌握了这些核心组成部分的概念，无论开发什么应用，都能理解它的全部处理过程。

现在开始查看代码清单 1.3。如果想让这些代码运行起来，你需要确认已经安装了 Node 和 npm。如果需要帮助，可参阅附录 A。

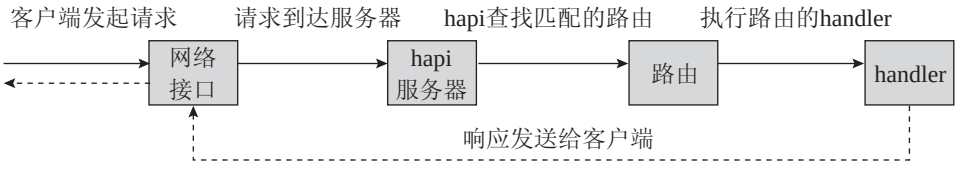


图 1.8　组成部分如何整合在一起

现在安装 hapi。运行 npm install hapi@13 将 hapi 安装到你的主目录下。这不是一个理想方式，但可以快速开始。很快你将学习如何在本地项目中更好地管理依赖。

代码清单1.3 一个使用了所有组成部分的简单hapi应用

```
const Hapi = require('hapi');

const server = new Hapi.Server();
server.connection({ port: 4000 });

server.route({
    method: 'GET',
    path: '/en',
    handler: function (request, reply) {

        reply('Hello!');
    }
});

const plugin = function (server, options, next) {

    server.route({
        method: 'GET',
        path: '/cn',
        handler: function (request, reply) {

            reply('你好!');
        }
    });

    next();
};

plugin.attributes = {
    name: 'My plugin'
};

server.register(plugin, (err) => {

    if (err) {
        throw err;
    }

    server.start((err) => {

        if (err) {
            throw err;
        }
        console.log('Server running at:', server.info.uri);
    });
});
```

创建一个服务器

在服务器的 4000 端口上创建一个连接

创建一个路由，响应到达 /en 的 GET 请求

为路由定义一个 handler

定义一个插件，在内部创建另一个路由

把插件加载到服务器中

启动服务器

上述代码是一个使用了所有组成部分的简单 hapi 应用示例。在本节结束之前，即使不能理解所有代码，也应该理解每个组成部分的职责。

注意

本书假定你运行 Node 4.0.0 以上的版本。本书的 hapi 版本依赖于 ES2015 的一些特性，如 let、const 和箭头函数，这些在早期的版本中不可用。

1.2.1　服务器

一切始于服务器，它是 hapi 应用的容器。本节列出的其他对象都被创建或用在服务器环境中。hapi 服务器不直接监听网络端口，而通过创建服务器连接来对外通信。

1.2.2　连接

为接收传入的请求，要使用连接来绑定 hapi 服务器和网络接口。通过使用连接，可使一个服务器监听多个端口，这对使用 TLS 的应用很有帮助。

1.2.3　路由

通过路由可告诉框架你对哪些类型的请求感兴趣。用一组选项创建路由，包括 HTTP 动词 (如 GET、POST 和 PATCH) 和路径 (如 /about)，然后把它加入服务器中。

当一个新请求到达服务器时，hapi 会尝试查找一个与此请求匹配的路由。如果其中一个路由与请求配对成功，就会执行路由的 handler。

1.2.4　handler

每当创建一个路由时，就需要指定一个 handler。handler 告诉 hapi 如何响应一个 HTTP 请求。一个 handler 可有多种形式。最灵活的 handler 就是定义一个 JavaScript 函数，它可以访问 request 对象和 reply 接口。通过 request 对象可以获取请求的详情。然后使用 reply 接口对请求做出任意响应。例如，可为一个 API 请求返回一个图片文件或一个 JSON 响应。

使用 hapi 插件，可添加预编程的新 handler 类型，使用它们不需要编写任何代码。例如 Inert 插件中的目录 handler：

npm package `inert v3`(http://npmjs.com/package/inert)

使用目录 handler，不需要编写任何代码就可以在一个目录中提供静态文件服务。你会在第 3 章中看到它是如何运作的。

1.2.5　插件

插件是为服务器扩展新功能的一种方式。它们可为服务器扩展一些全局的实用工具，

例如记录所有请求或为响应添加缓存。许多插件作为 npm 包发布，它们能处理像身份验证、日志记录等事项，由 hapi 的核心团队或社区编写。

可通过创建自己的插件把应用划分成较小的逻辑单元，这样更便于维护或在日后全部替换或移除。第 6 章将专门探讨如何以这种方式使用插件。

1.3 何时应该 (不该) 使用 hapi

hapi 是一个灵活的框架，适于构建各种类型的应用，尽管某些应用类型比其他更合适。本节讨论适于使用 hapi 的更通用场景，当然也有一些场景对你来说 hapi 或许不是最佳选择。

1.3.1 何时应该使用 hapi

尽管已经讨论过 API，但 hapi 不限于此场景。它支持更通用的 Web 应用框架所提供的功能，比如 HTML 模板渲染和静态文件服务。

它也具备一些更通用的网络功能，比如 HTTP 请求代理，这进一步扩展了它的能力。本节将关注 hapi 适用的一些应用类型。

1. JSON API

JSON API 是 hapi 适用的经典场景。hapi 的扩展功能集——包括路由、输入和输出验证、身份验证、缓存和自动生成文档——令使用 hapi 构建 API 变成一种令人愉悦的体验。可使用 hapi 构建为所有类型客户端服务的 API，如移动应用、单页面应用或其他服务。

2. 静态或数据库驱动的网站

hapi 可与一些 HTML 模板语言结合使用，比如 Handlebars 和 Jade，可根据动态数据轻松地创建 HTML 文档。在 npm 上可找到能与几乎任何数据库对接的包。有些应用使用基于 Cookie 的身份验证和会话来维持请求间状态，hapi 也使部署这类应用变得很简单。

3. 单页面应用

可使用 hapi 结合 Backbone、React 和 AngularJS 这类前端框架去创建令人印象深刻的单页面应用 (SPA)。SPA 通常由模块化的页面组件构造。使用 hapi，可简单地把 HTML 片段或 JSON 作为 AJAX 请求的响应。

4. 代理

hapi 也是构建代理和反向代理的不错选择。一个名为 h2o2 的 hapi 插件提供了一个 handler 类型，用于把 HTTP 请求代理到其他端点。沃尔玛用这种方式把来自移动端 API 的请求转发到大量遗留的 Java 应用。使用这个方法，开发人员可透明地把将要下线的服务替换掉。

1.3.2　何时不应该使用 hapi

hapi 特别适于构建 HTTP 应用，因为它是在 Node 基础上构建的，它在 I/O 处理和网络操作方面非常快。但也继承了 Node 不适用的应用场景。因为 Node 应用中的 JavaScript 代码运行在一个单线程中，不适用于 CPU 密集型应用，所以使用 hapi 开发视频解码、AI 软件或 3D 渲染这类应用是不合理的。开发这类应用无疑需要更合适的环境，比如 C++，而可以用 hapi 或 Node.js 的原生模块作为这些系统的 Web 前端。

尽管最初被设计为只响应标准 HTTP 请求，nes 插件通过在路由系统之上建立 websocket 通信层扩展了 hapi 的实时能力。这是为软实时应用设计的，在这里时间不是绝对的关键因素。所以完全适合像聊天、实时订阅和社交这些应用。不推荐使用 hapi 或 Node 开发航空电子、核能、高频金融交易系统这类硬实时应用。

1.4　hapi 的运作方式

现在已经介绍了一些 hapi 的原理和背景，让我们来看看它是如何运作的。构建任何 hapi.js 应用的第一步是创建一个服务器并建立一个到该服务器的连接。

1.4.1　安装 hapi

在电脑上的任意位置创建一个用于存储本示例的目录。这里在我的主目录 (~) 下创建一个名为 hapi-example 的目录：

```
mkdir ~/hapi-example && cd ~/hapi-example
```

然后在项目目录下创建一个 package.json 文件。npm 使用这个文件来管理项目的依赖。运行下面的命令，npm 可以创建这个文件：

```
npm init
```

你可以接受 npm 建议的所有默认选项，一旦完成，npm 会自动创建 package.json 文件。接下来要安装 hapi 了。在本书中，使用的 hapi 版本是 13。在控制台中输入下面的命令来安装：

```
npm install --save hapi@13
```

这步操作会在当前目录下创建一个 node_modules 文件夹并将 hapi 下载至此。也会更新 package.json 文件，把 hapi 作为项目的依赖。

npm Package `hapi v13` (http://npmjs.com/package/hapi)

最后一步是创建一个 JavaScript 文件用来存放这个例子的代码。为方便起见，将其命名为 index.js。

如果完成了上面几步，就可以准备开始写代码了。

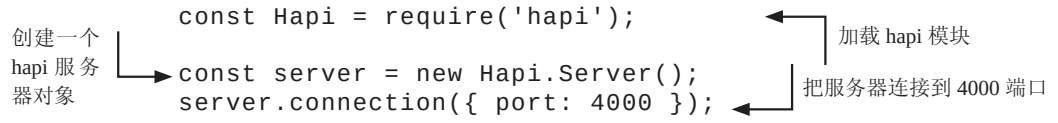
1.4.2 创建服务器

一个服务器就是 hapi 的一个对象，可接收和路由请求。在它可以接受新的请求之前，必须被连接到一个网络套接口。

hapi API `server.connection([options])` (http://hapijs.com/api#server connectionoptions)

代码清单 1.4 创建了一个连接到 4000 端口的服务器。

代码清单1.4 创建一个hapi服务器



为让这个服务器能做点有用的事情，必须添加一个路由。路由匹配一个路径和一个 HTTP 动词，如 GET 或 POST，并输出一些预期的行为。你要通过 handler 来告诉 hapi 预期的路由行为。

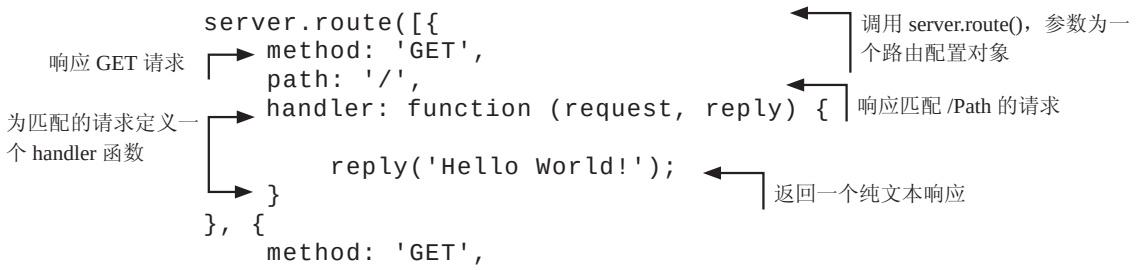
1.4.3 添加路由

代码清单 1.5 通过 server.route() 方法为服务器添加了两个路由。

hapi API `server.route(options)` (http://hapijs.com/api#serverrouteoptions)

第一个路由为路径为 / 的 GET 请求返回纯文本“Hello World!”。第二个路由为路径为 /json 的 GET 请求返回一个 JSON 响应。

代码清单1.5 为服务器添加路由



```
        path: '/json',
        handler: function (request, reply) {

            reply({ hello: 'World' });
        }
    }]);
```

返回一个 JSON 响应

注意，在这个代码清单中路由以不同的数据类型调用了 reply()。

hapi API `reply()` (http://hapijs.com/api#reply-interface)

第一个路由使用字符串调用 reply()，第二个则使用对象。hapi 非常聪明，它知道第一个例子需要一个 text/plain 类型的响应，而第二个例子需要一个 application/json 类型的响应，并会自动设置响应 header 的 Content-Type。这是一个 hapi 会为你做出合理假设的例子。

1.4.4　注册插件

现在假设要把服务器接收到的每个请求的日志详情输出到控制台中。可使用 good 插件来做这件事。需要使用 server.register() 方法注册这个插件。

hapi API `server.register(plugins, [options], [callback])`
(http://hapijs.com/api#serverregisterplugins)

还需要选择一个 reporter 并指定想要记录的事件类型。

npm Package `good v6` (http://npmjs.com/package/good)

good 是一个用于监控和报告事件的 hapi 插件。有一些 good 可用的 reporter，包括 console、HTTP 和 UDP。你也可创建自己的 reporter。

在这个例子中，我会选择 console reporter，即 goodconsole。

npm Package `good-console v5` (http://npmjs.com/package/good-console)

我只想上报 request 事件类型。这个例子会依赖 good 和 good-console 两个 npm 包，必须先安装它们：

```
npm install --save good@6 good-console@5
```

如代码清单 1.6 所示，在服务器开始监听请求前，先要启动它。为此使用 server.start() 方法。

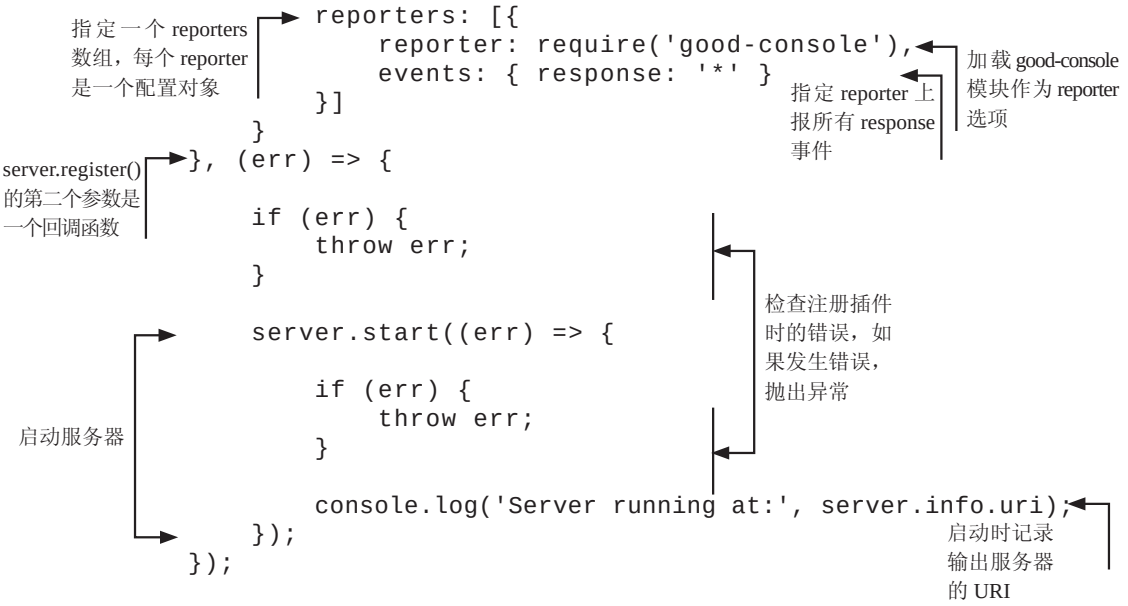
代码清单1.6　注册一个插件并启动服务器

```
server.register({
    register: require('good'),
    options: {
```

加载 good 模块作为 register 选项

server.register() 的第一个参数是插件配置对象

为插件指定 options 对象



下一步，让我们运行这个示例程序看看它做了什么。

1.4.5 运行 hapi

用 Node 程序运行 index.js 脚本来启动这个应用。

```
node index.js
```

打开浏览器并访问 http://localhost:4000/。在浏览器中可以看到一个 hapi 的响应。然后加载页面 http://localhost:4000/json。

每次生成一个请求，在控制台中都可以看到一条日志，其中包含一些关于此请求的信息。

```
    141226/162232.034, [response], http://localhost:4000: get / {}
200 (3ms)
    141226/162235.019, [response], http://localhost:4000: get /json
{} 200 (2ms)
```

按下 Ctrl+C 可以停止服务器。在这个非常基础的例子中，可看到如何去创建一个服务器，添加路由，定义简单的 handler，并通过注册和使用插件来扩展功能。

1.5 获得帮助

本书会尽量写得通俗易懂，但为了保持你的兴趣并确定下一个良好的节奏，不会过分探究 hapi 功能的每个细节。你可能会遇到困难或者在这里找不到问题的答案。这时知道

去哪里寻求帮助很重要。

1.5.1　hapi.js 网站

hapi.js 网站 (http://hapijs.com) 应该是获取 hapi 功能和 API 文档的第一站。它提供了丰富的基础教程，附带完整的 API 文档。如果想知道某个方法如何执行，或者它需要什么参数，建议先看一看这里。

1.5.2　Make Me hapi

hapi 的核心贡献者们开发了一个名为 Make Me hapi 的交互教程。这是一个自我练习的 hapi 教程，完成它会很有趣。图 1.9 展示了它的控制台界面。

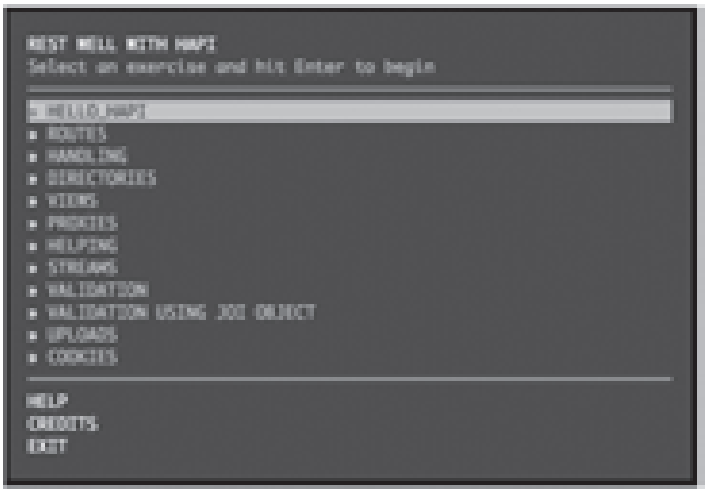


图 1.9　Make Me hapi 的控制台界面

我建议每位读者抽时间完成 Make Me hapi。在控制台中运行 npm install –g makemehapi 来安装。下载完成后，运行 makemehapi 开始练习。

1.5.3　GitHub

如果你发现了 hapi 的问题或 bug，应该在 GitHub 项目 (https://github.com/hapijs/hapi/issues) 上提交一个 issue。对于文档中没有解答的问题，也在 GitHub 提交 issue。

1.5.4　IRC

hapi.js 有一个 IRC 聊天室，在那里你可以直接提问。聊天室在 irc.freenode.net 上，名

为 #hapi。可能你对 IRC 不熟悉，它是一个开发者和软件用户闲逛的聊天室，在这里大家可以讨论框架和帮助他人解决问题。使用 Web 界面是连接到 IRC 聊天室的快速方式。访问链接 http://webchat.freenode.net/?channels=hapi 加入聊天室。

1.5.5 Stack Overflow

如果对问题有了答案，但是想听听其他人的意见，可以在 Stack Overflow 的 hapi.js 标记下提问。

1.5.6 阅读代码

使用开源框架的一个好处是所有代码可在线获得以便自学。如果你有胆量，阅读源代码是理解框架运行机制最有效和最快捷的方法。

文档可能有时会模糊或不准确，但代码永远不会说谎。hapi.js 的模块是经过精心编码的，并由易于理解的小模块组成，把它们弄明白非常简单。

本章的某些讨论有点抽象，但在下一章，你将分步构建一个小型的真实 hapi 应用。

1.6 小结

- hapi 是一个帮助你构建网站、API 或任意类型 HTTP 服务的 Web 应用框架。
- hapi 应用的主要组成部分是服务器、连接、路由、handler 和插件。
- hapi 鼓励你利用它强大的插件系统和 npm，采用模块化方式构建应用。
- hapi 的理念是“配置优于代码”。