Broadview® www.broadview.com.cn

OpenResty 完全开发指南

构建百万级别并发的Web应用

罗剑锋 著

中国工信出版集团
電子工業出版社 PUBLISHING HOUSE OF ELECTRONICS INDUSTRY http://www.phei.com.cn

OpenResty

完全开发指南

构建百万级别并发的Web应用

罗剑锋 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

OpenResty 是一个基于 Nginx 的高性能 Web 平台，能够方便地搭建处理超高并发的动态 Web 应用、Web 服务和动态网关。

与现有的其他开发语言/环境相比，OpenResty 有着高性能、高灵活性、易于学习和扩展等许多优点，近年来得到了越来越多开发者的关注，也有了很多成功的应用范例，如 Adobe、Dropbox、GitHub 等知名公司都基于 OpenResty 构建了自己的后端业务应用。

OpenResty 自带完善的帮助文档，开发社区也很活跃，但相关的学习资料——特别是中文资料较少。本书基于作者多年使用 OpenResty 的经验，系统地阐述了 OpenResty 相关的各方面知识和要点，帮助读者快速掌握这个高效易用的 Web 开发平台，进而实现 HTTP/HTTPS/TCP/UDP 等多种网络应用。

本书结构严谨、详略得当，具有较强的实用性，适合广大软件开发工程师、系统运维工程师、编程爱好者和计算机专业学生阅读参考。

图书在版编目（CIP）数据

OpenResty 完全开发指南：构建百万级别并发的 Web 应用 / 罗剑锋著. —北京：电子工业出版社，2018.9
ISBN 978-7-121-34896-9

Ⅰ. ①O… Ⅱ. ①罗… Ⅲ. ①互联网络－网络服务器－程序设计 Ⅳ. ①TP368.5

中国版本图书馆 CIP 数据核字(2018)第 188129 号

策划编辑：孙学瑛
责任编辑：牛　勇
印　　刷：北京天宇星印刷厂
装　　订：北京天宇星印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：14.25　字数：317 千字
版　　次：2018 年 9 月第 1 版
印　　次：2018 年 9 月第 1 次印刷
定　　价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前言

随感

本书肇始于三年多前我的《Nginx 模块开发指南》一书。最初是以书末的附录形式出现，只有短短的几页，粗略地介绍了 OpenResty 的核心组件 ngx_lua。连我自己也没有想到，几年后的今天它竟然“脱胎换骨”，进化成了一本颇具规模的正式图书。

写作本书还是有感于目前国内技术书刊市场的现状。

十几二十几年前只有少数资深专家掌握核心技术，通过著书立说的方式来分享知识，普惠大众。但随着互联网的高速发展，知识的获取方式变得越来越简单了，任何人都可以在网上轻松地查找到所需的资料，也可以在网上很容易地发表文章。书——曾经被誉为“进步的阶梯”“精神的食粮”——已经不是那么重要了。

另一方面，互联网的普及也降低了书的严肃性和出版门槛。个人“恶意推测”，也许是为了“图省事”或者“赚快钱”，有相当多的人只是把若干博客文章集合在一起，再加以少量修改就“攒”成了一本书。这种“乱炖”“杂烩”形式的书籍拼凑的痕迹十分明显，缺乏内在的逻辑和连贯性，不过凭借着网络上积累的“人气”也能够获得不错的销量，但在我看来实在是对读者的不尊重和不负责。

一个极端的例子是前段时间偶然遇到的名为《□□开发实战》的书，其粗制滥造程度简直是“超乎想象”，“不料，也不信竟会凶残到这地步”[①]——几乎 90%的内容都是原封不动地拷贝网络上现成的材料，再有就是直接复制数十页全无注释的杂乱代码，通篇看下来可能只有前言是“原创”，着实是“钦佩”该书作者厚颜无耻的“功力”。

① 原文出自鲁迅《记念刘和珍君》。

单纯地感慨“世风日下”“人心不古”是没有用的，我所能做的，就是尽自己“微茫”的努力，写出一些无愧于己于人的文字。

关于 OpenResty

有这样一种说法：“Nginx 是网络世界里的操作系统，而 OpenResty 则是 Nginx 上的 Web 服务器”。

Nginx 在 Web Server 业内的领军地位早已经得到了公认，是高性能服务器的杰出代表。它采用 C 语言开发，能够跨平台运行，把性能挖掘优化技术发挥到了极致。正因为如此，Nginx 也很自然地成为了一个超越原生操作系统的开发平台，程序员可以完全无视底层系统之间的差异，在 Nginx 的框架里调用丰富的数据结构和功能接口，开发出高性能高可移植的各种应用程序。[①]

但基于 Nginx 开发主要使用的语言是 C/C++，开发难度高周期长，虽然没有达到“望而生畏”的程度但亦不远矣。好在 OpenResty 应运而生，在 Nginx 里嵌入了 LuaJIT 环境和 Lua 语言，就如同给裸系统添加了一个高效易用的 Shell，瞬间就让 Nginx 开发的难度直线下降，降低到了普通的心智模型可以理解掌握的水平。

早期 OpenResty 对于自身的定位主要还是 HTTP Server（其实也是受到 Nginx 的限制），可以利用“胶水语言”Lua 来操纵 Nginx，灵活定制业务逻辑，方便快捷地搭建出超高并发的各种 Web 服务，从而节约时间和人力成本。多年来的实践证明，这方面它的确工作得非常出色。

近两年 OpenResty 的发展开始加速，支持了 TCP/UDP 协议，扩充了众多的专用库、应用框架以及外围工具，逐渐形成了一个比较独立自洽的生态体系。虽然 Nginx 仍然是核心，但看得出 OpenResty 有淡化自身“Nginx Bundle”色彩的趋势，力图成为一个更伟大的存在。

随着软件基金会和商业公司的成立，OpenResty 获得了前所未有的成长动力。“路远，正未有穷期”，在此借本书送上诚挚的祝福与期待。

① 通常来说 Nginx 适合运行单线程的 I/O 密集型应用，但实际上它也可以使用多线程技术运行 CPU 密集型应用。

致谢

首先要感谢 Nginx 的作者 Igor Sysoev 和 OpenResty 的作者 agentzh，正是因为他们多年来持续无私的奉献，我们才能够拥有如此强大易用的 Web Server。

接下来我要感谢父母多年来的养育之恩，感谢妻子和两个可爱的女儿（“点心”组合）在生活中的陪伴，愿你们能够永远幸福快乐。

我也要感谢读者选择本书，希望读者能从中汲取有用的知识，让 OpenResty 成为工作中的得力助手。

您的朋友 罗剑锋

2018 年 7 月 18 日 于 北京 798 园区

目录

第 0 章

导读

0.1 关于本书

OpenResty 是一个基于 Nginx 的高性能 Web 平台，能够方便地使用动态脚本语言 Lua 搭建高并发、高扩展性的 Web 应用和动态网关。

与 Go、PHP、Python、Node.js 等现有的其他 Web 开发语言/环境相比，OpenResty 具有高性能、高灵活性、易于学习和扩展等许多优点，在高性能 Web 编程领域得到了广泛的应用，已经有为数众多的国内外大公司基于 OpenResty 构建了自己的业务服务，例如 Adobe、DropBox、GitHub 等。

虽然 OpenResty 自带完善的帮助文档，开发社区也很活跃，但它毕竟还是一个较新的开发平台，相关的学习资料较少，了解的人不多。有鉴于此，作者基于多年使用 OpenResty 的经验编写了本书，希望能够为 OpenResty 的普及尽一份自己的力量，也希望读者能够利用 OpenResty 开发出更多更好的 Web 应用。

0.2 读者对象

本书适合以下各类读者：

- 追求高性能的 Web 应用研发工程师；
- 微服务、API 网关、Web 应用防火墙的研发工程师；
- 手机游戏、网络游戏后端服务器的研发工程师；
- 通用的 HTTP/TCP/UDP 应用服务研发工程师；
- 工作在 Linux 系统上的运维、测试工程师；

- 计算机编程爱好者和在校学生。

随着宽带网络的快速普及和移动互联网的高速发展，网站需要为越来越多的用户提供服务，处理越来越多的并发请求，要求服务器必须具有很高的性能才能应对不断增长的需求和突发的访问高峰。在超高并发请求的场景下，很多常用的服务开发框架都会显得“力不从心”，服务能力严重下降，很难优化。这时我们就可以选择 OpenResty，它内置高效的事件驱动模型和进程池机制，直接在“起点”上构建高性能的 Web 应用，而且可以很容易地调整配置参数进一步释放系统潜力。

现在的网站架构大都是分布式系统，经常部署有成百上千个内部模块，这些模块通常基于“微服务”“服务网格”等架构，彼此之间的联系十分复杂，在演化过程中系统会逐渐变得难以理解和维护。OpenResty 具有优秀的反向代理和负载均衡能力，在复杂的分布式系统中可以充当 API Gateway 的角色，分治整合不同种类的服务简化系统，并使用内嵌的 Lua 脚本添加缓冲、限流、防护、认证等额外功能。

游戏类应用是目前互联网特别是移动互联网上的一大类应用，用户量很庞大。这些应用多数会提供在线服务，要求后端稳定可靠，最好还能够快速修复错误或上线新功能。OpenResty 非常适合担当这样的后端服务器，它不仅性能高运行稳定，而且由于使用的是动态脚本语言 Lua，上手容易且功能丰富，能够很好地缩短产品的开发周期，实现快速迭代。此外，因为 Lua 语言也经常被用于游戏客户端开发，使用 OpenResty 还可以打通前后端，统一开发语言，轻松地成为掌控全局的“全栈”工程师。

除了开发标准的 Web 应用，OpenResty 也是一个通用的服务器开发框架，内部结构良好，基础设施完备，支持 HTTP/HTTPS/WebSocket/TCP/UDP 等多种网络协议和 Redis、MySQL 等常见的数据库，能够使用 Lua 以简单易理解的同步非阻塞模式编程，快捷实现各种业务逻辑，部分或完全取代自研框架或其他开源框架，实现任意的后台服务。

OpenResty 不仅是一个单纯的服务器软件，它还是一个完整的应用环境，其中就包含了一个非常有用的 Lua 脚本解释器。运维、测试工程师可以使用小巧灵活的 Lua 代替 Shell、Perl、Python 等语言，调用 OpenResty 的内部功能接口写出高效实用的各种脚本，如系统管理、状态监控、单元测试或压力测试，方便自己的工作。

最关键的一点，OpenResty 是完全开源的，拥有成熟活跃的开发社区和许多顶级的开发者，研究、参与它可以领会真正的“开源”精神，学习到各种前沿的理念、技术和知识，提高自身的能力。

0.3 读者要求

本书不要求读者具备专门的编程语言知识。

开发 Web 服务通常给人的感觉是很难，要掌握复杂的开发技巧和应用框架，但 OpenResty 把这个门槛大大降低了。它的开发模型非常直观容易理解，而且使用的主力编程语言是 Lua，一种小巧轻便的动态脚本语言，学习难度很低，只要具有初级编程经验就可以快速掌握。

但如果想要深入 OpenResty 底层编写出更高效的代码，还是建议读者多了解一些 C 语言和 Linux 的相关知识。

0.4 运行环境

OpenResty 可以跨平台编译和运行，支持 Linux、FreeBSD、macOS、Windows 等多种操作系统。但就目前市场来看，Linux 是应用最普及的服务器操作系统，故本书的开发环境选用 Linux。

Linux 有很多的发行版本，企业中使用较多的是偏重于稳定性的 CentOS 和偏重于易用性的 Ubuntu，出于个人喜好的原因本书选择了 Ubuntu，版本号是 16.04.03。

0.5 本书的结构

对于大多数读者来说，OpenResty 可能都是一个“陌生”的开发环境，所以本书采用循序渐进的方式组织全书的章节：首先介绍基本知识作为入门，然后解析运行机制和开发流程，再由浅入深地逐步讲解功能接口和如何开发各种 Web 服务。

全书共 15 章，各章的内容简介如下。

- 第 1 章：总论

本章简要介绍 OpenResty 的历史、组成和编译安装的方法。

- 第 2 章：Nginx 平台

Nginx 是 OpenResty 的核心部件，本章介绍了它的特点、进程模型和各种应用服务的配置方法。

- 第 3 章：Lua 语言

本章讲解 OpenResty 的工作语言 Lua，包括详细的语法和标准库。

- 第 4 章：LuaJIT 环境

本章介绍 OpenResty 使用的 Lua 运行环境 LuaJIT，它的运行效率更高，而且提供很多特别的优化和库，比原生的 Lua 更加强大。

- 第 5 章：开发概述

本章在宏观的层次介绍开发 OpenResty 应用的基本流程、配置指令、运行机制等知识，帮助读者从总体上理解掌握 OpenResty。

- 第 6 章：基础功能

本章介绍 OpenResty 里的一些基础功能，如系统信息、日志、时间日期、编码格式转换、正则表达式、高速缓存等。

- 第 7 章：HTTP 服务

本章介绍 OpenResty 为开发 HTTP 服务提供的大量功能接口，操纵 HTTP 请求和响应，学习完本章就能够轻松开发出高性能的 Web 应用。

- 第 8 章：访问后端

本章介绍 OpenResty 提供的两种高效通信机制：location.capture 和 cosocket，还有基于它们实现的一些客户端库，可以访问 HTTP、Redis、MySQL 等多种后端。

- 第 9 章：反向代理

本章介绍 OpenResty 的反向代理功能，搭建动态网关，并使用 ngx.upstream 和 ngx.balancer 实现深度定制。

- 第 10 章：高级功能

本章介绍 OpenResty 里的共享内存、定时器、进程管理和轻量级线程这四个高级功能。

- 第 11 章：HTTPS 服务

本章介绍如何在 OpenResty 里开发 HTTPS 服务，实践动态加载证书、动态查验证书和会话复用等 HTTPS 优化技术。

- 第 12 章：HTTP2 服务

本章介绍如何在 OpenResty 里开发 HTTP2 服务。

- 第 13 章：WebSocket 服务

本章介绍如何在 OpenResty 里开发 WebSocket 服务。

- 第 14 章：TCP/UDP 服务

本章介绍 OpenResty 里处理 TCP/UDP 协议的 stream 子系统，能够基于 TCP/UDP 协议开发出更通用的 Web 服务。

- 第 15 章：结束语

本章给出了读者在阅读完本书后进一步学习研究 OpenResty 的方向。

0.6 如何阅读本书

初次接触 OpenResty 的读者应当先阅读第 1 章至第 4 章，在本书的指导下安装配置 OpenResty，搭建自己的开发环境，了解 OpenResty 的组成、运行平台和工作语言，然后再学习后续的章节。

如果读者已经比较熟悉 OpenResty，那么可以跳过前 4 章，从第 5 章开始顺序阅读，或者检索目录直接跳到感兴趣的章节，研究 OpenResty 的内部运行机制、功能接口、各种服务的配置和开发方式。

0.7 本书的源码

为方便读者利用本书学习研究 OpenResty，作者在 GitHub 网站上发布了本书内所有示例程序的源代码，地址是：

```
https://github.com/chronolaw/openresty_dev.git      #OpenResty 开发示例
```

对于想深入钻研的读者，还有另外两个项目供进一步参考：

```
https://github.com/chronolaw/annotated_nginx.git    #Nginx 源码详细注解
https://github.com/chronolaw/favorite-nginx.git     #各种有用的 Nginx 资源
```

第 1 章

总论

由 agentzh 创立的开源项目 OpenResty 成功地把 Lua 语言嵌入了 Nginx，用 Lua 作为“胶水语言”粘合 Nginx 的各个模块和底层接口，以脚本的方式直接实现复杂的 HTTP/TCP/UDP 业务逻辑，降低了 Web Server——特别是高性能 Web Server 的开发门槛。

很多国内外大型网站都在使用 OpenResty 开发后端应用，而且越来越多，知名的国外公司有 Adobe、CloudFlare、Dropbox、GitHub 等，国内则有 12306、阿里、爱奇艺、京东、美团、奇虎、新浪等，充分地证明了 OpenResty 的优秀。[①]

本章将简略地介绍 OpenResty 的历史、特点和组成，带领读者初步感受它的风采。

1.1 简介

在官网上对 OpenResty 是这样介绍的（http://openresty.org）：

“OpenResty 是一个基于 Nginx 与 Lua 的高性能 Web 平台，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。

“OpenResty 通过汇聚各种设计精良的 Nginx 模块（主要由 OpenResty 团队自主开发），从而将 Nginx 有效地变成一个强大的通用 Web 应用平台。这样，Web 开发人员和系统工程师可以使用 Lua 脚本语言调动 Nginx 支持的各种 C 以及 Lua 模块，快速构造出足以胜任 10K 乃至 1000K 以上单机并发连接的高性能 Web 应用系统。

① 有统计数据表明，在 2017 年中，互联网上使用 OpenResty 的主机数量已经超过了 25 万。

“OpenResty 的目标是让你的 Web 服务直接跑在 Nginx 服务内部，充分利用 Nginx 的非阻塞 I/O 模型，不仅仅对 HTTP 客户端请求，甚至于对远程后端诸如 MySQL、PostgreSQL、Memcached 以及 Redis 等都进行一致的高性能响应。”

从这段描述里我们可以知道，OpenResty 以 Nginx 为核心，集成打包了众多侧重于高性能 Web 开发的外围组件，它既是一个 Web Server，也是一个成熟完善的开发套件。

OpenResty 基于 Nginx 和 Lua/LuaJIT，充分利用了两者的优势，能够无阻塞地处理海量并发连接，任意操纵 HTTP/TCP/UDP 数据流，而且功能代码不需要编译，可以就地修改脚本并运行，简化了开发流程，加快了开发和调试的速度，同时也缩短了开发周期，在如今这个快节奏的时代里弥足珍贵。

更广义地来看，OpenResty 不仅仅是一个单纯的 Web 服务开发套件。经过多年的发展，围绕着 OpenResty 已经聚集了很多的个人用户和商业公司，开发出大量的第三方库和应用框架，每年还会定期举办技术研讨会——这些都标志着 OpenResty 已经成长为了一个活跃的开源社区和完整的生态环境。

1.2 历史

2007 年，受到当时风行的 OpenAPI 和 REST 潮流的影响，agentzh 使用 Perl 语言（还有少量的 Haskell）开发出了一套 Web Service 框架，也就是如今 OpenResty 的雏形。由于 Perl 语言自身的限制，虽然 agentzh 做了很多优化工作，但性能始终无法令人满意。①

2009 年，在综合比较了 Apache、Lighttpd 和 Nginx 等服务器框架的优劣之后，agentzh 决定以 Nginx 作为新的开发平台，与同事 chaoslawful 合力用 C 语言重新设计和实现了之前的框架，并选择小巧紧凑的动态脚本语言 Lua 作为上层的用户语言。就这样，我们所熟悉的高性能服务器开发包 OpenResty 诞生了。

2011 年，随着 OpenResty 的用户逐渐增多，开源项目与本职工作的冲突越来越严重，agentzh 于是辞职在家，专心维护 OpenResty，为全世界的程序员提供“免费服务”。

2012 年，旧金山的一家公司向 agentzh 发出邀请，支持他以全职状态继续开发 OpenResty。没有了后顾之忧，agentzh 全心投入到了开源事业中，为 OpenResty 增加了大量的新功能，这段时间是 OpenResty 的迅速成长期。

2015 年，首届 OpenResty 开发大会在北京召开。大会汇集了多个国内外公司和开发者，

① 第一代的 OpenResty 可参见 https://github.com/agentzh/old-openresty。

agentzh 本人也亲自莅临会场，总结回顾 OpenResty 的历程，展望将来的发展目标。

2016 年，OpenResty 软件基金会在香港成立，并获得了国内某科技公司 100 万元的捐赠，基金会的主要目标是促进、资助 OpenResty 相关的开源项目。

2017 年，agentzh 在旧金山成立了公司 OpenResty Inc，探索商业化的可能，并很快于年中发布了流量管理产品“OpenResty Edge 2”。

1.3 组成

OpenResty 并不是个“单块”（Monolithic）的程序，而是由众多设计精良的组件集合而成的，这些组件可以灵活组装或拆卸，共同搭建起了完整的高性能服务器开发环境。

核心组件

OpenResty 的核心组成部分有四个，分别是：

- Nginx ：高性能的 Web 服务器（不熟悉的读者可阅读第 2 章）；
- LuaJIT ：高效的 Lua 语言解释器/编译器；
- ngx_lua（http_lua） ：处理 HTTP 协议，让 Lua 程序嵌入在 Nginx 里运行；
- stream_lua ：与 ngx_lua 类似，但处理的是 TCP/UDP 协议。

使用这四个核心组件，OpenResty 就可以完成相当多的网络应用开发工作了，但 OpenResty 远不止如此，它还包含了其他一些非常有用的 Nginx 组件和 Lua 组件，进一步增加了开发工作的便利。[①]

Nginx 组件

OpenResty 里的 Nginx 组件以 C 模块的方式提供，集成在 Nginx 内部，较常用的有：

- ngx_iconv ：转换不同的字符集编码；
- ngx_encrypted ：使用 AES-256 算法执行简单的加密运算；
- ngx_echo ：提供一系列“echo”风格的指令和变量；
- ngx_set_misc ：增强的“set_xxx”指令，用来操作变量；
- ngx_headers_more ：更方便地处理 HTTP 请求头和响应头的指令；
- ngx_memc ：支持各种 memcached 操作；

① 为了叙述方便，本书约定用 C 语言实现的 OpenResty 组件名字加“ngx”前缀，用 Lua 语言实现的 OpenResty 组件名字加“lua”或“lua-resty”前缀。

- ngx_redis2 ：支持各种 Redis 操作；
- ngx_dizzle ：支持各种 MySQL 操作；
- ngx_postgres ：支持各种 PostgreSQL 操作。

Lua 组件

OpenResty 里的 Lua 组件通常以 Lua 源码的方式提供(*.lua)，但个别组件为追求效率会以 C 语言实现，是动态链接库的形式（*.so)。

较常用的 Lua 组件有：

- lua_core ：OpenResty 的核心功能库；
- lua_cjson ：处理 JSON 格式的数据，速度很快（使用 C 语言实现)；
- lua_string ：hex/md5/sha1/sha256 等字符串功能；
- lua_upload ：流式读取 HTTP 的上行数据；
- lua_healthcheck ：后端集群健康检查；
- lua_limit_traffic ：定制流量控制策略；
- lua_lock ：基于共享内存的非阻塞锁；
- lua_lrucache ：高效的 LRU 缓存功能；
- lua_dns ：高效、非阻塞的 DNS 解析功能；
- lua_websocket ：高效、非阻塞的 WebSocket 功能；
- lua_redis ：Redis 客户端，用起来比 ngx_redis2 更灵活；
- lua_memcached ：Memcached 客户端，用起来比 ngx_memc 更灵活；
- lua_mysql ：MySQL 客户端，用起来比 ngx_dizzle 更灵活。[①]

辅助工具

核心组件、Nginx 组件和 Lua 组件实现了 OpenResty 的主要功能，但作为集成开发环境，辅助开发、调试和运维的工具也是必不可少的。OpenResty 目前提供的辅助工具有：

- opm ：类似 rpm、npm 的管理工具，用来安装各种功能组件；
- resty-cli ：以命令行的形式直接执行 OpenResty/Lua 程序；
- restydoc ：类似 man 的参考手册，非常详细。

组件示意图

综上可见，OpenResty 是一个功能非常完备的服务器开发包，大多数 Web 应用所需的功

① 目前 OpenResty 发行包暂不含有操作 PostgreSQL 的 Lua 组件，但可以通过 opm 安装。

能都已经包含在了里面，也就是所谓的“out of box”，我们只需要简单地在自己的程序里引用，就能够轻松享用这些高质量的模块和库，从而快速实现新的业务。

OpenResty 的组成可以用图 1-1 来表示：

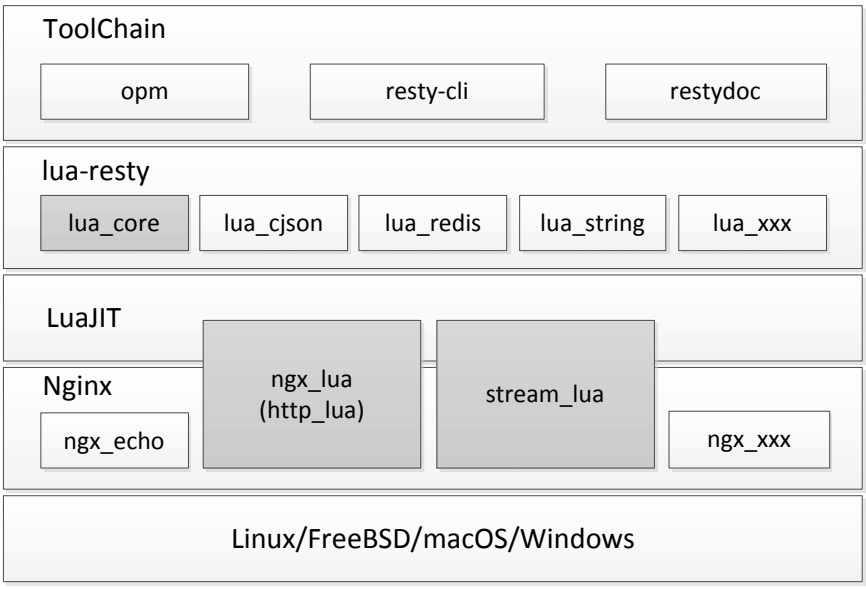

图 1-1 OpenResty 的组成

1.4 版本

OpenResty 使用四位数字作为版本号，形式是：a.b.c.x，其中前三位数字是内部 Nginx 的版本，作为大版本号，第四位数字是 OpenResty 自己的发布版本号，也就是小版本号。[①]

例如，OpenResty 1.13.6.1 表示包内部使用的是 Nginx 1.13.6，是本次大版本的第一个发布版本；OpenResty 1.9.7.5 表示包内部使用的是 Nginx 1.9.7，是第五个发布版本。

由于 OpenResty 每次更新都会增加很多新功能和错误修复，建议读者及时使用最新的版

① Nginx 是 OpenResty 里最核心的组成部分，是 OpenResty 的“基础运行平台”，所以 OpenResty 的发布版本就“追随”了 Nginx。个人认为这种方式比有的 Nginx Fork“另起炉灶”的版本号要好，核心的 Nginx 版本一目了然。

本，本书使用的 OpenResty 是 2018 年中发布的 1.13.6.2 版。

1.5 安装

OpenResty 主要以源码的方式发布，可以在多种操作系统上编译和运行，例如 Linux、FreeBSD、macOS、Windows 等，源码可以从官网直接下载（www.openresty.org）。

当然，从源码编译安装还是比较麻烦的，不利于企业大规模的部署，OpenResty 也对某些主流操作系统提供了预编译包，只需要很简单的操作即可完成安装，无须编译源码的长时间等待。如果使用 docker，更可以直接从 Docker Hub 上拉取现成的镜像。

但本书还是推荐以源码的方式安装 OpenResty，不仅能够支持任意操作系统，更可以更灵活地定制所需的功能。

1.5.1 直接安装

本节简要叙述 OpenResty 在 Linux、Windows 和 Docker 上的安装方式。

Linux

OpenResty 可以在 Linux 系的 Ubuntu/Debian、CentOS/Fedora/RHEL 等系统里直接安装，具体的方法可参见官网。

以 Ubuntu 为例，需执行下面的几条命令：

```
#导入 GPG key
wget -qO - https://openresty.org/package/pubkey.gpg | sudo apt-key add -

#安装命令 add-apt-repository
sudo apt-get -y install software-properties-common

#添加 OpenResty 官方源
sudo add-apt-repository -y \
  "deb http://openresty.org/package/ubuntu $(lsb_release -sc) main"

#更新源
sudo apt-get update

#开始安装 OpenResty
sudo apt-get install openresty
```

OpenResty 会默认安装到“/usr/local/openresty/”目录下。

Windows

对于 Windows 系统，OpenResty 提供两个 zip 包，里面是全套编译好的 Win32/Win64 可执行程序，解压后即可使用，非常方便。

由于 Windows 系统的原因，运行在 Windows 上的 OpenResty 的性能和稳定性没有 Linux 上的高，所以建议 Windows 版的 OpenResty 仅用于学习和测试，最好不要用于正式的生产环境（如果对性能和稳定性要求不高则另当别论）。

Docker

Docker 用户安装 OpenResty 是最简单的，用命令“docker pull openresty/openresty”就可以获取打包好的镜像。

1.5.2 源码安装

以源码的方式安装 OpenResty 有一些编译依赖，需要系统里有 C 编译器（通常是 gcc）、Perl、libpcre、libssl 等，可以使用 apt-get 或者 yum 等工具安装，例如：

```
apt-get install gcc libpcre3-dev \          #安装 gcc 等编译依赖
    libssl-dev perl make build-essential
```

之后我们就可以从官网上下载源码压缩包，解压后执行 configure 再 make 编译：

```
wget https://openresty.org/download/openresty-1.13.6.2.tar.gz
tar xvfz openresty-1.13.6.2.tar.gz          #解压缩
cd openresty-1.13.6.2                       #进入源码目录

./configure                                 #编译前的配置工作
make                                        #编译
sudo make install                           #安装
```

与直接安装相同，OpenResty 也会默认安装到“/usr/local/openresty/”目录下。

为了方便使用，建议在“~/.bashrc”文件里把安装目录添加到环境变量 PATH：

```
export PATH=/usr/local/openresty/bin:$PATH
```

1.5.3 定制安装

使用源码安装 OpenResty 有一个好处：可以在编译前的 configure 时指定各种配置选项，如编译参数、安装目录、添加或删除功能组件等，让 OpenResty 更符合我们的实际需要。不过这属于 OpenResty 比较高级的特性，通常默认的配置就足够了。

使用参数“--help”可以列出 configure 的详细说明，为节省篇幅这里不一一列出（也无必要）。下面仅举一个小例子，把 OpenResty 安装到“/opt/openresty”目录，启用 HTTP2 和真实 IP 功能，禁用 FastCGI 和 SCGI，OpenSSL 使用 1.0.2k：

```
./configure                                        #编译前的配置工作
    --prefix=/opt/openresty                \       #指定安装到/opt/openresty 目录下
    --with-http_v2_module                  \       #支持 HTTP2
    --with-http_realip_module              \       #反向代理时可转发客户端真实 IP 地址
    --without-http_fastcgi_module          \       #不使用 fastcgi
    --without-http_scgi_module             \       #不使用 scgi
    --with-openssl="path/to/openssl-1.0.2k" #使用 OpenSSL 1.0.2k
```

更深入地定制 OpenResty 需要学习 Nginx 相关的知识，读者可参考附录 A 的推荐书目[5]以及附录 B。

1.6 目录结构

安装之后 OpenResty 的目录结构如下（以默认安装目录为例）：

```
/usr/local/openresty/                              #安装主目录
├── bin                                            #存放可执行文件
├── luajit                                         #LuaJIT 运行库
├── lualib                                         #Lua 组件
├── nginx                                          #Nginx 核心运行平台
├── pod                                            #参考手册（restydoc）使用的数据
└── site                                           #包管理工具（opm）使用的数据
```

通常我们需要关注的是 bin 和 lualib 目录。

bin 目录里存放的是 OpenResty 可执行文件，关系到 OpenResty 的运行，较重要的有：

- openresty ：可执行文件，用来启动 OpenResty 服务（见 1.7 节）。[①]
- opm ：组件管理工具，用来安装各种功能组件（见 1.8 节）；
- resty ：命令行工具，可直接执行 Lua 程序（见 1.9 节）；
- restydoc ：参考手册（见 1.10 节）。

lualib 目录里存放的是 OpenResty 自带的 Lua 组件，如 lua_cjson、lua_core 等。

① bin/openresty 是对安装目录里 nginx/sbin/nginx 的符号链接，实际上就是 Nginx。这种做法更好地凸显了 OpenResty，而且屏蔽了内部的目录结构细节，避免了与系统里可能存在的其他 Nginx 实例的冲突。

1.7 启停服务

启动和停止 OpenResty 需要以 root 身份，或者使用 sudo。

直接运行 bin/openresty 就可以启动 OpenResty：

```
/usr/local/openresty/bin/openresty                    #启动 OpenResty 服务
```

OpenResty 默认开启了 localhost:80 服务，使用 wget 或者 curl 这样的工具就可以验证 OpenResty 是否正常工作：

```
curl -vo /dev/null http://localhost/index.html      #curl 命令发送 HTTP 请求
```

如果 OpenResty 正在运行，那么 curl 的部分输出可能如下：

```
* Connected to localhost (127.0.0.1) port 80 (#0)  #连接到 localhost:80

> GET /index.html HTTP/1.1                         #获取文件 index.html
> User-Agent: curl/7.35.0                          #curl 的版本号

< HTTP/1.1 200 OK                                  #响应码 200，工作正常
< Server: openresty/1.13.6.2                       #服务器是 OpenResty
< Content-Type: text/html                          #响应内容是普通文本
< Content-Length: 558                              #HTTP 正文长度是 558 字节
```

参数“-s stop”可以停止 OpenResty（注意同样需要以 root 身份或 sudo）：

```
/usr/local/openresty/bin/openresty  -s stop        #停止 OpenResty 服务
```

更多的 OpenResty 运行命令可以参见 5.2 节。

1.8 组件管理工具

很多开发语言/环境都会提供配套的包管理工具，例如 npm/Node.js、cpan/Perl、gem/Ruby 等，它们可以方便地安装功能组件，辅助用户的开发工作，节约用户的时间和精力。OpenResty 也有功能类似的工具，名字叫 opm。

OpenResty 维护一个官方组件库（opm.openresty.org），opm 就是库的客户端，可以把组件库里的组件下载到本地，并管理本地的组件列表。[①]

① opm 不仅适用于 OpenResty 用户，也适用于 OpenResty 库开发者，允许他们上传组件到官方网站，只需要编写一个简单的 dist.ini 即可，本书暂不做介绍。

opm 的用法很简单，常用的命令有：

- search ：以关键字检索相关的组件；
- get ：安装功能组件（注意不是 install）；
- info ：显示已安装组件的详细信息；
- list ：列出所有本地已经安装的组件；
- upgrade ：更新某个已安装组件；
- update ：更新所有已安装组件；
- remove ：移除某个已安装组件。

opm 默认的操作目录是“/usr/local/openresty/site”，但我们也可以在命令前使用参数“--install-dir=*PATH*”安装到其他目录，或者用参数“--cwd”安装到当前目录的“./resty_modules/”目录里。

下面的命令示范了 opm 的部分用法：

```
opm search    http                          #搜索关键字 http
opm search    kafka                         #搜索关键字 kafka
opm get       agentzh/lua-resty-http        #安装组件，注意需要 sudo
opm info      agentzh/lua-resty-http        #显示组件的版本、作者等信息
opm remove    agentzh/lua-resty-http        #移除组件，同样需要 sudo

opm --install-dir=/opt   get xxx            #把组件安装到/opt 目录下
opm --cwd                get xxx            #安装到当前目录的/resty_modules 下
```

需要注意的是 opm 里组件的名字，使用的是类似 GitHub 的格式，即“作者名/组件名”，允许一个组件有多个不同的作者和版本，方便组件开发者“百家争鸣”，由用户来评估决定使用哪一个。

由于 opm 在 OpenResty 里出现的较晚（2016 年），目前库里可用的组件还不多，希望假以时日能够丰富壮大。

1.9 命令行工具

OpenResty 在 bin 目录下提供一个命令行工具 resty（注意名字不是 resty-cli），可以把它作为 Lua 语言的解释器（但运行在 OpenResty 环境里）代替标准的 Lua 5.x，写出类似 Perl、Python 那样易用的脚本，是测试/运维工程师的利器。[①]

① resty 的工作原理是启动了一个“无服务”的 Nginx 实例，禁用了 daemon 等大多数指令，也没有配置监听端口，只是在 worker 进程里用定时器让 Lua 代码在 Nginx 里执行。

使用“-e”参数可以在命令行里直接执行 Lua 代码，例如：

```
./resty -e "print('hello OpenResty')"        #执行 Lua 代码，打印一个字符串
```

这种方式只适合执行很小的代码片段，更好的方式是利用 UNIX 的“Shebang”(#!)，在脚本文件里的第一行指定 resty 作为解释器，能够书写任意长度和复杂度的代码，而且更利于管理维护。

刚才的命令行用法可以改写成下面的脚本文件：①

```
#!/usr/local/openresty/bin/resty          -- 使用 resty 作为脚本的解释器
print('hello OpenResty')                  -- 执行 Lua 代码，打印一个字符串
```

脚本文件也支持传递命令行参数，参数存储在表 arg 里，用 arg[*N*]的方式即可访问：

```
#!/usr/local/openresty/bin/resty          -- 使用 resty 作为脚本的解释器

local n = #arg                            -- 得到参数的数量
print("args count = ", n)                 -- 打印参数的数量

for i = 1,n do                            -- 变量参数表，注意 Lua 下标从 1 开始
    print("arg ", i , ": ", arg[i])       -- 输出参数
end                                       -- 循环结束
```

使用参数执行脚本 hello.lua，结果是：

```
./hello.lua FireEmblem Heroes             #执行 Lua 代码，带两个参数
args count = 2                            #打印参数的数量
arg 1: FireEmblem                         #输出第一个参数
arg 2: Heroes                             #输出第二个参数
```

resty 工具还有很多选项用于配置行为，非常灵活，“-e”之外较常用的有：

- -c ：指定最大并发连接数（默认值是 64）；
- -I ：指定 Lua 库的搜索路径；
- -l ：指定加载某个 Lua 库；
- --http-conf ：定制在 http 域里的指令；
- --main-include ：定制在 main 域里的指令；
- --shdict ：定制使用的共享内存（参见 10.2 节）；
- --resolve-ipv6 ：允许解析 ipv6 的地址。

其他选项如-j、-gdb 等读者可以参考 help 或者 restydoc。

本书之后在讲解 Lua 语言和 LuaJIT 环境时均采用 resty 作为解释器执行 Lua 程序。

① 在某些系统上可能要使用“#!/usr/bin/env /usr/local/openresty/bin/resty”的形式。

1.10 参考手册

OpenResty 附带了非常完善的用户参考手册 restydoc，提供与 UNIX 手册 man 相同的功能，可以检索 OpenResty 里所有组件的帮助文档，包括但不限于：

- OpenResty 各个组件的介绍和用法；
- OpenResty 指令和功能接口的用法；
- Nginx 介绍、用法、基本工作原理；
- Lua/LuaJIT 语法要素。

下面示范了一些 restydoc 的用法，其中的“-s”参数用来指定搜索手册里的小节名：

```
restydoc nginx                          #Nginx 的说明
restydoc luajit                         #LuaJIT 的说明
restydoc opm                            #包管理工具 opm 的说明
restydoc resty-cli                      #命令行工具 resty 的说明
restydoc ngx_echo                       #ngx_echo 组件的说明
restydoc ngx_lua                        #ngx_lua 的说明
restydoc stream_lua                     #stream_lua 的说明
restydoc lua-cjson                      #lua-cjson 的说明

restydoc -s proxy_pass                  #反向代理指令 proxy_pass 的说明
restydoc -s content_by_lua_block        #content_by_lua_block 指令的说明
restydoc -s ngx.say                     #功能接口 ngx.say 的说明
restydoc -s concat                      #Lua 函数 concat 的说明
```

对于使用 opm 安装的组件，需要使用“-r”参数指定安装目录，例如：

```
restydoc -r /usr/local/openresty/site -s lua-resty-http
```

多使用 restydoc 可以帮助我们尽快熟悉 OpenResty 开发。

1.11 性能对比

虽然 OpenResty 基于高性能的 Nginx，目前也已经有了诸多的成功应用案例，但仍然有很多人对它抱有疑虑、持观望态度。一个可能的原因是它使用了较为“小众”的脚本语言 Lua，与其他常见的开发语言相比社区很小，而且也没有大公司为之“背书”，知名度低导致不了解和偏见。但实际上，OpenResty 在开发效率和运行效率上都超过了它的竞争对手。

我们可以用实际的例子来对比验证一下 OpenResty 的运行效率，比较的对象是与 OpenResty 类似、目前较为流行的 Web 开发语言/环境：Node.js、Go、PHP 和 Python。

测试方式是各自实现一个最简单的 HTTP 服务，不做任何额外的优化调整，直接返回“Hello World”字符串（具体的程序可以在 GitHub 上找到，位于 benchmark 目录）。

各语言/环境的详细信息如下：

- OpenResty ：版本号 1.13.6.1，源码编译。
- Node.js ：版本号 4.2.6，apt-get 安装。
- Go ：版本号 1.6.2，apt-get 安装。
- PHP ：版本号 7.0.22（运行在 Apache2.4.18 上），apt-get 安装。
- Python ：版本号 2.7.12，apt-get 安装。①

测试环境是一个单核 Linux 虚拟机，下面的表格是使用“ab -c 100 -n 10000”（并发 100 个连接，共 10000 个请求）测试得到的结果：②

	OpenResty	Node.js	Go	PHP	Python
Time(seconds)	1.328	4.133	2.171	3.368	N/A
Transferred(bytes)	1750000	1130000	1290000	1790000	N/A
RPS(#/s)	7529.50	2419.58	4605.53	2969.21	N/A
TPS(ms)	13.281	41.330	21.713	33.679	N/A
Transfer rate(kb/s)	1286.78	267.00	580.19	519.03	N/A

ab 测试结果如图 1-2 所示：

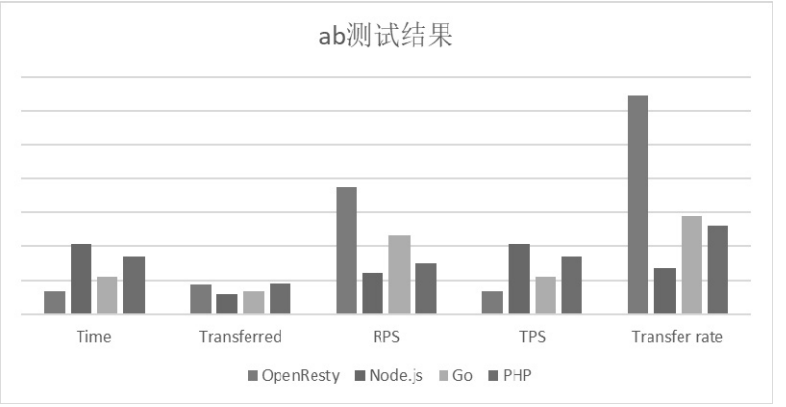

图 1-2 ab 测试结果

① 本书作者并不很擅长 Python，所以测试程序只使用了内置的、最简单的 HTTPServer，并未使用 twisted、tornado、gevent 等框架，可能有些不公平，望见谅。

② 在这次测试中 Python 发生了“Broken Pipe”错误，未能完成测试，故没有数据。

由表中的数据可见 OpenResty 的运行效率是最高的，在 RPS 指标上是 Node.js 的 3.1 倍，Go 的 1.6 倍，PHP 的 2.5 倍，远远胜出。

单使用 ab 测试可能还不足以说明问题，我们还可以使用 http_load 再运行另一个测试，参数是“-p 50 -s 5”（并发 50 个连接，持续 5 秒），测试结果如下：

	OpenResty	Node.js	Go	PHP	Python
fetches	53581	16954	29060	18167	8630
fetches/sec	10716.2	3390.8	5809.38	3633.4	1726
bytes/sec	128594	40689.6	69712.5	43600.8	20712
msecs/connect	0.0614273	0.04395150	0.03695	0.0387042	2.40234
msecs/first-response	4.14641	14.5356	8.53696	13.6824	3.97899

http_load 测试结果如图 1-3 所示：

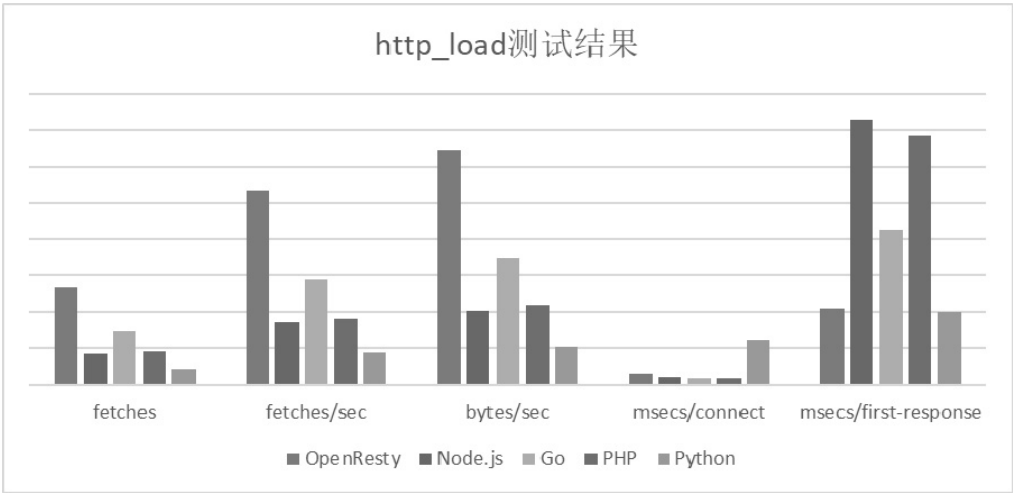

图 1-3 http_load 测试结果

毫无意外，在这次测试中 OpenResty 仍然是遥遥领先，在重要的 fetches/sec 指标上是 Node.js 的 3.2 倍，Go 的 1.8 倍，PHP 的 2.9 倍，Python 的 6.2 倍。

对于高负荷的网站来说，即使是 5%~10%的性能提升都是非常有价值的，更何况是 50%~200%。注意这还是未经优化的结果，实际上 OpenResty 还可以轻松开启多个进程服务，成倍地扩充服务能力。

相信经过这两轮测试，读者心中应该可以得到明显的结论了。

1.12 应用架构

OpenResty 功能丰富、开发简单而且性能极高，处理静态内容或动态内容都很擅长，所以在大中型应用系统中能够扮演多种角色，胜任多种工作，是不折不扣的“多面手”。

一个典型的以 OpenResty 为核心的应用系统架构如图 1-4 所示：

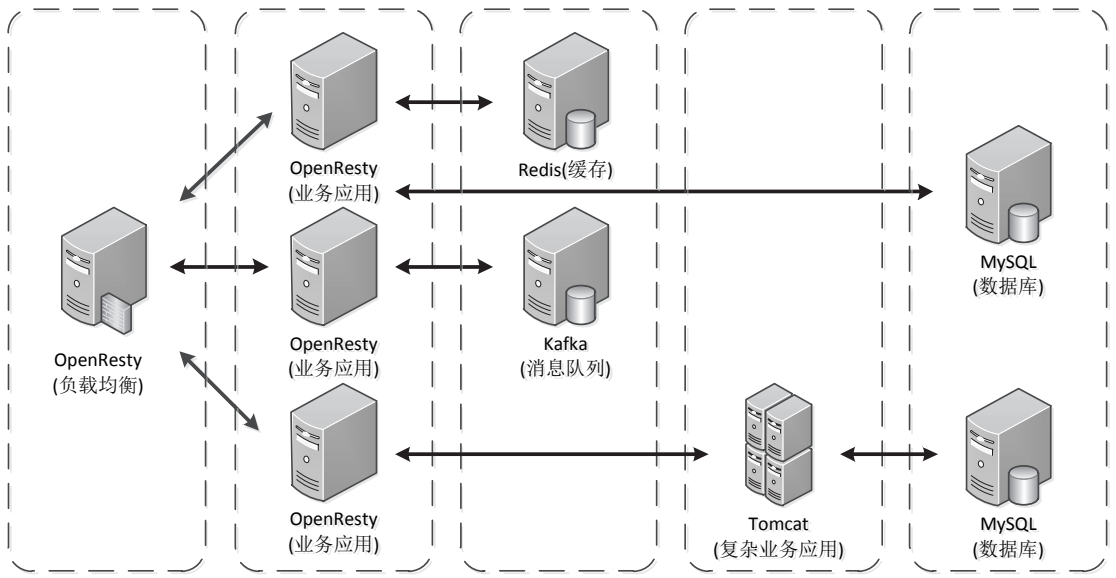

图 1-4 以 OpenResty 为核心的应用系统架构

由于 OpenResty 具有优秀的反向代理功能，以及负载均衡、内容缓冲、安全防护等高级特性，所以最常见的用法是部署在网站架构的最前端，作为流量的总入口，提高系统的整体稳定性和可靠性。

OpenResty 内嵌方便快捷的 Lua 脚本，完全能够取代 PHP、Python、Ruby 来编写应用服务，把业务逻辑跑在高性能的 Nginx 里，去掉不必要的中间环节直接操作 Redis、MySQL 等数据库，减少内部的网络消耗，节约系统资源。

如果系统里已经存在了大量其他语言实现的应用服务，改造起来有困难甚至不可行，OpenResty 也可以充当 API Gateway，以 RESTful 接口为基础聚合整理各种后端服务，并增加监控、缓存、权限控制等功能，改善系统的运行效率。

总之，OpenResty 提供了多种多样的功能，灵活可适配，我们总能够在新系统或旧系统中找到它的合适位置，发挥出它的应有价值。

1.13 总结

OpenResty 以 Nginx 为基础，集成众多设计精良的模块和工具，搭建出了完善易用的服务器开发环境，可以轻松地开发出支持超高并发的 Web 应用和动态网关，正在被越来越多的个人和公司学习和使用。

OpenResty 的核心是 Nginx + ngx_lua/stream_lua，此外还有大量的外围模块和辅助工具，提供了非常丰富的功能，让我们能够使用简单的 Lua 语言作为“胶水”自由拼装组合，操纵复杂的 Web 处理流程。

虽然已经诞生了近十年的时间，但 OpenResty 仍然算是 Web 应用开发的“新生力量”，不过凭借高性能、高扩展性和高易用性的特点，OpenResty 的前景必然是“一片光明”，值得我们花费时间和精力深入研究。

第 2 章

Nginx平台

Nginx[①]的首个公开版发布于 2004 年，相对于 Apache、Lighttpd、Jetty、Tomcat 等服务器“前辈”可以说是真正的“后起之秀”。它能够被 OpenResty 选定为核心运行组件，作为基础运行平台，必然有着不同于其他服务器的独到之处。

本章将简要介绍 Nginx 的特点和各种应用服务的配置方法，这是使用 OpenResty 前必备的基本知识。

2.1 简介

Nginx 是一个高性能、高稳定的轻量级 HTTP、TCP、UDP 和反向代理服务器。它运行效率高，资源消耗低，不需要很高的硬件配置就可以轻松地处理上万的并发请求，是当今 Web 服务器中的佼佼者，被国内外许多知名网站所采用。

Nginx 最突出的特点是卓越的性能。它采用事件驱动，不使用传统的进程或线程服务器模型，没有进程或线程切换时的成本，并且有针对性地对操作系统进行了特别优化，能够无阻塞地处理 10K 乃至 100K 的海量连接。

Nginx 的另一大特点是高度的稳定性。Nginx 内部结构设计非常精妙，内存池避免了常见的资源泄漏，模块化的架构使得各个功能模块完全解耦，消除了相互间可能造成的不良影响，而独特的进程池机制则实现了自我监控和管理，保证即使服务发生严重错误也可以快速恢复。在实际应用中，Nginx 服务器一经启动，就可以稳定地运行数天甚至数月之久。

在高性能和高稳定之外，Nginx 还能够运行在多种操作系统上，安装和配置都很容易，

---

① Nginx 的正确发音是“engine eks”，不过也可以像 UNIX/Linux 那样称它为“engine ks”。

可以灵活组合数量庞大的功能模块，实现策略限速/分流、负载均衡、安全防护、定制日志、平滑升级、热部署等许多重要的运维功能。

正是因为 Nginx 有着如此之多的优点，它才能够在与 Apache、Lighttpd 等的“竞争”中脱颖而出，获得 OpenResty 的“青睐”，成为了 OpenResty 的核心运行平台。

2.2 进程模型

Nginx 采用了 master/workers 进程池机制，这是它能够稳定运行的保证，也是理解 OpenResty 运行机制的要点。

通常情况下，Nginx 会启动一个 master 进程和多个 worker 进程。master 进程又称监控进程，它并不处理具体的 TCP/HTTP 请求，只负责管理和监控 worker 进程。多个 worker 进程从属于 master 进程，构成一个“池”，真正对外提供 Web 服务，执行主要的业务逻辑，可以充分利用多核 CPU 高效率地处理 HTTP/TCP 请求。

Nginx 的进程模型如图 2-1 所示：

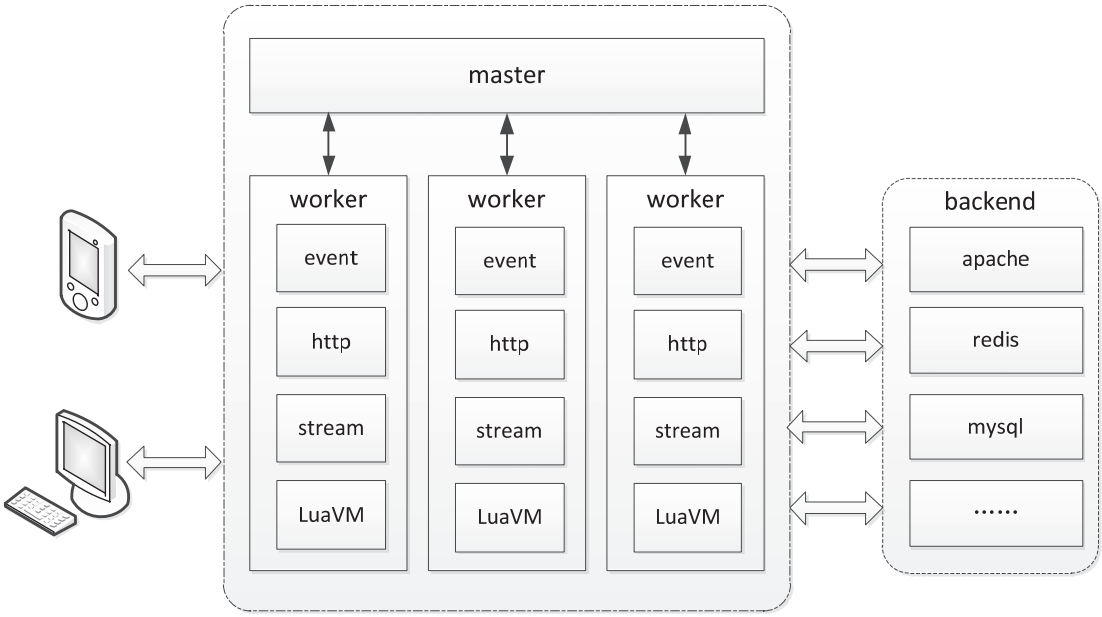

图 2-1 Nginx 的进程模型

使用 Linux 的 ps 命令配合 grep 可以看到 OpenResty 启动的 Nginx 进程，也可以验证 OpenResty 是否正常运行：

```
ps aux|grep nginx
root      16985 nginx: master process /usr/local/openresty/bin/openresty
nobody    16986 nginx: worker process
```

从 ps 的输出我们可以看到当前共有两个 Nginx 进程，其中进程号为 16985 的是 master 进程，而 16986 号进程则是 worker 进程。

2.3 配置文件

在 OpenResty 里，Nginx 配置文件不仅定义了服务的基本运行参数（进程数量、运行日志、优化调整等），还定义了 Web 服务的接口和功能实现，只有熟悉配置文件才能维护好 OpenResty。

Nginx 的配置文件使用了自定义的一套语法，规则严谨而简洁，完全可以把它理解成一个小型的编程语言，要点简略叙述如下：

- 与 Shell/Perl 相同，使用#开始一个注释行；
- 使用单引号或者双引号来定义字符串，允许用“\”转义字符；
- 使用$*var* 可以引用预定义的一些变量；
- 配置指令以分号结束，可以接受多个参数，用空白字符分隔；
- 配置块（block）是特殊的配置指令，它有一个{}参数且无须分号结束，{}里面可以书写多个配置指令，配置块也允许嵌套；
- 使用“include”指令可以包含其他配置文件，支持“*”通配符；
- 不能识别或错误的配置指令会导致 Nginx 启动失败。

下面列出 OpenResty 自带的配置文件片段，部分较重要的配置指令用黑体表示：

```
worker_processes  1;                          #设置 worker 进程的数量为 1

events {                                      #events 块，使用的事件机制
  worker_connections  1024;                   #单个 worker 的最大连接数
}                                             #events 块结束

http {                                        #定义 HTTP 服务
  server {                                    #server 块，定义一个 Web 服务
    listen        80;                         #服务使用的是 80 端口
    server_name   localhost;                  #HTTP 服务对应的域名

    location / {                              #location 块，定义匹配的 URI
        ...
    }                                         #location 块结束
```

```
    }                                                    #server 块结束
}                                                        #http 块结束
```

这个配置文件片段的第一行是配置指令 worker_processes，要求 Nginx 启动一个 worker 进程，我们在实际应用时应当根据 CPU 数量适当调整，以最大化 Nginx 的性能。

events 块里面只有一个 worker_connections 指令，确定每个 worker 进程可以处理的最大连接数，它与 worker_processes 指令共同确定了 Nginx 的服务能力，也就是能够支持的最大并发连接数（即 worker_processes*worker_connections）。

http 块是我们在开发 OpenResty 应用时最需要关注的，它定义了对外提供的 Web 服务和功能接口，示例里是一个监听标准 80 端口的服务，详细解说可参见 2.5 节。

2.4 变量

“变量”是 Nginx 内部保存的运行时 HTTP/TCP 请求相关数据，可以在编写配置文件时任意引用，使得编写 Nginx 配置文件更像是编写程序（但注意不要与编程语言里的变量概念混淆，两者是完全不同的）。

在配置文件里使用变量需要以“$”开头，例如$request_method、$args、$uri 等（这与 Shell 和 Perl 是一样的）。变量的用法很多，例如记录访问日志，设置反向代理的参数，或者传递给 Lua 程序获取各种运行时信息。

以下列举了几个在 HTTP 服务里较常见的变量：

- $uri ：当前请求的 URI，但不含“？”后的参数；
- $args ：当前请求的参数，即“？”后的字符串；
- $arg_*xxx* ：当前请求里的某个参数，“arg_”后是参数的名字；
- $http_*xxx* ：当前请求里的 *xxx* 头部对应的值；
- $sent_http_*xxx* ：返回给客户端的响应头部对应的值；
- $remote_addr ：客户端 IP 地址。

如果执行下面的 curl 命令：

```
curl 'http://localhost/index.html?a=1&b=2' -H 'hello: world'
```

那么这些变量的值就是：

```
$uri                = /index.html
$args               = a=1&b=2
$arg_a              = 1
```

```
$arg_b                = 2
$http_hello           = world
$sent_http_server     = openresty/1.13.6.2
$remote_addr          = 127.0.0.1
```

Nginx 内置的变量非常多，详细的列表可以参考 Nginx 官网文档。此外，Nginx 也允许使用指令自定义变量，最常用的就是 set，例如：

```
set $max_size        10000;              #定义变量$max_size="10000"
```

2.5 HTTP 服务

配置 HTTP 相关的功能需要使用指令 http{}，定义 OpenResty 里对外提供的 HTTP 服务，通常的形式是：

```
http {                                   #http 块开始，所有的 HTTP 相关功能

    server {                             #server 块，第一个 Web 服务
        listen 80;                       #监听 80 端口

        location uri {                   #location 块，需指定 URI
            ...                          #定义访问此 URI 时的具体行为
        }                                #location 块结束
    }                                    #server 块结束

    server {                             #server 块，第二个 Web 服务
        listen xxx;                      #监听 xxx 端口
        ...                              #其他 location 定义
    }                                    #server 块结束
}                                        #http 块结束
```

由于 http 块内容太多，如果都写在一个文件里可能会造成配置文件过度庞大，难以维护。在实践中我们通常把 server、location 等配置分离到单独的文件，再利用 include 指令包含进来，这样就可以很好地降低配置文件的复杂度。

使用 include 后 http 块就简化成了：

```
http {                                   #http 配置块开始，所有的 HTTP 相关功能

    include common.conf                  #基本的 HTTP 配置文件，配置通用参数

    include servers/*.conf               #包含 servers 目录下所有 Web 服务配置文件
}                                        #http 配置块结束
```

2.5.1 server 配置

server 指令在 http 块内定义一个 Web 服务，它必须是一个配置块，在块内部再用其他指令来确定 Web 服务的端口、域名、URI 处理等更多细节。

listen *port;*

listen 指令使用 port 参数设置 Web 服务监听的端口，默认是 80。此外还可以添加其他很多参数，例如 IP 地址、SSL、HTTP/2 支持等。

server_name *name* ...;

server_name 指令设置 Web 服务的域名，允许使用“*”通配符或“~”开头的正则表达式。例如“www.openresty.org”“*.openresty.org”。当 OpenResty 处理请求时将会检查 HTTP 头部的 Host 字段，只有与 server_name 匹配的 server 块才会真正提供服务。

对于我们自己的开发研究来说，可以直接使用 localhost 或者简单的通配符*.*，用类似“curl http://localhost/...”这样的命令就能够访问 OpenResty。

2.5.2 location 配置

location 指令定义 Web 服务的接口（相当于 RESTful 里的 API），也就是 URI，它是 OpenResty 处理的入口，决定了请求应该如何处理。

location 是一个配置块，但语法稍多一些，除{}外还有其他的参数：

location [= | ~ | ~* | ^~] *uri* { ... }

location 使用 uri 参数匹配 HTTP 请求里的 URI，默认是前缀匹配，也支持正则表达式，uri 参数前可以使用特殊标记进一步限定匹配：

- = ：URI 必须完全匹配；
- ~ ：大小写敏感匹配；
- ~* ：大小写不敏感匹配；
- ^~ ：前缀匹配，匹配 URI 的前半部分即可。

在 server 块里可以配置任意数量的 location 块，定义 Web 服务接口。Nginx 对 location 的顺序没有特殊要求，并不是按照配置文件里的顺序逐个查找匹配，而是对所有可能的匹配进行排序，查找最佳匹配的 location。

不同的 location 里可以有不同的处理方式，灵活设置 location 能够让 OpenResty

配置清晰明了，易于维护。比如，我们可以在一个 location 里存放静态 html 文件，在另一个 location 里存放图片文件，其他的 location 则执行 Lua 程序访问 MySQL 数据库处理动态业务，这些 location 互不干扰，修改其中的一个不会影响其他的正常运行。例如：

```
location =    /502.html              #只处理/502.html 这一个文件
location      /item/                 #前缀匹配/item/*
location ^~   /image/                #显式前缀匹配/image/*
location ~    /articles/(\d+)$       #正则匹配/articles/*
location ~    /api/(\w+)             #定义 RESTful 接口
location      /                      #匹配任意的 URI
```

需要注意最后一个“/”，根据前缀匹配规则，它能够匹配任意的 URI，所以可以把它作为一个“黑洞”，处理所有其他 location 不能处理的请求（例如返回 404）。

如果 location 配置很多，我们同样可以用 include 的方式来简化配置。

2.6 TCP/UDP 服务

配置 TCP/UDP 相关的功能需要使用指令 stream{}，形式与 http 块非常类似，例如：

```
stream {                                  #stream 块开始，TCP/UDP 相关功能

    server {                              #server 块，第一个 Web 服务
        listen 53;                        #监听 TCP 53 端口
        ...
    }                                     #server 块结束

    server {                              #server 块，第二个 Web 服务
        listen 520 udp;                   #监听 UDP 520 端口
        ...
    }                                     #server 块结束
}                                         #stream 块结束
```

定义 TCP/UDP 服务同样需要使用 server 指令，然后再用 listen 指令确定服务使用的具体端口号。但因为 TCP/UDP 协议里没有“Host”“URI”的概念，所以 server 块里不能使用 server_name 和 location 指令，这是与 HTTP 服务明显不同的地方，需要注意。

2.7 反向代理

反向代理（Reverse Proxy）是现今网络中一种非常重要的技术，它位于客户端和真正的服务器（即所谓的后端）之间，接受客户端的请求并转发给后端，然后把后端的处理结果返

回给客户端。从客户端的角度来看，访问反向代理和真正的后端服务器两者没有任何区别。

反向代理的网络结构如图 2-2 所示：

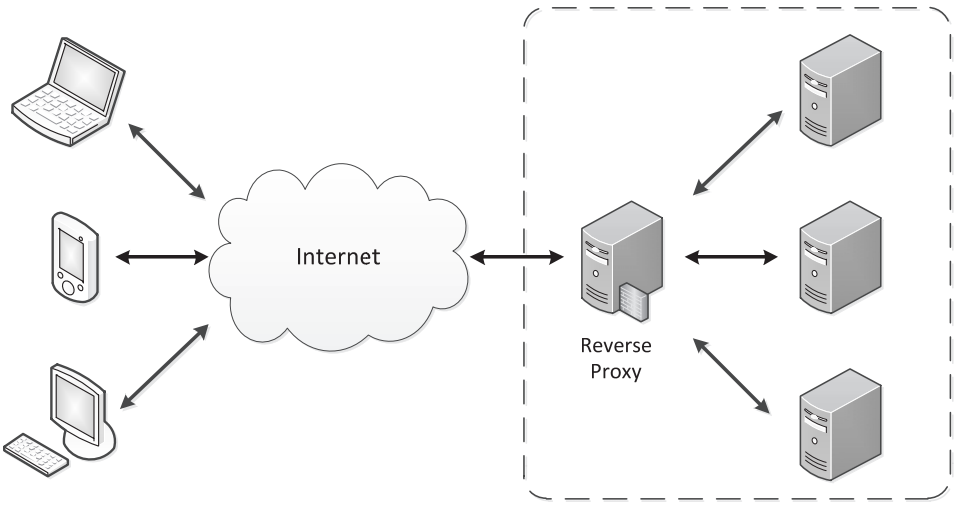

图 2-2 反向代理的网络结构

由于反向代理在客户端和服务器之间加入了中间层，可以执行复杂的逻辑，所以它有很多的用途，例如：

- 负载均衡 ：最常用的功能，均衡多个后端服务器的访问请求，实现服务集群化；
- 安全防护 ：使用 WAF 等工具防御网络入侵，保护后端服务器；
- 内容缓存 ：缓存上下行数据，减轻后端服务器的压力；
- 数据加密 ：加密验证外部通信过程，而内部服务器之间没有加密成本。

Nginx 提供了优秀的反向代理功能，不仅支持 HTTP 反向代理，也支持 TCP/UDP 反向代理，非常适合用在网络的核心位置担当“中流砥柱”的重任。[①]

2.7.1 上游集群

upstream 块定义了在反向代理时需要访问的后端服务器集群和负载均衡策略(在 Nginx 里术语“upstream”代替了“backend”)，可以在 http{}或 stream{}里配置。

① 著名的 LVS（Linux Virtual Server）也是一种反向代理，但与 Nginx 不同的是它集成在 Linux 内核里，工作在网络四层以下，而且只能用于负载均衡。

upstream 块的基本形式是：

```
upstream backend {                          #upstream 需要有一个名字
    least_conn;                             #负载均衡策略
    server 127.0.0.1:80;                    #一台上游服务器
    server ... weight=3;                    #可以指定多台上游服务器
    server ... backup;                      #备份用的上游服务器

    keepalive 32;                           #使用连接池，长连接复用

}
```

upstream 块的配置比较简单，主要使用 server 指令列出上游的服务器域名或 IP 地址，还可以用 weight/max_fails/down/backup 等附加参数来进一步描述这些服务器的状态。least_conn 指令确定了集群里服务器的负载均衡策略，类似的还有 hash、ip_hash 等。如果不给出明确的策略，Nginx 就使用简单的加权轮询（round robin）。

2.7.2 代理转发

在使用 upstream 配置了上游集群后，我们需要在 location（http）或 server stream）里用“proxy_pass”等指令把客户端的请求转发到后端，由 Nginx 根据负载均衡算法选择一台恰当的服务器提供服务，例如：

```
location /passto {                          #一个转发的 location
    proxy_set_header Host $host;            #使用变量转发原始请求的 host 头部
    proxy_pass http://backend;              #转发到 upstream 块定义的服务器集群
}
```

Nginx 代理转发相关的指令比较多，用来应对各种复杂的场景，proxy_pass 只是其中最基本的一个（转发 HTTP/HTTPS 服务），其他的还有 fastcgi_pass、memcached_pass 等，篇幅所限这里就不详细解说了，读者可参考 Nginx 文档。

2.8 运行日志

日志是 Web 服务器非常重要的数字资产，它记录了服务器运行期间的各种信息，可用于数据分析或者排查故障。

Nginx 的运行日志分为两种：记录 HTTP/TCP 访问请求的 access_log 和记录服务器各种错误信息的 error_log。

2.8.1 访问日志

访问日志保存了所有连接到服务器的客户端访问记录，在访问日志里可以记录每次请求的 IP 地址、URI、连接时间、收发字节数等许多信息。大多数网站会定期收集访问日志，然后使用大数据平台进行加工处理，进而调整优化服务。

在 Nginx 里需要用两个指令来设定访问日志：

```
log_format name  format_string;

access_log path [format [buffer=size] [flush=time]];
```

log_format 指令定义日志的格式，格式字符串里可以使用变量（见 2.4 节）来任意记录所需的信息，之后就可以用 access_log 指令决定日志的存储位置和格式。为了优化磁盘读写，可以设置 buffer 和 flush 选项，指定写磁盘的缓冲区大小和刷新时间。

例如，下面的配置使用了 8KB 的缓存，每 1 秒刷新一次，使用格式 main：

```
log_format  main  '$remote_addr ... ';
access_log /var/logs/openresty/access.log main buffer=8k flush=1s;
```

2.8.2 错误日志

当 Nginx 运行发生异常时（例如拒绝访问、缓冲区不足、后端不可用等）就会记录错误日志。错误日志的格式不能自定义，存放位置由 error_log 指令确定：

```
error_log file level;
```

默认的日志存放位置是安装目录下的 logs/error.log。我们也可以用参数 file 改为其他路径。第二个参数 level 是日志的允许输出级别，取值是“debug|info|notice|warn|error|crit|alert|emerg”，只有高于这个级别的日志才会记录下来，默认值是 error。

对于我们开发 OpenResty 应用来说 error_log 非常重要，优化系统、排查问题的时候首先要做的事情就是查看 error_log。

2.9 总结

Nginx 是 OpenResty 最基本的核心组成部分，本章首先介绍了 Nginx 的特点和进程模型，然后简要阐述了 Nginx 的配置文件格式和各种应用服务的配置方法。

Nginx 是一个高性能高稳定的服务器软件，运行效率高，资源消耗低，可以轻松地处理

上万甚至百万的并发请求。模块化的架构让它具有良好的扩展性，可以任意组合功能模块实现策略限速、负载均衡、安全防护等功能。OpenResty 选择 Nginx 作为运行平台，正是“站在了巨人的肩膀上”。

Nginx 采用独特的 master/workers 进程池机制。master 进程管理和监控 worker 进程，worker 进程真正对外提供 Web 服务。这种机制保证了服务的稳定运行，也能够充分利用多核心的 CPU，轻易扩充服务能力。

Nginx 使用配置文件定义对外提供的服务，支持 HTTP/TCP/UDP 等多种通信协议，语法很类似其他的编程语言。HTTP 服务需要使用 http{}，里面再使用 server/listen/location 等指令定义服务的具体细节。TCP/UDP 服务使用 stream{}，与 HTTP 服务不同的是没有 location 概念。当 Nginx 用作反向代理时需要使用指令 upstream 定义后端集群和负载均衡策略，再配合 proxy_pass、fastcgi_pass 等指令实现高效的代理转发。

Nginx 提供了 access_log 和 error_log 两种运行日志，可以灵活配置格式和存放位置，方便我们进行数据分析、性能优化或者故障排查。

Nginx 的功能非常强大，本章的内容仅仅是“冰山之一角”，篇幅所限不可能完整介绍所有的配置选项，有的重要功能例如缓存、重定向、访问控制、CPU 绑定等都没有涉及，请读者及时参考 restydoc、Nginx 官网或者其他资料。