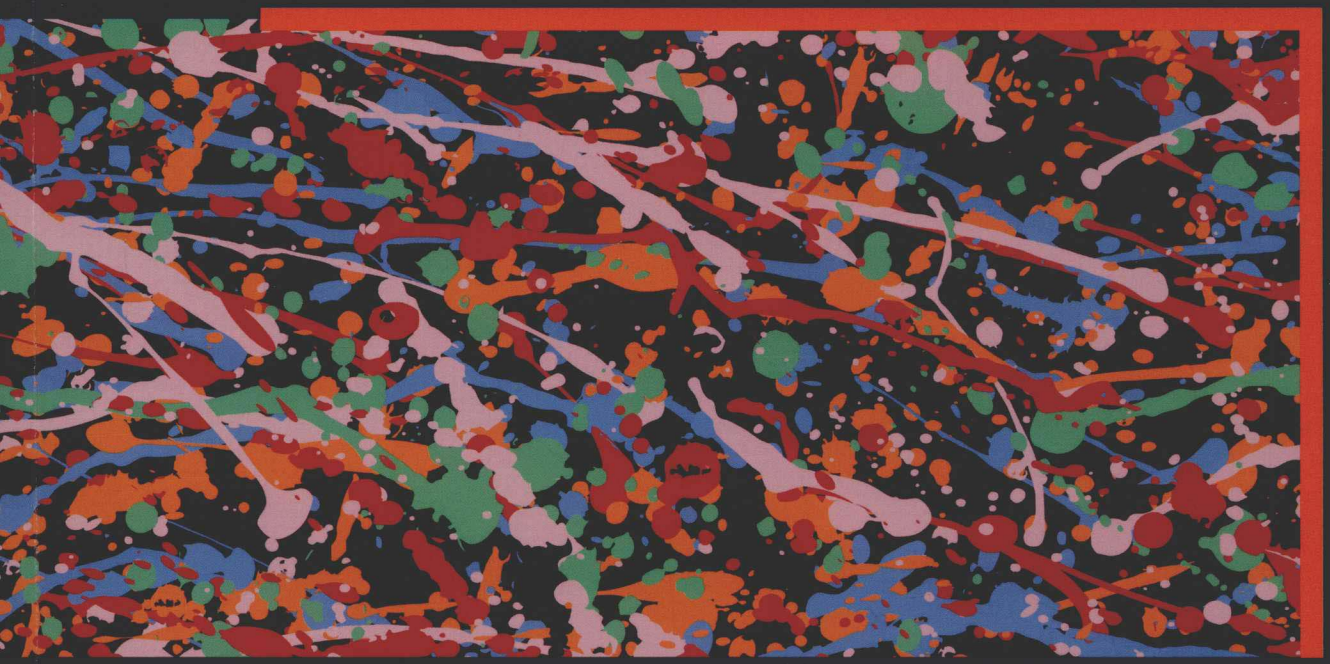
华章程序员书库

[PACKT] PUBLISHING

HZ BOOKS 华章IT

并发模式与应用实践

Concurrent Patterns and Best Practices

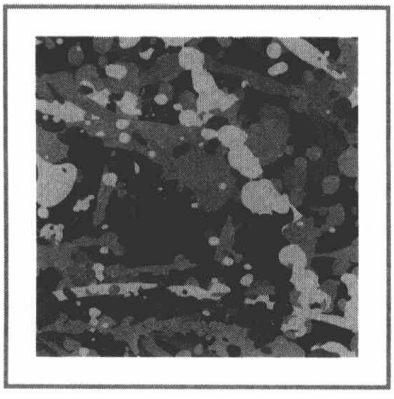


[印度] 阿图尔·S. 科德（Atul S. Khot）著

徐坚 译

机械工业出版社
China Machine Press

华章程序员书库



Concurrent Patterns and Best Practices

并发模式与应用实践

[印度] 阿图尔·S. 科德（Atul S. Khot）著

徐坚 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

并发模式与应用实践 /（印）阿图尔·S. 科德（Atul S. Khot）著；徐坚译 . —北京：机械工业出版社，2019.4
（华章程序员书库）
书名原文：Concurrent Patterns and Best Practices

ISBN 978-7-111-62506-3

I. 并… II. ① 阿… ② 徐… III. 并行程序 - 程序设计 IV. TP311.11

中国版本图书馆 CIP 数据核字（2019）第 071514 号

本书版权登记号：图字 01-2018-8346

并发模式与应用实践

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：杨宴蕾 责任校对：殷 虹
印 刷：北京瑞德印刷有限公司 版 次：2019 年 5 月第 1 版第 1 次印刷
开 本：186mm×240mm 1/16 印 张：13.75
书 号：ISBN 978-7-111-62506-3 定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 读者信箱：hzit@hzbook.com

The Translator's Words 译 者 序

并发能极大地整合和提高系统的计算性能，特别在以大数据、云计算为特征的信息时代，关于并发的学习和实践具有重大意义。然而，要学好并发，需要遵循一定的章法、范式，这就是并发模式。掌握好并发模式，将使读者的并发设计及开发能力如虎添翼。不过，要掌握好并发模式并能付诸实践，也并非易事。所幸的是，这本书的横空出世为读者学习并发带来了福音。

Atul S. Khot 作为一名自学成才的优秀程序员，有丰富的编程经验和对设计模式的深入研究和深刻洞见。本书通俗易懂，理论与实践紧密结合，书中给出的代码简练、质量高，配图也直观明了、贴近生活。本书对于渴望学习并发模式并希冀能快速实践的读者，将会带来立竿见影的效果。

本书的翻译得到了同行、老师、学生和朋友的帮助与鼓励，在此表示真挚的谢意，特别感谢甘健侯、张姝、李佳蓓、张利明以及姚贤明。书中的文字与内容力求忠实于原著，但由于译者水平有限，加上时间仓促，译文中难免有疏漏之处，敬请读者批评指正。

徐 坚

2019 年 1 月于昆明

前　言 *Preface*

感谢你购买本书！我们生活在一个并发的世界中，并发编程是一项越来越有价值的技能。

我还记得当我理解了UNIX shell管道的工作原理的那一刻，便立即对Linux和命令行“一见钟情”，并尝试了许多通过管道连接的组合过滤器（过滤器是一种程序，它从标准输入设备读取数据，再写入标准输出设备）。我一直都在和并发程序打交道，我对命令行的创造性和力量感到很惊讶。

后来，由于项目变化，我致力于用多线程范式编写代码。所使用的编程语言是我钟爱的C或C++，然而，令我惊讶的是，我发现维护一个用C/C++编写的多线程遗留代码库是一项艰巨的任务。这是因为共享状态是随意管理的，一个小错误就可能让我们陷入调试噩梦！

大约在那个时候，我开始了解面向对象设计模式和一些多线程模式。例如，我们希望将一个大的内存数据结构安全地显露给多个线程。我读过有关reader/writer锁模式的内容，该模式使用智能指针（一种C++习语），并据此编写解决方案。

采取此方法后并发错误就消失了！此外，该模式使得线程很容易理解。在我们的示例中，writer线程需要对共享数据结构进行不频繁但独占的访问，reader线程只是将这个结构作为不可变的实体来使用。看啊，没有锁！

没锁带来巨大的好处，随着锁的消失，死锁、竞争和饥饿的可能性也随之消失。感觉真棒！

我在这里得到了一个教训！我得不断学习设计模式，根据不同模式努力思

考手边的设计问题。这也帮助我更好地理解代码！最终，我对如何驯服并发这头野兽有了初步的了解！

设计模式是用于解决常见设计问题的可重复使用的解决方案。设计解决方案就是设计模式。你的问题领域可能会有所不同，也就是说，你需要编写的业务逻辑将用于解决你手中的业务问题。但是，一旦使用模式，任务就能很快完成！

例如，当我们使用面向对象范式编写代码时，我们使用“四人组”（GoF）开发的设计模式（http://wiki.c2.com/?DesignPatternsBook）。这本名著[⊖]为我们提供了一系列设计问题及其解决方案。虽然这种策略模式一直保持不变，但它仍被人们广泛使用。

几年后，我转战到 Java 领域，并使用 ExecutorService 接口来构建我的代码。开发代码非常容易，几个月的运行中没有出现任何重大问题。（虽然有一些其他问题，但没有数据冲突，也没有烦琐的调试！）

随后，我进入函数式编程的世界，并开始编写越来越多的 Scala 程序。这是一个以不变数据结构为标准的新领域，我学到了一种截然不同的范式。

Scala 的 future 模式和 actor 模式给出了全新的视角。作为程序员，我能感受到这些工具带来的力量。一旦你跨越了认知曲线（诚然在开始时有点畏惧），就能编写许多更安全且经得起推敲的并发代码。

本书讲述了许多并发设计模式，展示了这些模式背后的基本原理，突出了设计方案。

本书目标读者

我们假定你有一定的 Java 编程基础，理想情况下，你已经用过多线程 Java 程序，并熟悉“四人组”的设计模式，你还能轻松地通过 maven 运行 Java 程序。

本书将带你进入下一个阶段，同时向你展示许多并发模式背后的设计主题。本书希望帮助开发人员通过学习模式来构建可扩展、高性能的应用程序。

⊖ 该书中文版已由机械工业出版社引进出版，书号为 978-7-111-07575-2。——编辑注

本书包含的内容

第 1 章介绍并发编程。正如你将看到的，并发本身就是一个领域。你将了解 UNIX 进程以及并发模式的管道和过滤器。本章涉及并发编程的综述，你可能已对这方面有所了解。

第 2 章涵盖一些关键的基本概念，并介绍 Java 内存模型的本质。你将了解共享状态模型中出现的竞争条件和问题，并尝试第一个并发模式：手拉手锁定。

第 3 章包括显式同步可变状态和监视器模式，你会看到这种方法存在很多问题。我们将详细介绍主动对象设计模式。

第 4 章介绍线程如何通过生产者 / 消费者模式相互通信，并介绍线程通信的概念，然后解释主 / 从设计模式。本章还将介绍 fork-join 模式的一个特例：map-reduce 模式。

第 5 章讨论构建块，还将讨论阻塞队列、有界队列、锁存器、FutureTask、信号量、屏障、激活和安全等内容。最后，描述不可变性以及不可变数据结构固有的线程安全性。

第 6 章介绍 future 并讨论它的一元性质，包括转型和单子模式，还将阐释 future 模式的构成，同时会介绍 Promise 类。

第 7 章介绍 actor 范式。再次回顾主动对象模式，然后解释 actor 范式，特别是未明确的锁定性质。还将讨论 ask 与 tell、become 模式（并强调其不变性）、流水线、半同步或半异步，并通过示例代码进行说明。

读者水平及所需环境

为了充分利用本书，你应该掌握一定水平的 Java 编程知识和 Java 线程基础知识，能够使用 Java 构建工具 maven。书中提供了需要复习的重要内容，并通过 Java 线程示例对此进行补充。

使用诸如 IntelliJ Idea、Eclipse 或 Netbeans 等集成开发环境会很有帮助，但并非必须。为了说明函数并发模式，最后两章使用 Scala，这两章的代码使用基本的 Scala 结构。我们建议读者最好浏览一下介绍性的 Scala 教程，这样做很

有益处。

下载示例代码及彩色图像

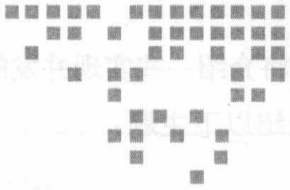
本书的示例代码及所有截图和样图，可以从 http://www.packtpub.com 通过个人账号下载，也可以访问华章图书官网 http://www.hzbook.com，通过注册并登录个人账号下载。另外还可以从 https://github.com/PacktPublishing/Concurrent-Patterns-and-Best-Practices 访问这些代码。

作者 / 评阅者简介 *About the Author*

作者 Atul S. Khot 是一位自学成才的程序员，他使用 C 和 C++ 编写软件，并用 Java 进行过大量编程，另外还涉猎多种语言。如今，他越来越喜欢 Scala、Clojure 和 Erlang。Atul 经常在软件大会上发表演讲，还曾经担任 Dobb 博士产品奖评委。他是 Packt 出版社出版的《*Scala Functional Programming Patterns*》和《*Learning Functional Data Structures and Algorithms*》的作者。

评阅者 Anubhava Srivastava 是一名首席架构工程师，拥有超过 22 年的系统工程和 IT 架构经验。他撰写了 Packt 出版社出版的《*Java 9 Regular Expressions*》一书。作为一名开源传播者，他积极参与各种开源开发，在一些流行的计算机编程问答网站如 Stack Overflow 上声誉 / 得分超过 17 万，并且在整体声誉排名中名列前 0.5%。

Contents 目　录



第 1 章 Chapter 1

并发简介

什么是并发和并行？我们为什么要研究它们？本章将介绍并发编程领域的诸多方面。首先简要介绍并行编程，并分析我们为什么需要它，然后快速讨论基本概念。

“巨大的数据规模”和“容错”作为两股主力推动并发程序设计技术不断向前。在我们阅读本章时，里面的一些示例将涉及一些集群计算模式，例如MapReduce。对当今的开发人员来说，应用程序扩展性是非常重要的概念，我们将讨论并发如何帮助对应用程序进行扩展。水平扩展（https://stackoverflow.com/questions/11707879/difference-between-scaling-horizontally-and-vertically-for-databases）是当今大规模并行软件系统背后的关键技术。

并发使得并发实体之间必须实现通信。我们将研究两个主要的并发模型：消息传递模型和共享内存模型。我们将使用“UNIX shell 管道”描述消息传递模型，然后，我们将描述共享内存模型，并讨论显式同步为何带来如此多的问题。

设计模式是上下文中出现的设计问题的解决方案。通过掌握模式目录，有助于我们针对特定问题提出一个更好的设计方案，本书将介绍常见的并发设计模式。

本章最后将介绍一些实现并发的替代方法，即 actor 范式和软件事务性内存。

本章将介绍以下主题：

- 并发
- 消息传递模型
- 共享内存和共享状态模型
- 模式和范式

如需完整的代码文件，可以访问 https://github.com/PacktPublishing/Concurrent-Patterns-and-Best-Practices。

1.1 并发轻而易举

我们从一个简单的定义开始本章的学习。比如，当事情同时发生时，我们说事情正在并发。然而，就本书而言，只要可执行程序的某些部分同时运行，我们就是在进行并发编程。我们使用术语“并行编程”作为并发编程的同义词。

这个世界充满了并发现象。举个现实生活中的例子，假设有一定数量的汽车行驶于多车道高速公路，然而，在同一车道上，汽车需要跟随前面的车辆，在这种情况下，车道就是一种共享资源。

当遇到收费站时，情况会发生变化，每辆车会在其车道停留一两分钟去支付通行费和拿收据。当收费员对一辆车收费时，后面的车辆需要排队等待。但是，收费站有不止一个收费窗口，多个窗口的收费员会同时向不同的汽车收费。如果有三个收费员，每人服务一个窗口，那么三辆车可以在同一时间点支付费用，也就是说，它们可以并行接受服务，如图 1-1 所示。

请注意，在同一队伍排队的汽车是按顺序缴费的。在任何给定时刻，收费员只能为一辆车提供服务，因此队列中的其他车需要等待，直到轮到他们。

当我们看到一个收费站只有一个收费窗口的时候会感到奇怪，因为这不能提供并行的收费服务，严格的按顺序缴费会使大家不堪忍受。

当车流量过大（比如假期）时，即使有很多收费窗口，每个窗口也都会成

为瓶颈，用于处理工作负载的资源会变得更少。

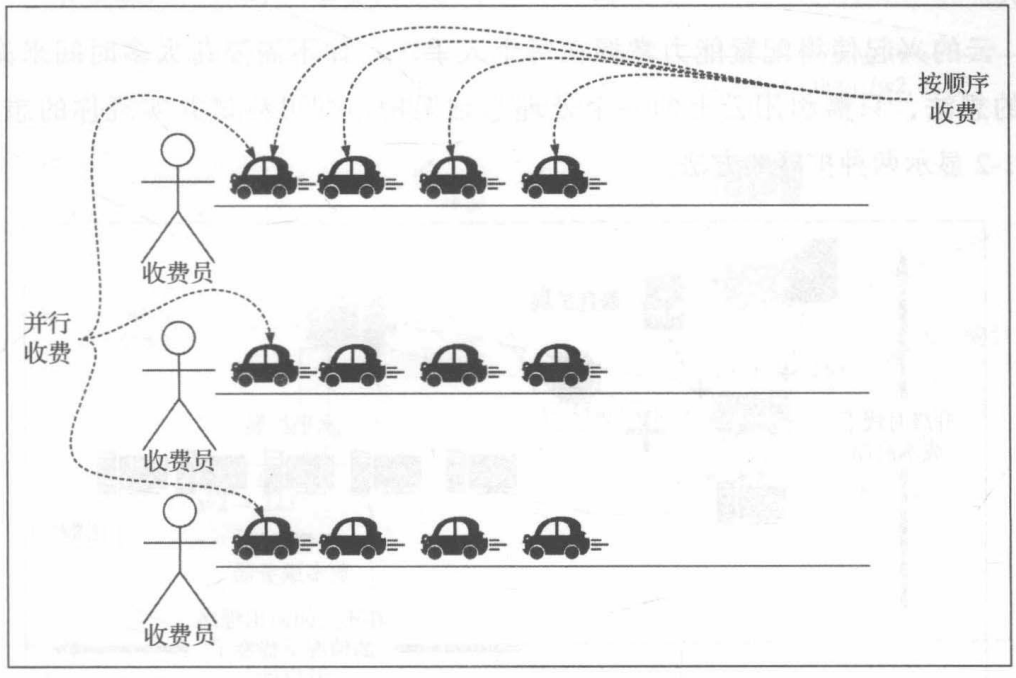


图 1-1 汽车并行收费

1.1.1 推动并发

让我们回到软件世界，比如你想边听音乐边写文章，这不是一个基本需求吗？是的，你的电子邮箱程序也应该并行工作，以便你可以及时收到重要的电子邮件。如果这些程序都不能并行运行，很难想象人们怎么工作。

随着时间的推移，软件占用的内存变得越来越大，需要更多更快的 CPU。例如，现在的数据库事务每秒都在增加，数据处理需求超出了任何一台机器的能力，因此，人们采用了分治策略（divide and conquer strategy）：许多机器在不同的数据分区上同时工作。

另一个问题是芯片制造商正在触及芯片速度的极限，改进芯片以使 CPU 更快的办法具有固有的局限性。有关此问题的清晰解释，请参见 http://www.gotw.ca/publications/concurrency-ddj.htm。

今天的大数据系统每秒处理数万亿条消息，并且全部使用商业硬件（我

们在日常开发中用的普通硬件），没有什么特别的，它们就像超级计算机一样强大。

云的兴起使得配置能力掌握在每个人手中。你不需要花太多时间来测试新的想法，只需租用云上的一个处理基础架构，即可测试并实现你的想法。图 1-2 显示两种扩展的方法。

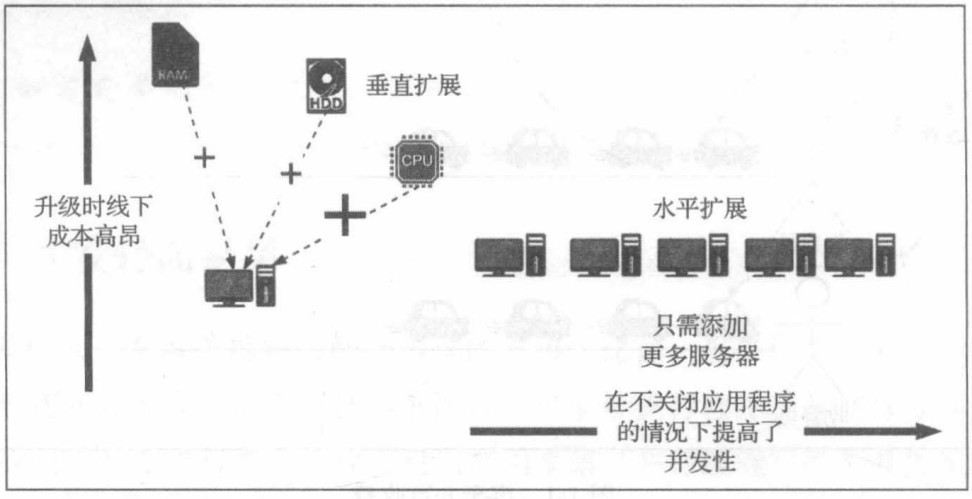


图 1-2　两种扩展方法

中央基础设施的设计主要有两种扩展方法：水平扩展与垂直扩展。水平扩展本质上意味着使用分布式并发模式，它具有成本效益，是大数据领域的一个突出理念。例如，NoSQL 数据库（比如 Cassandra）、分析处理系统（比如 Apache Spark）和消息代理（比如 Apache Kafka）都使用水平扩展，这意味着分布式和并发处理。

另一方面，在单台计算机中安装更多内存或提高处理能力是垂直扩展的一个很好的例子。在网站 https://www.g2techgroup.com/horizontal-vs-vertical-scaling-which-is-right-for-your-app/ 中可以看到两种扩展方法的比较。

我们将研究水平扩展系统的两个常见并发主题：MapReduce 和容错。

1.1.1.1　MapReduce 模式

MapReduce 模式是需要并发处理的常见例子。图 1-3 显示一个单词频率计数器，如果有数万亿字的文本流，我们需要查看文本中每个单词出现的次数。

该算法非常简单：我们将每个单词的计数保留在哈希表中，单词为键，计数器为值。哈希表允许我们快速查找下一个单词，并递增相关值（计数器）。

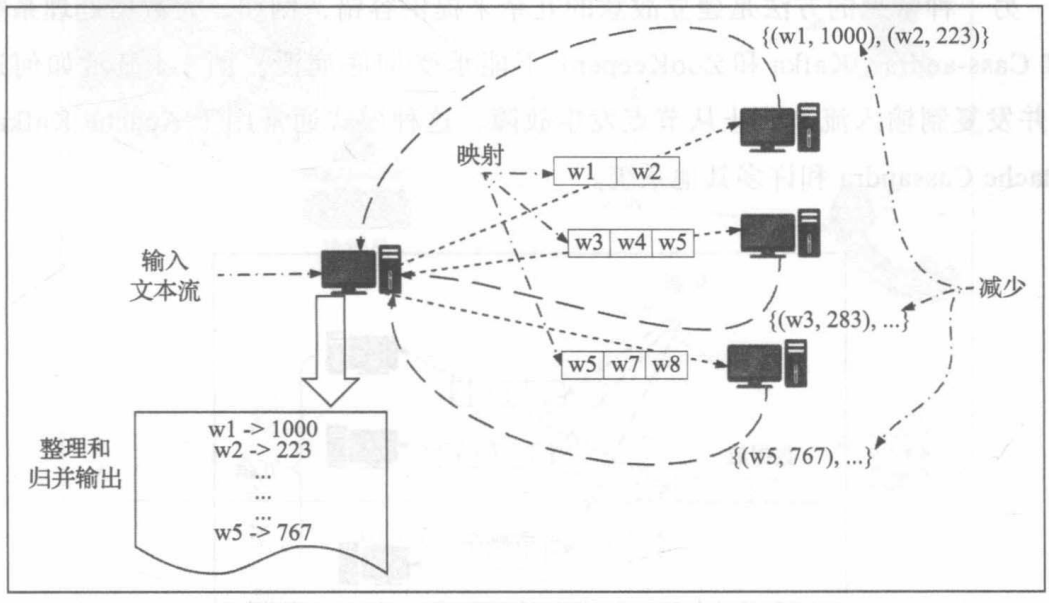


图 1-3 词频计数器

在给定输入文本大小的情况下，单个节点的内存无法容纳整个哈希表。通过使用 Map-Reduce 模式，可以为并发提供一种解决方案，如图 1-3 所示。

解决方案是分治策略：维护一个分布式哈希表，并运行适用于集群的相同算法。

主节点读取并分析文本，然后将其推送到一组“从属处理节点”（简称为“从节点”，与“主节点”对应）。这个想法是以一种由一个从节点处理一个单词的方式去分发文本。例如，给定三个从节点，如图 1-3 所示，我们将按范围划分：将以字符 {a..j} 开头的节点推送到节点 1，将以 {k..r} 开头的节点推送到节点 2，再将其余以 {s..z} 开头的节点推送到节点 3。这就是映射的部分（将工作分散）。

一旦流处理完之后，每个从节点将其频率结果发送回主节点，主节点打印结果。

从节点全部都在同时进行相同的处理。请注意，如果我们添加更多的从节