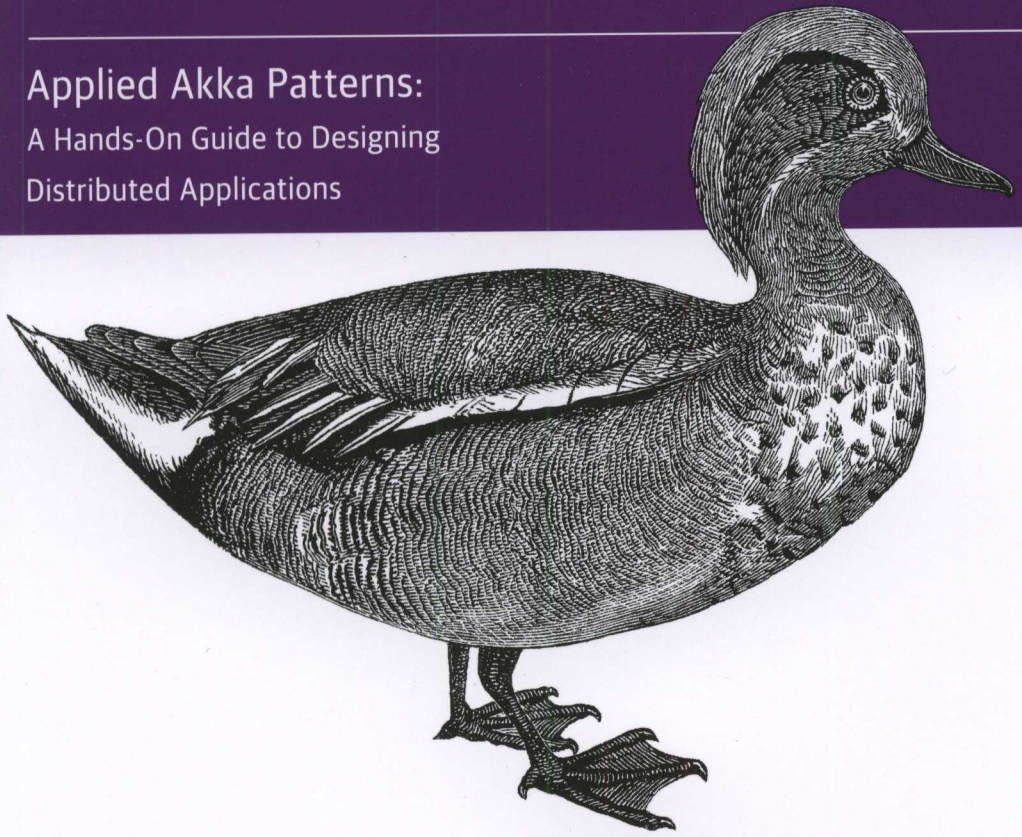
O'REILLY®

Broadview®
www.broadview.com.cn

Akka应用模式：

分布式应用程序设计实践指南

Applied Akka Patterns:
A Hands-On Guide to Designing
Distributed Applications



[美] Michael Nash [加] Wade Waldron 著
虞航仲 译

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

Akka应用模式：分布式应用程序设计实践指南

Applied Akka Patterns: A Hands-On Guide to Designing Distributed Applications

[美] Michael Nash
[加] Wade Waldron 著
虞航仲 译

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

内 容 简 介

本书一开始介绍了 Actor 模型，因为该模型比其他模型更能反映现实世界。另外，本书介绍了 Actor 模型的一个实现框架 Akka 以及它的工具，而后讨论了在充分利用 actor 架构的基础上使用 Akka 框架来设计软件系统的方法，以及使用它来开发并发性和分布式应用程序的方法。本书还介绍了领域驱动设计（DDD），指导通过结合 Akka、Actor 模型和 DDD 构建强大的、高可扩展的、高可维护的系统。

遵循并灵活运用本书介绍的思路和方法，一定能创建出集可伸缩性、弹性、灵活性、长期易于维护性等优点于一身的应用程序和系统。

版权贸易合同登记号 图字：01-2017-3966

图书在版编目（CIP）数据

Akka 应用模式：分布式应用程序设计实践指南 /（美）迈克尔·纳什（Michael Nash），（加）韦德·沃尔德龙（Wade Waldron）著；虞航仲译 . —北京：电子工业出版社，2017.10

书名原文：Applied Akka Patterns: A Hands-On Guide to Designing Distributed Applications

ISBN 978-7-121-32529-8

Ⅰ . ① A… Ⅱ . ①迈… ②韦… ③虞… Ⅲ . ① JAVA 语言－程序设计 Ⅳ . ① TP312.8

中国版本图书馆 CIP 数据核字 (2017) 第 200914 号

策划编辑：孙奇俏

责任编辑：张春雨

封面设计：Karen Montgomery 张健

印　　刷：三河市良远印务有限公司

装　　订：三河市良远印务有限公司

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：11.5 字数：255 千字

版　　次：2017 年 10 月第 1 版

印　　次：2017 年 10 月第 1 次印刷

定　　价：65.00 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。

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始,O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来,而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者,O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”;创建第一个商业网站(GNN);组织了影响深远的开放源代码峰会,以至于开源软件运动以此命名;创立了Make杂志,从而成为DIY革命的主要先锋;公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖,共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择,O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程,每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列(真希望当初我也想到了)非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人,他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了:‘如果你在路上遇到岔路口,走小路(岔路)。’回顾过去Tim似乎每一次都选择了小路,而且有几次都是一闪即逝的机会,尽管大路也不错。”

——Linux Journal

推荐序

Akka是在Scala语言中实现的一个明星级产品，被广泛地应用在构建基础框架和业务架构中，它把Erlang中被工业界证实为高可用、高容错的Actor模型成功应用到JVM之中，并产生了巨大的影响。我所在的挖财公司，其生产环境中也有一些基于Akka的系统，经过长时间的观察，这些系统运行得都很稳定。

Actor模型本身是一种非常出色的编程模型，但要充分发挥它的优势，还需要开发者对它的各种细节及使用模式非常熟悉。例如，在常见的模式中，有一种“隔离阻塞”模式经常会让初学者很头疼。

actor系统的核心在于各个actor之间的通信，但基于JVM的Akka，其底层仍是基于线程池来工作的，actor之间的通信也是靠线程调度来实现的。因此，当逻辑中含有一些阻塞操作时，需要额外注意并反复确认这些阻塞操作使用的线程与actor通信中使用的线程是否相同，并发请求中多了这些阻塞操作是否会导致所有的线程都被“block”住，进而导致actor之间的通信受到影响。

我自己曾经遇到过一个因为未正确设置隔离阻塞操作而造成严重bug的案例：有一个项目在我自己的笔记本电脑上运行时不会发生任何异常，但当部署到测试环境中运行时却发生了导致整个actor系统不可用的问题。

深入系统调试后发现，系统里有一个作为Router角色的actor，其背后竟然有40个实例在执行同一个会发生阻塞的远程调用（在这种情况下系统至少需要40个可用的线程）。我们知道actor系统底层调度的线程池大小是由并发因子和CPU核心数决定的，我的个人笔记本的CPU核心数是8，测试环境中机器的CPU核心数只有2，两个不同环境的线程池大小相差很大。当设置的并发因子数为10时，系统在我个人笔记本电脑上的线程池大小是80，大于40，因此那40个实例执行远程调用的同时还有空闲的线程可用于actor

之间的通信，系统可以正常运行；而测试环境中的线程数只有 20，是少于 40 的，这使得所有线程都被那 40 个实例的远程调用所占用，系统中没有可用于 actor 之间通信的多余线程，从而导致系统出现了不可用的情况。

除了常见模式，还有一个对于开发者和架构师而言都非常重要的事情——如何集成已有的非 Actor 模型的框架、库或服务。

例如，已有的系统都是基于 Spring 生态的，那么 Akka 应用是否也应该基于 Spring 生态？或者，HTTP 服务究竟该选用运维熟悉的 Tomcat，还是 Akka 自带的 Akka HTTP 呢？这些都是需要技术人员根据实际工程认真思考的问题，不是单单权衡技术利弊就可以做出选择的，可见 Akka 中值得我们去探索的问题还有很多。

这本书对于 Akka 中各个技术点的阐述非常简练，作者在不到 200 页的篇幅中介绍了从 Actor 模型到 Akka 组件，再到领域驱动设计等许多知识点，为读者提供了正确应用 Akka 框架进行分布式设计开发的思路。另外，这本书中有很多干货，内容非常务实，对于应用 Akka 框架进行实际项目开发的人有很大的帮助。

强烈推荐从事 Akka 分布式开发的的技术人员都能仔细阅读这本书，希望有更多的读者能够通过这本书了解 Akka 这个非常强大的工具。

挖财中间件团队负责人，王宏江

2017 年 8 月

译者序

我在大学期间开始接触并使用 Scala，当时就发现了这门语言的强大。正如其名字所表达的那样，Scala 是一门可扩展的编程语言，融合了函数式编程和面向对象编程的特点，支持并发，让异步编程变得很自然，同时表达能力也特别强。虽然 Scala 的学习成本会比较高，尤其和 Golang、Python 这类语言相比，但是这门语言还是非常实用的，对提高开发效率很有帮助。而且 Scala 的创始人 Martin Odersky 说过："Scala 现在是为聪明人创造的，以后也是为聪明人服务的。" Scala 相信程序员的聪明才智，让程序员自行选择合适的结构，以应对千变万化的任务需求，这一点是 Scala 做得很不错的地方。

编写正确的具有容错性和可扩展性的分布式、高并发程序非常困难，主要是因为我们使用了错误的工具和错误的抽象层次，而 Akka 的出现改变了这种情况。Akka 是参照 Apache 2 许可证（一种公认的开源许可证）发布的开源项目，通过使用 Actor 模型来提高抽象的级别，并提供了一个更好的平台来构建可扩展的、有弹性的、响应式的应用程序，详细信息可参阅 *The Reactive Manifesto*。对于容错，Akka 采用"让它崩溃（Let It Crash）"的模式，这种模式可以帮助构建可自我修复和永不停止的软件系统。其中 actor 还提供了透明的分布式抽象，以及真正的可扩展与高容错应用的基础设施。

毕业后，我在腾讯微信团队的后台架构部从事软件开发工作，作为腾讯的战略级产品，微信平均每天面临亿级的在线用户。面对这种用户规模的挑战，基本每一行代码都需要考虑高并发和分布式的场景，大道至简的思想基本贯彻在整个技术产品线上。例如，大系统小做，让一切可扩展；剥离复杂，让剩下的更简单；在容灾之前面向最坏情况思考，防止雪崩；精细监控，迅速响应等。微信团队内部的很多技术点也和 Actor 模型很像，比如微信最核心的消息模型和 Actor 的邮件模型就很有渊源。消息被发出后，会先在后台临时存储，为了使接收者能更快接收到消息，系统会推送通知给接收者，最后客户端主动到服务器端拉取消息。当然整个微信架构就是微服务的架构，每一个请求后面可能

会涉及几百个服务。如何扩展、如何高容错、如何弹性，这些基本是每天都会遇到的挑战，并且也都是设计和开发系统的时候需要考虑的事情。

后来，我离开腾讯，在猎豹移动的广告系统架构部以及新闻团队从事系统架构开发工作，同样涉及微服务架构，每天都要考虑到这些分布式系统可能遇到的并且需要处理的问题。尤其广告系统涉及金钱，因而需要严格保证其高可用性和一致性。

现在，我在创业公司担任技术负责人，同时负责公司内部多个系统的架构工作，也需要时时刻刻考虑和处理高并发、高容错等分布式问题，同样用到了高并发的分布式微服务架构。

本书原名是 *Applied Akka Patterns : A Hands-On Guide to Designing Distributed Applications*，书中介绍了一些很好的分布式系统的设计原则，而且也介绍了 Actor 模型和 Akka 工具包，对于使用 JVM 体系结构的开发者来说是非常值得一看的，因为通过本书能很快地学习并掌握一个强大的工具，在工作中提高生产效率。对于那些使用非 JVM 体系结构的开发者来说，通过这本书能了解到一个更强大的工具，也是极有帮助的。同时，本书介绍的很多指导原则对于分布式系统设计有很大的借鉴意义，可以避免让自己陷入困境。这本书是一个起点，帮助我们发现新大陆，但绝不会是我们的终点，开拓这块新大陆还需要自己不断努力。其实 Scala 本身就是一门很强大的语言，最近也因为 Spark、Kafka 等项目在国内掀起了一波关注热潮。Akka 也很优秀，以至于被 Lightbend 收购，并直接用 Akka 的 actor 替代了 Scala 本身的 actor。总之，本书中介绍的内容都是非常值得探索和学习的。

决定翻译这本书，不仅因为我参与和主导了不少分布式的项目，也因为对分布式系统设计和开发的热爱，以及对 Scala 语言本身的喜欢。我抱着把 Actor 模型以及 Akka 传递给中国的工程师并让更多人能接触、了解它们的态度来尝试翻译这本书。虽然它们在国内还不算太火，但是在国外已经非常受欢迎。

这本书虽然只有 160 多页，但是翻译过程还是比较辛苦的，很多地方的意思都比较隐晦，用中文直译会很有挑战。所以我平时会留意大家在社区里讨论的内容，参考社区里的资料，在遇到一些和自己项目经验不太一致或者比较不确定的地方，也会尝试去查看 Akka 的源码来尽量保证自己的理解无误，避免误导读者。

在本书的翻译工作结束之际，我首先要感谢博云科技的 CTO 李亚琼老师，他的引荐让我获得了翻译这本书的机会。还要感谢孙国立、曾杰瑜、付冉、刘岸，他们在百忙中帮我审阅了本书翻译稿的大部分章节，并针对涉及的专业概念提出不少修改意见。最后要感谢本书的策划编辑孙奇俏，她为本书的编辑和校对做了大量细致的工作。

翻译过程中虽力求理解作者意图，把握全文，但是难免会发生错误，若广大读者发现错误，我在此深表歉意。欢迎广大读者及时与我和出版社联系，提交勘误，方便后续读者更好地阅读。如果有任何好的想法和建议，也欢迎和我邮件沟通，我的邮箱地址是hangzhong.yu@gmail.com。

虞航仲

2017 年 8 月于北京

目录

前言

响应式应用开发是软件开发的新前沿。随着联网设备的普及，数据量也在增加。以前的单线程批处理数据的旧技术根本无法满足这个新领域所提出的需求。大数据的概念已经兴起，我们需要新的工具和新的技术来应对它。

通常，解决现有问题的灵感并不是来自现有的技术，而是来自过去的经验。许多现今用于处理大数据的新工具实际上是基于旧的 actor 的概念而产生的。actor 是构建 Akka 的关键概念，但其根源追溯起来应该属于过去。actor 不是一个新概念，相反，它是一个被重新关注的旧概念。

当开始探索 Akka、actor、streams 和其他与之相关的技术时，我们将从现实世界的角度来看待它们：如何在一系列项目中安排一组人，同时优化他们的可用时间及技能？这是一个复杂的问题，并不是只用一个下午就可以解决的。但这又是一个有趣的问题，为深度探索提供了很大的空间。这也是大多数软件开发人员在职业生涯中的某个时刻一定会遇到的问题。在对 Akka 进行探索的过程中，我们将回顾这个问题。

在解决问题之前，我们必须先了解可用的工具，还需要了解这些工具为什么存在，以及它们可以解决什么样的问题。我们需要知道 Akka 的起源及其在 Actor 模型中的根源。我们需要一套指导原则将应用程序拼接在一起，这套原则会在探索域驱动设计（DDD）的过程中被发现。有了这些基础，便可以开始使用 Akka 提供的所有工具来构建域了。我们可以探索简单的 actor 的使用方法以及它如何与流关联，可以让系统分布在多个节点上，使其具有更好的容错性、可用性及可扩展性。

首先，我们需要知道这一切的根源在哪里。