计算机科学先进技术译丛

MANNING

Akka实战

Akka IN ACTION

[美] 雷蒙德·罗斯腾伯格（Raymond Roestenburg）
[美] 罗勃·贝克尔（Rob Bakker）
[美] 罗勃·威廉姆斯（Rob Williams）
著

程继洪　肖　川　译

从实战角度介绍了Akka工具及其重要模块

入门+手册的内容安排，代码测试贯穿全书

实例丰富、内容实用，上手容易、起点较低

在线提供源代码

机械工业出版社
CHINA MACHINE PRESS

计算机科学先进技术译丛

Akka 实战

[美] 雷蒙德·罗斯腾伯格（Raymond Roestenburg）
[美] 罗勃·贝克尔（Rob Bakker）
[美] 罗勃·威廉姆斯（Rob Williams）
著

程继洪 肖 川 译

机械工业出版社

Akka 是 Java 虚拟机 JVM 平台上构建高并发、分布式和容错应用的工具包和运行库，同时提供了 Scala 和 Java 的开发接口。本书主要介绍了 Akka 的 Actor 开发模型、并行编程、消息传递、路由功能、集群、持久化等内容，还介绍了 Akka 的配置、系统集成和性能分析与度量等有关知识，全面介绍了 Akka 的主要功能，并给出了丰富的实例。

本书可作为程序员、软件工程师和架构师关于开发分布式并行应用的参考书，也可以作为高等院校分布式并行开发的教材，还可以作为对分布式并行开发感兴趣的读者的入门参考书。

北京市版权局著作权合同登记　图字：01-2016-8236 号。

图书在版编目(CIP)数据

Akka 实战 /（美）雷蒙德·罗斯腾伯格(Raymond Roestenburg)，（美）罗勃·贝克尔(Rob Bakker)，（美）罗勃·威廉姆斯(Rob Williams)著；程继洪，肖川译．—北京：机械工业出版社，2018.11
（计算机科学先进技术译丛）
书名原文：Akka in Action
ISBN 978-7-111-61342-8

Ⅰ.①A…　Ⅱ.①雷…　②罗…　③罗…④程…　⑤肖…　Ⅲ.①JAVA 语言-程序设计　Ⅳ.①TP312.8

中国版本图书馆 CIP 数据核字(2018)第 258729 号

机械工业出版社(北京市百万庄大街 22 号　邮政编码　100037)
策划编辑：李培培　　责任编辑：李培培
责任校对：张艳霞　　责任印制：张　博

三河市宏达印刷有限公司印刷

2019 年 1 月第 1 版·第 1 次印刷
184 mm×260 mm·21.5 印张·448 千字
0001-3000 册
标准书号：ISBN 978-7-111-61342-8
定价：89.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务
服务咨询热线：010-88361066
读者购书热线：010-68326294
　　010-88379203

网络服务
机 工 官 网：www.cmpbook.com
机 工 官 博：weibo.com/cmp1952
金　书　网：www.golden-book.com
教育服务网：www.cmpedu.com

序

编写好的并行和分布式的应用程序是很难的。当时，我刚刚完成了一个需要许多低层的Java并行编程技术的项目，为了下一个项目，我打算寻找更简单的工具，这将更具有挑战性。

2010年3月，我注意到Dean Wampler的一条推文，使我认识了Akka。

通过观察源代码和构建原型，我们决定使用Akka。其效果立竿见影，这个新的编程工具大大简化了以前项目中遇到的问题。

我和Rob Bakker一起进行了一场新技术冒险，一起用Scala和Akka构建了第一个项目。我们首先尝试寻求Jonas Bonér（Akka的创建者）的帮助，但后来发现我们是Akka的第一批使用者。我们完成了项目，之后还有许多其他的项目，每一次使用Akka的优势都非常明显。

那时候网上的可用信息不多，因此我决定开始撰写有关Akka的博客，也算是对Akka项目的一点贡献。

当让我撰写本书时，我感觉非常吃惊。我询问Rob Bakker是否愿意和我一起撰写，后来我们意识到需要更多的帮助，Rob Williams加入了我们，他曾经做过Java和Akka的项目。

很高兴我们最终完成了这本书，并描述了Akka 2.4.9版所提供的构建分布式和并发应用的一系列工具。读者也给了我们很多的反馈，我们十分高兴。

在没有使用Akka之前，我们一致认为编写基于JVM的分布式并发应用需要更好、更简单的工具。我们希望通过本书能够让读者相信，Akka就是这样的工具。

雷蒙德·罗斯腾伯格

致谢

写作本书是一个漫长的过程，在这期间许多人给予了我们帮助，对他们的付出表示感谢。对于那些购买 MEAP 版本的读者，非常感谢你们的反馈，这些反馈使本书有了很大提高，并对这些年来你们的耐心表示感谢。

特别感谢 Akka 核心项目组成员，特别是 Jonas Bonér、Viktor Klang、Roland Kuhn、Patrik Nordwall、Björn Antonsson、Endre Varga、和 Konrad Malawski，他们给我们提供了灵感和珍贵的资料。

我们还要感谢荷兰的 Edwin Roestenburg 和 CSC 交通管理部门，他们相信我们可以使用 Akka 开发关键任务项目，并为我们获得 Akka 的初步经验提供了难得的机会。我们还要感谢 Xebia，使得雷蒙德可以投入本书的撰写，并提供了进一步获取 Akka 经验的工作场所。

感谢 Manning 出版公司对我们的信任。这是我们第一次写书，这对于他们来讲是有很大风险的。我们要感谢以下工作优秀的 Manning 员工：Mike Stephens、Jeff Bleiel、Ben Berg、Andy Carroll、Kevin Sullivan、Katie Tennant 和 Dottie Marsico。

感谢 Doug Warren，他对本书的所有章节进行了全面的技术校对。许多审稿人在编写和开发过程中给予了有益的反馈：Andy Hicks、David Griffith、Dušan Kysel、Iain Starks、Jeremy Pierre、Kevin Esler、Mark Janssen、Michael Schleichardt、Richard Jepps、Robin Percy、Ron Di Frango 和 William E. Wheeler。

最后，要感谢成书过程中在生活上支持我们的人。雷蒙德感谢他的妻子 Chanelle，罗勃·威廉姆斯感谢他的妈妈 Gail 和 Laurie。

罗勃·贝克尔

译者序

经过三个多月的不懈努力，终于完成了一部400多页大作的翻译工作。作为高校的一名教师，紧跟技术前沿是十分必要的。大数据应用、数据科学、云计算、机器学习等正在如火如荼地发展着，作为传播知识的教书匠，传播知识是我的职责。

在企业锻炼时，本人也做过不少项目，对于Java底层的并行编程深有感触。虽然Java不断地对并行编程进行改进，但对于普通程序员来说仍然是不小的挑战。分布式应用要考虑许多问题：各结点间的消息通信、负载均衡、结点监督与管理、数据分布、分布式测试等，如果没有一个好用的工具，真是件让人头疼的事！幸好Akka出现了。

Akka作为构建高并发、分布式和容错应用的工具包和运行库，为并行分布式应用的开发带来了便利，并使应用程序的扩展变得容易。Akka已被广泛用于事务处理、服务后端、并发/并行、仿真、批处理、通信、游戏、商业智能、数据挖掘、通用数据处理和复杂事件流处理等领域，体现了Akka框架的强大功能和旺盛生命力，开发并发分布式应用的春天到来了。

本书的翻译忠于原文，尽量准确表达作者的意图，但为了符合中文的表达习惯，对个别语序进行了调整，使读者不至于感觉行文有跳跃感。翻译本书时，可参考的资料很少，网络上虽然也有不少关于Akka的论述，但都是支言片语，不够专业。对于Akka中的术语，都经过了查阅资料、几经斟酌，用符合中文习惯的词汇进行表示，并在括号中给出英文原文，供各位读者参考。对于众所周知的术语则直接采用，没有给出相应的原文。如果各位读者有什么好的意见或建议，请通过jiphoenixsoft@163.com与译者联系。

虽然翻译时斟字酌句，力图符合中文习惯，但也难于符合全部读者的口味，请各位读者谅解。由于译者水平所限，书中错误和疏漏在所难免，望各位读者、专家和同行不吝指正。

最后，感谢我的家人，在繁忙的工作生活中甘愿承担更多的责任，支持我三点一线的译书生活！

程继洪

于烟台南山学院

2017年4月24日

关于本书

本书介绍了 Akka 工具及其重要的模块，集中介绍 Actor 编程模型和支持 Actor 构建并发和分布式应用的模块。代码测试贯穿了全书，这是软件开发每天面临的重要工作。本书所有例子代码用 Scala 编写。

在有了 Actor 编写和测试的基础之后，本书还介绍了使用 Akka 构建实际应用可能遇到的重要方面。

面向读者

本书面向所有想学习利用 Akka 构建应用程序的读者。因为所有例子用 Scala 编写，所以希望读者已经有一定 Scala 语言基础或者在学习过程中乐于学习一些 Scala 知识，还希望读者比较熟悉 Java，因为 Scala 运行于 JVM（Java Virtual Machine，Java 虚拟机）之上。

主要内容

本书分为 17 章。

第 1 章介绍了 Akka 的 Actor，主要介绍了 Actor 编程模型如何解决一系列应用程序难于扩展的问题。

第 2 章介绍了使用 Akka 构建的示例 HTTP 服务，以显示如何快速地在云中获得服务并运行。

第 3 章介绍了使用 ScalaTest 和 Akka-testkit 模块进行 Actor 单元测试。

第 4 章介绍了如何对 Actor 进行监督和监控，构建可靠的、容错的系统。

第 5 章介绍了 Future，以及如何组合 Future 与 Actor。

第 6 章介绍了 Akka-remote 模块，以及如何对分布式 Actor 系统进行单元测试。

第 7 章介绍了如何使用 Typesafe Config 库配置 Akka，以及如何使用这个库配置自己的应用组件。

第 8 章详细介绍了基于 Actor 应用的结构模式。

第 9 章介绍了路由（Router）的使用。路由用于 Actor 之间交换、广播和负载均衡信息。

第 10 章介绍了 Actor 之间发送信息的消息通道，包括 Actor 的点到点和发布-订阅消息通道，以及死信和保证投递通道。

第 11 章介绍了如何使用 FSM 模块构造有限状态机，还介绍了用于异步共享状态的代理。

第 12 章介绍了与其他系统的集成，包括如何通过 Apache Camel 集成各种协议，以及使用 Akka-http 模块构建 HTTP 服务。

第 13 章介绍了 Akka-stream 模块，详细介绍了如何利用 Akka 构建流（streaming）应用程序，以及如何构建处理日志事件的 HTTP 流服务。

第 14 章介绍了 Akka-cluster 模块，详细介绍了如何在网络集群上动态扩展 Actor。

第 15 章介绍了 Akka-persistence 模块，详细介绍了如何使用持久化 Actor 记录和恢复持久化状态，以及如何使用单例集群和分片集群扩展构建集群化的购物车应用。

第 16 章介绍了 Actor 系统的关键性能参数，并提供了如何分析性能问题的提示。

第 17 章对两个即将到来的重要特性做了展望：在编译时检查 Actor 消息的 Akka-typed 模块和 Akka-distributed-data 模块。

代码约定和下载

所有清单和文本中的源代码都以这种固定宽度的字体以便与普通文本相区别。许多代码清单都有说明，以突出重点概念。实例代码可以在出版者网站 www. manning. com/books/akka-in-action 和 https://github. com/RayRoestenburg/akka-in-action 上下载。

软件要求

书中所有代码使用 Scala 编写，所有代码已在 Scala 2. 11. 8 中调试通过。可以在 http://www. scala-lang. org/download/下载 Scala。

确保安装最新版本的 sbt（本书用的是 sbt0. 13. 12）。如果 sbt 版本较旧，则可能会遇到运行问题。可以在 http://www. scala-sbt. org/download. html 找到 sbt。

Akka 2. 4. 9 需要 Java 8 的支持，因此也必须安装 Java 8。Java 8 的下载地址：http://www. oracle. com/technetwork/java/javase/downloads/jdk8-downloads-2133151. html。

作者在线

购买本书可以免费访问由 Manning 出版公司主管的私有网络论坛，在这里可以发表书评、询问技术问题，并可以从作者和其他读者那里获得帮助。为了访问论坛和订阅消息，请访问 https://www. manning. com/books/akka-inaction。此页面提供有关注册后如何获得论坛信息，可获得的帮助以及论坛上的行为规则。

Manning 向广大读者提供了读者之间、读者与作者之间对话的平台，但不对参与其中的作者数量做出承诺，他们对 AO 论坛的贡献是出于自愿的（而且是没有报酬的）。AO 论坛和以前的讨论文档，只要本书还在发行，就可以通过出版者网站访问。

关于作者

雷蒙德·罗斯腾伯格是一位经验丰富的软件工程师、多语言程序员和软件架构师。他是一位 Scala 社区的活跃成员和 Akka 的贡献者，参与了 Akka-camel 模块的开发。

罗勃·贝克尔是一位经验丰富的软件开发人员，专注于并行后端系统和系统集成。他从 0.7 版本开始就使用 Scala 和 Akka。

罗勃·威廉姆斯是 ontometrics 的创始人，专注于包括机器学习在内的 Java 解决方案，十多年前开始从事基于 Actor 的编程开发，从那时起已经完成了几个项目。

关于封面插图

Akka 实战封面中的中国皇帝像选自于 1757～1772 年在伦敦发行的 Thomas Jefferys 的《不同国家服饰集锦，古代与现代（四卷）》。扉页显示，这些都是手工彩色铜版画，用阿拉伯树胶润色。Thomas Jefferys（1719～1771 年）被称为“地理学家英王乔治三世”。他是英国制图匠，是当时地图供应商的领导者。他为政府和其他机构刻板印刷地图，生产了许多商业地图和地图集，尤其是北美的。他的地图匠工作点燃了他对测绘地区服饰的兴趣，它们灿烂地显示在这四卷的图集中。

迷恋遥远的土地和休闲旅行是 18 世纪晚期新的潮流，这种收藏比较流行，既吸引了旅行者和神游旅行者（armchair traveler），也吸引了其他国家的居民。Jefferys 画卷的多样性生动地道出了一个世纪以前世界各国的独特性和个性特征。

在计算机图书难于区分的时代，Manning 创造性地提出了以两个世纪前区域生活的多样性作为图书封面，暗示计算机图书的多样性，并由 Jefferys 的画作付诸实践。

目录

序
致谢
译者序
关于本书
第 1 章　Akka 简介 …… 1
1.1　什么是 Akka? …… 3
1.2　Actor 简介 …… 3
1.3　两种扩展方法：建立实例 …… 4
1.4　传统扩展 …… 5
1.4.1　传统扩展和持久性：一切移入数据库 …… 5
1.4.2　传统扩展和交互应用：轮询 …… 7
1.4.3　传统扩展和交互应用：Web 服务 …… 8
1.5　用 Akka 进行扩展 …… 9
1.5.1　用 Akka 扩展和持久化：发送和接收消息 …… 10
1.5.2　用 Akka 扩展和交互应用：消息推送 …… 11
1.5.3　用 Akka 扩展和容错：异步解耦 …… 12
1.5.4　Akka 方式：发送和接收消息 …… 12
1.6　Actor：向上和向外扩展的编程模型 …… 13
1.6.1　异步模型 …… 14
1.6.2　Actor 操作 …… 15
1.7　Akka Actor …… 17
1.7.1　ActorSystem …… 17
1.7.2　ActorRef、邮箱和 Actor …… 18
1.7.3　分发器 …… 18
1.7.4　Actor 和网络 …… 20
1.8　总结 …… 20
第 2 章　搭建和运行 …… 21
2.1　克隆、构建和测试接口 …… 21
2.1.1　用 sbt 进行构建 …… 22

2.1.2 快进到 GoTicks.com REST 服务器 …… 23
2.2 探索应用中的 App …… 26
2.2.1 App 结构 …… 26
2.2.2 处理销售的 Actor：TicketSeller …… 30
2.2.3 BoxOffice Actor …… 31
2.2.4 RestApi …… 32
2.3 部署到云上 …… 34
2.3.1 在 Heroku 上创建 App …… 34
2.3.2 在 Heroku 上部署并运行 …… 35
2.4 总结 …… 36
第 3 章 Actor 测试驱动开发 …… 37
3.1 测试 Actor …… 37
3.2 单向消息 …… 39
3.2.1 SilentActor 实例 …… 39
3.2.2 SendingActor 实例 …… 42
3.2.3 SideEffectingActor 实例 …… 46
3.3 双向消息 …… 48
3.4 总结 …… 49
第 4 章 容错 …… 50
4.1 容错概述 …… 50
4.1.1 普通对象与异常 …… 52
4.1.2 Let it crash …… 55
4.2 Actor 生命周期 …… 58
4.2.1 启动事件 …… 58
4.2.2 停止事件 …… 58
4.2.3 重启事件 …… 59
4.2.4 生命周期综合 …… 60
4.2.5 生命周期监控 …… 62
4.3 监视 …… 63
4.3.1 监视器层次结构 …… 63
4.3.2 预定义策略 …… 65
4.3.3 自定义策略 …… 66
4.4 总结 …… 70
第 5 章 Future …… 71
5.1 Future 的应用实例 …… 71
5.2 Future 无阻塞 …… 75
5.3 Future 错误处理 …… 79

5.4 Future 组合 …… 82
5.5 Future 组合 Actor …… 89
5.6 总结 …… 90
第 6 章 第一个分布式 Akka App …… 91
6.1 向外扩展 …… 91
6.1.1 通用网络术语 …… 91
6.1.2 采用分布式编程模型的原因 …… 92
6.2 远程扩展 …… 93
6.2.1 把 GoTicks.com app 改造成分布式应用 …… 94
6.2.2 远程 REPL 活动 …… 95
6.2.3 远程查找 …… 98
6.2.4 远程部署 …… 104
6.2.5 多 JVM 测试 …… 107
6.3 总结 …… 111
第 7 章 配置、日志和部署 …… 113
7.1 配置 …… 113
7.1.1 尝试 Akka 配置 …… 113
7.1.2 使用默认值 …… 116
7.1.3 Akka 配置 …… 117
7.1.4 多系统 …… 118
7.2 日志 …… 120
7.2.1 Akka 中的日志记录 …… 120
7.2.2 使用日志 …… 121
7.2.3 Akka 的日志控制 …… 122
7.3 部署基于 Actor 的应用 …… 123
7.4 总结 …… 126
第 8 章 Actor 的结构模式 …… 127
8.1 管道和过滤器 …… 127
8.1.1 企业集成模式：管道和过滤器 …… 128
8.1.2 Akka 中的管道和过滤器 …… 128
8.2 企业集成模式：分发-收集模式 …… 131
8.2.1 适用性 …… 131
8.2.2 Akka 处理并行任务 …… 132
8.2.3 使用接收者列表实现分发组件 …… 133
8.2.4 使用聚合器模式实现收集组件 …… 134
8.2.5 组合组件实现分发-收集模式 …… 138
8.3 企业集成模式：路由表模式 …… 139

8.4 总结 …… 143
第9章 路由消息 …… 144
9.1 企业集成路由模式 …… 144
9.2 使用 Akka Router 实现负载平衡 …… 145
9.2.1 Akka Router 池 …… 147
9.2.2 Akka Router 群组 …… 152
9.2.3 ConsistentHashing Router …… 156
9.3 用 Actor 实现路由模式 …… 160
9.3.1 基于内容的路由 …… 160
9.3.2 基于状态的路由 …… 160
9.3.3 Router 的实现 …… 162
9.4 总结 …… 163
第10章 消息通道 …… 164
10.1 通道类型 …… 164
10.1.1 点对点通道 …… 165
10.1.2 发布-订阅通道 …… 165
10.2 特殊通道 …… 172
10.2.1 死信 …… 172
10.2.2 保证投递 …… 174
10.3 总结 …… 178
第11章 有限状态机和代理 …… 179
11.1 使用有限状态机 …… 179
11.1.1 有限状态机简介 …… 179
11.1.2 创建 FSM 模型 …… 180
11.2 FSM 模型的实现 …… 182
11.2.1 实现转换 …… 182
11.2.2 实现入口动作 …… 185
11.2.3 FSM 定时器 …… 189
11.2.4 FSM 的终止 …… 191
11.3 使用代理实现共享状态 …… 192
11.3.1 使用代理简单地共享状态 …… 192
11.3.2 等待状态更新 …… 194
11.4 总结 …… 195
第12章 系统集成 …… 196
12.1 消息终端 …… 196
12.1.1 归一化 …… 197
12.1.2 规范数据模型 …… 199

12.2 使用 Apache Camel 实现终端 …… 200
12.2.1 创建从外部系统接收消息的消费者终端 …… 201
12.2.2 实现生产者向外部系统发送消息 …… 206
12.3 实现 HTTP 接口 …… 209
12.3.1 HTTP 实例 …… 209
12.3.2 用 Akka-http 实现 REST 终端 …… 212
12.4 总结 …… 216
第 13 章 流 …… 217
13.1 基本流处理 …… 217
13.1.1 使用源和接收器复制文件 …… 221
13.1.2 实体化可运行图 …… 223
13.1.3 用 Flow 处理事件 …… 227
13.1.4 处理流中的错误 …… 230
13.1.5 用 BidiFlow 创建协议 …… 231
13.2 HTTP 流 …… 233
13.2.1 接收 HTTP 流 …… 233
13.2.2 HTTP 响应流 …… 235
13.2.3 内容类型和协调的自定义编组与解组 …… 235
13.3 用 Graph DSL 进行扇入和扇出 …… 239
13.3.1 广播流 …… 239
13.3.2 合并流 …… 241
13.4 协调生产者和消费者 …… 243
13.5 图的速率隔离 …… 246
13.5.1 对较慢的消费者，对事件进行汇总 …… 247
13.5.2 快速消费者的扩展度量 …… 247
13.6 总结 …… 248
第 14 章 集群 …… 249
14.1 为什么使用集群？ …… 249
14.2 集群成员关系 …… 251
14.2.1 加入集群 …… 252
14.2.2 离开集群 …… 257
14.3 集群作业处理 …… 261
14.3.1 启动集群 …… 264
14.3.2 使用路由进行工作分配 …… 265
14.3.3 弹性作业处理 …… 267
14.3.4 测试集群 …… 271
14.4 总结 …… 274

第 15 章　Actor 持久化 …… 276
15.1　事件提取恢复状态 …… 277
15.1.1　适时更新记录 …… 277
15.1.2　不使用 update 持久化状态 …… 278
15.1.3　Actor 的事件提取 …… 279
15.2　持久化 Actor …… 280
15.2.1　持久化 Actor …… 281
15.2.2　测试 …… 283
15.2.3　快照 …… 285
15.2.4　持久化查询 …… 288
15.2.5　序列化 …… 290
15.3　集群持久化 …… 294
15.3.1　单例集群 …… 297
15.3.2　分片集群 …… 299
15.4　总结 …… 302
第 16 章　性能提示 …… 304
16.1　性能分析 …… 304
16.1.1　系统性能 …… 305
16.1.2　性能参数 …… 306
16.2　Actor 性能测量 …… 307
16.2.1　收集邮箱数据 …… 308
16.2.2　收集处理数据 …… 313
16.3　解决瓶颈、提高性能 …… 314
16.4　配置分发器 …… 315
16.4.1　识别线程池问题 …… 315
16.4.2　使用多个分发器实例 …… 316
16.4.3　静态调整线程池的大小 …… 318
16.4.4　使用动态线程池 …… 320
16.5　改变线程释放策略 …… 321
16.6　总结 …… 324
第 17 章　展望 …… 325
17.1　Akka-typed 模块 …… 325
17.2　Akka 分布式数据 …… 328
17.3　总结 …… 328

第1章 Akka简介

本章导读

- Actor编程模型简介。
- Akka Actor。

直到20世纪90年代中期，互联网革命之前，编写只在一台计算机，单个CPU上运行的程序是十分正常的。如果程序运行不够快，那标准的反应就是等CPU速度变得更快，而不会去考虑改变任何代码。问题解决了，全世界的程序员都在享用免费的午餐，生活过得好不惬意。

2005年Herb Sutter在Dr. Dobb's Journal中写道："根本的改变势在必行。"简言之，CPU时钟频率的增加达到了一个极限，免费的午餐没有了。

如果程序需要执行得更快，或者需要支持更多的用户，则必须是并发的（concurrent）。

扩展性（scalability）是系统适应资源变化而不会对性能有负面影响的一种度量方式。并发（concurrency）是实现扩展性的手段：前提是如果需要，可以向服务器添加更多的CPU，应用程序可以自动地使用它们。这也是一份免费的午餐。

在2005年，有些公司在集群多处理器服务器上运行应用程序。程序语言已经出现对并发支持，但非常有限，且被广大程序员认为是黑色的诅咒。Herb Sutter在他的论文中预言"编程语言……将逐步被迫处理好并发的实现"。直到今天并发技术持续高速发展，并且可以使应用程序运行在云端更多的服务器上，可以集成很多系统，并跨很多数据中心。终端用户不断增长的需求推动了系统性能和稳定性的发展。

那么这些新的并发功能在哪里？许多编程语言对并发的支持，特别是在JVM（Java Virtual Machine，Java虚拟机）上，几乎不能改变。虽然并发API的细节有了明显的提高，但仍然必须与底层的线程和锁打交道，这些东西是非常难使用的。

另一种方法是扩展（增加资源，如增加现有服务器的CPU）。外向扩展（scaling out）是指大幅增加集群的服务器数量。20世纪90年代后，编程语言对网络的支持几乎没有什么改变。许多技术仍然必须使用RPC（远程调用）进行网络通信。同时，云计算服务和多核

CPU 架构的发展，使得计算资源空前充裕。

PaaS（Platform as a Service，平台即服务）大大简化了分布式系统的配置和部署。像 AWS EC2（亚马逊 Web 服务弹性计算云）和 Google 计算引擎这样的云服务可以在几分钟内启动上千台服务器，如 Docker、Puppet、Ansible 和其他工具使得易于管理和打包虚拟服务器上的应用程序。

而设备中的 CPU 数目也在持续增加，即使智能手机和平板电脑都已拥有多个 CPU 内核。

但这并不意味着，对于任何问题都可以提供无限的资源。最终，任何事情都要考虑成本和效率。因此，所有这一切都与如何有效扩展应用程序有关。就像你没有用过指数级时间复杂度的排序算法一样，这需要考虑扩展的成本。

在扩展应用程序时，有以下两个期望：

- 用有限的资源应付任意增长的需求是不现实的，因此在理想状态下，使需求增长所需的资源增长速度变慢，或呈线性关系更好。图 1-1 显示了需求和所需资源数目的关系。
- 如果必须增加资源，那么理想的情况是程序的复杂度不变或略有增加。图 1-2 显示了资源数目和复杂性之间的关系。

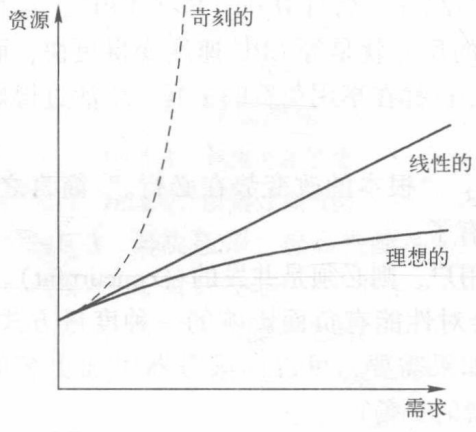

图 1-1　需要和所需资源数目的关系

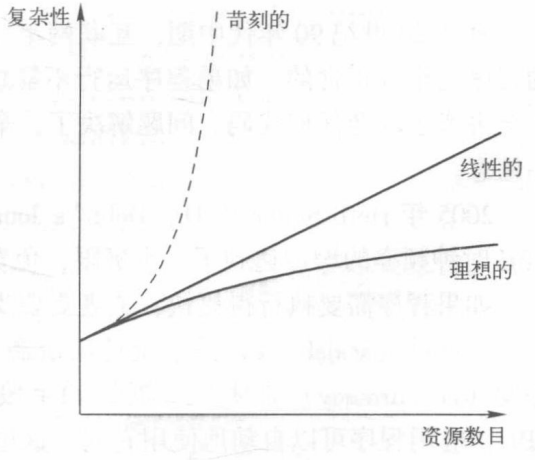

图 1-2　资源数目和复杂性之间的关系

资源的数目和复杂性决定了扩展的成本。我们的计算忽略了许多因素，但很容易看出，这两个比率对扩展成本有很大影响。

苛刻的关系意味着需要付出越来越多的未充分应用的资源。另一噩梦般的场景是：增加了更多的资源，却使程序的复杂性难以控制。

这说明了我们要追求两个目标：复杂性要保持尽可能低，扩展应用程序时资源必须高效利用。

可以使用当今的工具（线程和 RPC）达到这两个目标吗？RPC 扩展和低层的线程扩展都不是好办法。RPC 假定网络调用和本地方法调用没有区别。每次 RPC 调用需要阻塞当前线程等待网络的响应，类似于本地方法调用，这是比较耗时的，阻碍了资源的高效利用。