Broadview®
www.broadview.com.cn

高性能
服务系统构建与实战

服务系统 / 构建与实战
高性能 / 高可用 / 可伸缩

银文杰 / 编著

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

银文杰

笔名：说好不能打脸

博客地址：blog.csdn.net/yinwenjie

资深IT码农，最大爱好就是敲敲代码，写写博客，研究研究创业热点。CSDN博客作者、CSDN Java EE知识库特约编辑。曾参与电信行业、物流行业多个核心系统建设，对系统顶层设计、技术线路规划、业务系统性能调整有较丰富的经验；也曾有几年头脑发热拍案创业，兼职市场销售、电话客服、公司保安以及清洁大叔。

高性能
服务系统构建与实战

银文杰 / 编著

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

内 容 简 介

影响业务系统性能的因素很多，计算机系统的各个层面都有涉及：从硬件、网络、操作系统、中间件、存储，直到自身代码质量。所有技术团队都曾为解决性能问题、提高性能峰值绞尽脑汁，从千头万绪到生不如死。本书基于作者 10 余年工作经历中踩过的技术神坑，总结整理而成。虽然不能将计算机系统各个层面中影响性能的因素全部介绍完，但还是希望通过讨论业务系统负载层、网络通信层解决性能问题的过程，启发读者，为读者在工作中解决性能问题提供借鉴思路。

本书适合计算机软件领域中立志在架构师职业路线上长期发展的技术人员阅读，无论读者是有一定工作经验的软件工程师、运维工程师还是在校大学生，都适合阅读本书。本书知识点横跨系统架构领域和软件架构领域，所以为了更好地阅读本书，读者最好曾经使用过 Linux 操作系统，也最好有 Java 编程语言的使用能力。

图书在版编目（CIP）数据

高性能服务系统构建与实战 / 银文杰编著. —北京：电子工业出版社，2017.7
ISBN 978-7-121-31509-1

Ⅰ. ①高… Ⅱ. ①银… Ⅲ. ①软件设计 Ⅳ.①TP311.1

中国版本图书馆 CIP 数据核字(2017)第 105157 号

策划编辑：付　睿
责任编辑：石　倩
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
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：27.5　字数：595 千字
版　　次：2017 年 7 月第 1 版
印　　次：2017 年 7 月第 1 次印刷
定　　价：89.00 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。

序 言

科学的学习方法将大脑的状态分为三个区：舒适区、学习区和恐慌区。在舒适区中你可以基于自己熟悉的知识去做一些习以为常的事情，因为你已经具备了处理这些事情所需要的知识，所以做这些事情一般都会很得心应手。大脑状态处于恐慌区时给你的体验和舒适区刚好相反，由于你不具备处理这些事情的任何知识，或者说处理这些事情已经超出了你最大的知识范围，所以你会对事情的结果感到不确定，甚至沮丧、焦虑、崩溃、放弃。处于恐慌区时是不利于学习的，因为你的大脑思维不能被顺序整理，不能被归纳总结。

学习区又叫作脱离舒适区，处于这个区间的大脑，在做对应的事情时会感觉到挑战，并处于亢奋状态。让大脑进入学习区的事情都有这样的特点：你的大脑可以利用既有知识引申总结新的知识，并对自身知识树的缺失部分进行补全。所以，**让大脑脱离舒适区进入非舒适区是个人能力进步的一个根本要素。**例如，你可以使用 Java 进行编程活动，熟练自如后再在这个基础上学习 Groovy、Scala 等编程语言；再例如，你拥有了自己的编程习惯，再在此基础上融入别人的编程方法。以上两个例子都是在同一知识领域体系下的大脑状态区域平移。你也可以让你的大脑状态在不同的知识领域下进行平移，例如做开发的朋友可以去尝试做产品团队、市场团队的一些工作。我的偶像，罗辑思维的罗振宇老师对此有一个非常棒的总结：持续地做你不擅长的事。

总之，不能让自己的大脑在舒适区待得太久。在舒适区待久了的人也有一些共同表现，例如对新生事物天生持抵制态度，听不进去别人的建议，在职场“混资历”，甚至看不得别人取得任何成绩。总之如果有一天你发现自己听到类似“我都工作 20 年了，什么没见过？”“像你这样的项目，我当年一个人带 20 个！”这样的话，那么讲这种话的人一定是一个让自己在舒适区待得太久的人。不要混资历，不要用你的战术勤奋掩盖你的战略懒惰。

银文杰

2017 年 5 月

前 言

本书主要的代码示例采用 Java 写成，对于一些相对独立章节中的代码，笔者将其整理后形成示例工程。例如实战章节中的日志采集工程、图片服务工程，笔者已经上传到了 CSDN 的线上资源管理中，可供读者自行下载。本书一共分为四个部分，第一部分对日常开发任务中经常遇到的问题进行了总结，并将这些问题分类，分解出这些问题在整个软件架构中的位置。第二部分、第三部分和读者一起讨论软件架构中的负载层性能设计、业务层性能设计并穿插讲解了一些存储层的设计关注点，其中将详细讨论一些具体的软件/组件应用以及它们的工作原理。第四部分为实践章节，这一部分将基于已经介绍过的知识点和读者一起将它们用于工程实战，对于之前没有涉及的新知识点，也会在其中进行简要说明。

本书大量使用操作系统、Java 知识体系、软件设计中的基础知识，包括但不限于：操作系统线程原理、悲观锁/乐观锁、软件设计模式等。例如本书中至少使用的设计模式包括：命令模式、构建者模式、观察者模式、责任链模式；本书中至少涉及的 Java 基础知识包括：有限/无限队列、悲观锁/乐观锁、SPI 规则、concurrent 工具包、状态机；本书还关联至少如下第三方组件：分布式文件系统、Redis、关系型数据库、Keepalived、ZooKeeper。因为篇幅所限，本书并不可能用太多的文字对这些基础知识、第三方组件进行详细介绍，甚至不会专门说明某些技术点。所以本书更适合有一定一线业务系统开发经验的软件工程师阅读，并且在工作过程中使用过 Linux 系列操作系统（最好是 CentOS），因为本书讲解的知识点、介绍的安装运行方法、讨论的工作原理、描述的操作过程、给出的示例代码环境全部都是基于 Linux 操作系统的。如果你想从一名开发人员成长为一名软件架构师，那么本书绝对是你合适的一块“垫脚石”；本书还适合有一定系统运维工作经验的 IT 工程师阅读，如果你想完成从传统的 IOE 系统运维到移动互联网系统领域运维的蜕变，那么本书所介绍的知识也会给你一定的启发。

由于本书内容较丰富，文字讲解部分就占用了相当的篇幅，所以为了尽可能节约篇幅，本书在列举代码段落时往往只保留了主要的代码片段，并以“……”表示代码段落中有省略的片

段。另外，如果在代码片段中使用 Java 标准注释规范，则将占据多余的空间，所以本书大部分使用了 Java 中的单行注释方式对代码目的进行说明。类似成体系的工程示例，如日志采集案例、图片处理案例等，都在相关章节中注明了完整的工程示例下载地址，以便读者查看更详尽的实现方法。最后，本书中多数图片由笔者自行绘制（90%以上），有一部分图片来源于互联网资源，凡是后者笔者都在图片下方进行了明确的说明。

本书成书于笔者对自己博客文章的整理，其中有 30%的内容为成书整理时新增。在这个过程中有很多朋友给予笔者帮助，帮助笔者校正博客文章中的错误。特此感谢以下网友（CSDN 账号，排名不分先后）：amadis_chen、自然的发呆、thisisgpy、github_33423142、shadabing、fyc198610、zkq1989、sinat_25444367、Tony_tec、周创、gongfengying、z3133464733、qq_32159081、weixin_33750642、fengyong7723131、白糖、a35946729、zzpapzzp、qq_16387501、LX_871225、littlebugu 。

笔者还要感谢家人，他们都以自己的方式在笔者写作期间默默地给予支持。笔者最后还要特别感谢电子工业出版社博文视点编辑付睿老师和参与本书校对整理工作的各位编辑，没有他们的勤劳付出就不会有本书的出版发行。

目 录

第三部分 系统间通信

第四部分　场景实战

第一部分

前序

本部分将介绍在实际工作中技术人员最容易遇到的和系统可用性、系统高性能有关的一些问题，使读者对本书将要讨论的问题有一个概要性的理解，这样便于各位读者选择性地阅读本书章节。第 1 章中将分类讨论这些问题的由来，描述的这些问题场景也将越来越会对软件系统造成更大的伤害。第 2 章中将对绝大多数业务系统进行具有共性的层次分解，并结合第 1 章的内容介绍每一层最可能出现的问题。

第 1 章
那些年一起踩的坑

翻开此书的读者一定是一名软件技术人员。无论职位高低、技术生熟、资历深浅，一名软件技术人员一定是从一个一个软件项目/产品中熬出来的：各种系统磨难造就了今天的你。可能你夜以继日写代码，只为了能按时完成 JIRA（Atlassian 公司出品的项目与事务跟踪工具）每一次发布；可能你是一名公司的救火队员，坐着红眼航班从一个火场奔向另一个火场；可能你是带领着一帮项目/产品团队的技术人员，一上班就坐在会议室和负责需求的同事讨论，然后在对方下班时感叹：今天又要加班写代码了；你还可能被上级要求电话 24 小时开机，而且确实常常在熟睡时被叫醒——因为接口又调不通了；这都算好的，更悲催的是由于莫名其妙的脏数据问题，所以团队被要求每周都要手工进行一次数据比对和清理，无论是盘点订单数据、交易数据、库存数据还是用户积分；或者你比较幸运，被公司选中设计新系统，但是却无意发现下面的程序员经常抱怨接口故障率太高，经常为此加班……

无论你在公司扮演什么样的技术角色，以上类似的问题肯定是遭遇过的。实际上大多数情况下单个系统独立工作都是没有问题的，或者说不会出现很多难以解决的问题。而一旦一个业务需要多个系统联合工作才能完成，那么往往就会出现各种问题了。

1.1 性能问题

读者可能遇到过这样的问题，线上的某系统某功能模块的 TPS/OPS（每秒请求/事务数量）时常保持在几千上下，监控环境和系统日志所表现出的结果都是正常的。但是随着业务的成熟，这个系统模块的 TPS 开始出现上万的情况。这时系统就会出现莫名其妙的错误，要么就是大量客户端请求超时，要么就是磁盘 I/O 尤其是写操作时特别高，要么就是数据库出现无数的表锁。引起这些问题的原因却多种多样，有可能是 Linux 操作系统的若干参数没有优化，有可能

是基础设施薄弱，还有可能是数据表设计阶段没有联系业务方进行聚集索引和非聚集索引的调整。这些问题的最终结果就是导致系统无法承受较猛烈的流量洪峰。

性能问题的调优涉及多个方面：硬件层面、网络层面、操作系统层面、应用软件层面和代码质量层面。例如，根据笔者经验，如果使用 Sonar 进行代码审查出现圈复杂度较高的代码片段，那么这些代码片段不仅难以维护，而且代码性能一般也不会太好。如果使用 JVM 系列语言（例如 Groovy）来编写高性能的代码，那么首先就应该熟练线程/线程池的定义和使用方式。因为类似 JVM 系列的语言都直接采用了操作系统层面的线程和锁的概念来管理并发性，并在业务集中点优先选用乐观锁解决冲突。再例如，如果你使用了 JVM 承载业务服务，但是 JVM 的垃圾回收器没有做调整，例如没有使用-server 模式，那么这个业务服务的性能肯定不会太好。再举一个基础设施层面的例子，首先技术团队必须注重基础设施建设，使用磁盘阵列 RAID1+0 方案为数据库搭建的块存储设施，在实际应用方面肯定强于单块企业级硬盘，当然如果是使用类似阿里的云存储方案，就不需要担心这个问题，因为别人已经帮你做了。

以上提到的这些“质量门”，本书当然不可能全部进行讲解（但是笔者的个人博客对这些问题都有分析，可参见 http://blog.csdn.net/yinwenjie）。本书将依据如下的思路来讲解性能问题的优化方式：既然无法减轻整个系统的性能压力，那么就在系统内部分散压力并且缓存待完成的任务。分散压力到多个服务节点，然后从系统原理的各个层面优化每一个服务节点处理单个请求的性能，最终达到提升整个系统性能的目的，不能立即达到这个目的也没关系，至少要保持服务节点持续工作，按照自身资源的限制能力对任务进行异步处理。这就是本书的两部分内容：负载均衡技术和消息队列技术所要达到的讲解效果。

1.2 可用性问题

技术人员经常会遇到这样的问题，一个已经上线的业务系统突然宕机。但是又没有可接替它继续工作的备用系统。这种情况就算单个服务节点的性能再高也没有任何意义。高可用性是系统建设的另一个方面，不是说让性能无限制地“高下去”就可以了，而且还要注意系统的容灾容错性。试想一下一个可以承受较高吞吐量峰值的业务系统，用着用着突然宕机且不能在短时间恢复了怎么办。对于客户来说这样的“高性能”是没有意义的，因为它最终也是不可用的，所以高可用性的前提是：保证服务系统能够持续工作。

实现高可用性一般有两种手段：一种是通过第三方软件/组件保证系统的可用性；另一种是软件/组件自身已具备高可用的技术实现。例如我们经常使用 Keepalived 来监控和热切换两台 LVS 节点的工作状态，保证当某台 LVS 节点宕机后（虽然实际经验中它不怎么会宕机），另一台能立刻接替它的工作。而有的分布式系统自带高可用解决方案，例如类似 HDFS 的分布