Java 程序性能优化

——让你的Java程序更快、更稳定

葛一鸣　等编著

一个优秀的程序员，不仅要会编写程序，更要会编写高质量的程序
感受Java开发中的大智慧，让你的Java程序更优美

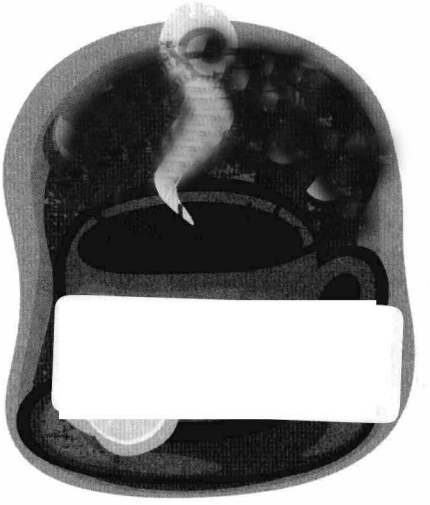
专注于Java应用程序的优化方法、技巧和思想
深入剖析软件设计层面、代码层面、JVM虚拟机层面的优化方法
理论结合实际，使用丰富的示例帮助读者理解理论知识

清华大学出版社

Java 程序性能优化

——让你的Java程序更快、更稳定

葛一鸣　等编著



清华大学出版社
北　京

内 容 简 介

Java 是目前应用最为广泛的软件开发平台，学习针对 Java 程序的优化方法有重要的现实意义。本书以 Java 性能调优为主线，系统地阐述了与 Java 性能优化相关的知识与技巧。

本书共 6 章，先后从软件设计、软件编码、JVM 调优以及程序故障排除等方面介绍针对 Java 程序的优化方法。第 1 章介绍性能的基本概念、定律、系统调优的过程和注意事项；第 2 章从设计层面介绍与性能相关的设计模式、组件。第 3 章从代码层面介绍如何编写高性能的 Java 程序；第 4 章介绍并行开发和如何通过多线程提高系统性能；第 5 章立足于 JVM 虚拟机层面，介绍如何通过设置合理的 JVM 参数提升 Java 程序的性能；第 6 章为工具篇，介绍获取和监控程序或系统性能指标的各种工具，包括相关的故障排查工具。

本书适合所有 Java 程序员、软件设计师、架构师以及软件开发爱好者，对于有一定经验的 Java 工程师，本书更能帮助突破技术瓶颈，深入 Java 内核开发！

图书在版编目（CIP）数据

Java 程序性能优化：让你的 Java 程序更快、更稳定 / 葛一鸣等编著. —北京：清华大学出版社，2012.10
ISBN 978-7-302-29625-6

Ⅰ. ①J…　Ⅱ. ①葛…　Ⅲ. ①JAVA 语言－程序设计　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2012）第 184193 号

责任编辑：夏兆彦
封面设计：欧振旭
责任校对：徐俊伟
责任印制：张雪娇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：北京国马印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：26　　**字　　数**：649 千字
版　　次：2012 年 10 月第 1 版　　**印　　次**：2012 年 10 月第1 次印刷
印　　数：1～5000
定　　价：59.00 元

产品编号：047877-01

前　言

关于 Java

Java 是目前应用最为广泛的软件开发平台之一。随着 Java 以及 Java 社区的不断壮大，Java 早已不再是简简单单的一门计算机语言了，它更是一个平台、一种文化、一个社区。

作为一个平台，JVM 虚拟机扮演着举足轻重的作用。除了 Java 语言，任何一种能够被编译成字节码的计算机语言都属于 Java 这个平台。Groovy、Scala、JRuby 等都是 Java 平台的一个部分，它们依赖于 JVM 虚拟机，同时，Java 平台也因为它们变得更加丰富多彩。

作为一种文化，Java 几乎成为了“开源”的代名词。在 Java 程序中，有着数不清的开源软件和框架，如 Tomcat、Struts、Hibernate、Spring 等。就连 JDK 和 JVM 自身也有不少开源的实现，如 OpenJDK、Harmony。可以说，“共享”的精神在 Java 世界里体现得淋漓尽致。

作为一个社区，Java 拥有无数的开发人员，数不清的论坛、资料。从桌面应用软件、嵌入式开发到企业级应用、后台服务器、中间件，都可以看到 Java 的身影。其应用形式之复杂、参与人数之庞大也令人咋舌。可以说，Java 社区已经俨然成为了一个良好而庞大的生态系统。

此外，纯粹作为一门软件开发语言，Java 非常容易学习，其学习曲线较 C++等老牌计算机语言相比，也比较平缓。因为它尽力简化或去除了 C++中许多晦涩、多余和难以理解的部分，如指针、虚函数、多继承等。

本书架构

本书主要介绍 Java 应用程序的优化方法和技巧，总共分为 6 章。

第 1 章是综述，介绍了性能的基本概念、两个重要的定律（木桶原理和 Amdahl 定律），以及系统调优的一般过程与注意事项。

第 2 章从设计层面，介绍与性能相关的设计模式、组件以及有助于改善性能的软件设计思想。

第 3 章从代码层面介绍如何编写高性能的 Java 代码。涉及的主要内容有字符串的优化处理、文件 I/O 的优化、核心数据库结构的使用、Java 的引用类型以及一些常用的惯例。

第 4 章介绍并行程序开发的相关内容，以及如何通过多线程提高系统性能。先后介绍了并发设计模式、线程池、并发数据结构的使用、并发控制方法、“锁”的优化、无锁的使用以及协程。

第 5 章立足于 JVM 虚拟机层面，介绍如何通过设置合理的 JVM 参数提升 Java 程序的性能。

第 6 章为工具篇，主要介绍获取和监控程序或系统性能指标的各种工具，以及 Java 应用程序相关的故障排查工具。

本书特点

本书的主要特点有：

- 专注于介绍 Java 应用程序的优化方法、技巧和思想，并深度剖析 JDK 的部分实现。
- 具有较强的层次性和连贯性，依次介绍了在软件设计层面、代码层面、JVM 虚拟机层面的优化方法。
- 理论结合实际，使用丰富的示例帮助读者理解理论知识。

阅读人群

要通读本书并取得良好的学习效果，要求读者具备 Java 的基本知识。本书不是一本帮助初学者入门的书籍。因此，本书适合以下读者：

- 拥有一定开发经验的 Java 开发人员；
- Java 软件设计师、架构师；
- 系统调优人员；
- 有一定的 Java 基础并希望更进一步的程序员。

本书的约定

本书在叙述过程中，有如下约定：

- 本书中所述的 JDK 1.5、JDK 1.6 分别等同于 JDK 5、JDK 6；
- 如无特殊说明，JVM 虚拟机均指 Hot Spot 虚拟机；
- 如无特殊说明，本书的程序、示例均在 JDK 1.6 环境中运行。

下载提示

本书涉及的源程序请读者直接登录清华大学出版社网站（http://www.tup.com.cn），搜索到本书页面后按照提示进行下载。

本书作者

本书由葛一鸣编著。其他参与编写和资料整理的人员有武冬、郅晓娜、孙美芹、卫丽行、尹翠翠、蔡继文、陈晓宇、迟剑、邓薇、郭利魁、金贞姬、李敬才、李萍、刘敬、陈慧、刘艳飞、吕博、全哲、余勇、宋学江、王浩、王康、王楠、杨宗芳、张严虎、周玉。

本书的写作过程远比我想象中的艰辛。为了让全书能够更清楚、更正确地表达和论述，笔者经历了无数个不眠之夜。即使现在回想起来，也忍不住让我打个寒战。但由于写作水平和写作时间的限制，书中难免会有不妥之处。为此，读者可以通过邮箱 bookservice2008@163.com 与笔者联系。

致谢

在本书的写作过程中，我充满着感激之情。首先是对我的家人，在本书完稿前，父亲病重，但我由于工作上的繁忙未能抽出太多时间照顾他，幸好得到了母亲的大力支持和父亲的谅解，我才能够鼓足勇气，全身心投入到写作之中。同时，母亲对我的悉心照料也让我能够更加专注到工作之中。

同时，我要感谢我的工作单位 UT 斯达康以及两位前辈 Rex Zhu 和 Tao Tao。正是他们在平时工作中对我的细心指导，才能让我有所进步和积累。而这一切，正是本书的基础。

最后，再次感谢我的母亲，祝她身体健康。

葛一鸣

目　录

第 1 章　Java 性能调优概述

本章对性能优化技术进行整体性的介绍，让读者了解性能的概念和性能优化的基本思路与方法。掌握这些内容，有助于读者对性能问题进行系统性的分析。

本章涉及的主要知识点有：

- 评价性能的主要指标；
- 木桶原理的概念及其在性能优化中的应用；
- Amdahl 定律的含义；
- 性能调优的层次；
- 系统优化的一般步骤和注意事项。

1.1　性 能 概 述

为什么程序总是那么慢？它现在到底在干什么？时间都花到哪里去了？也许，你经常会抱怨这些问题。如果是这样，那说明你的程序出了性能问题。和功能性问题相比，性能问题在有些情况下，可能并不算什么太大的问题，将就将就，也就过去了。但是，严重的性能问题会导致程序瘫痪、假死，直至奔溃。本节就先来认识性能的各种表现和指标。

1.1.1　看懂程序的性能

对客户端程序而言，拙劣的性能会严重影响用户体验。界面停顿、抖动、响应迟钝等问题会遭到用户不停的抱怨。一个典型的例子就是 Eclipse IDE 工具在 Full GC 时会出现程序假死现象，相信一定被不少开发人员所诟病。对于服务器程序来说，性能问题则更为重要，相信不少后台服务器软件都有各自的性能目标。以 Web 服务器为例，服务器的响应时间、吞吐量就是两个重要的性能参数。当服务器承受巨大的访问压力时，可能出现响应时间变长、吞吐量下降，甚至是抛出内存溢出异常而崩溃。这些问题，都是性能调优需要解决的。

一般来说，程序的性能通过以下几个方面来表现：

- 执行速度：程序的反映是否迅速，响应时间是否足够短。
- 内存分配：内存分配是否合理，是否过多地消耗内存或者存在泄漏。
- 启动时间：程序从运行到可以正常处理业务需要花费多长时间。
- 负载承受能力：当系统压力上升时，系统的执行速度、响应时间的上升曲线是否平缓。

1.1.2 性能的参考指标

为了能够科学地进行性能分析，对性能指标进行定量评测是非常重要的。目前，一些可以用于定量评测的性能指标有：

- 执行时间：一段代码从开始运行到运行结束，所使用的时间。
- CPU 时间：函数或者线程占用 CPU 的时间。
- 内存分配：程序在运行时占用的内存空间。
- 磁盘吞吐量：描述 I/O 的使用情况。
- 网络吞吐量：描述网络的使用情况。
- 响应时间：系统对某用户行为或者事件做出响应的时间。响应时间越短，性能越好。

1.1.3 木桶原理与性能瓶颈

木桶原理又称“短板理论”，其核心思想是：一只木桶盛水的多少，并不取决于桶壁上最高的那块木块，而是取决于桶壁上最短的那块，如图 1.1 所示。

图 1.1 木桶原理示意图

将这个理论应用到系统性能优化上，可以这么理解，即使系统拥有充足的内存资源和 CPU 资源，但是如果磁盘 I/O 性能低下，那么系统的总体性能是取决于当前最慢的磁盘 I/O 速度，而不是当前最优越的 CPU 或者内存。在这种情况下，如果需要进一步提升系统性能，优化内存或者 CPU 资源是毫无用处的。只有提高磁盘 I/O 性能才能对系统的整体性能进行优化。而此时，磁盘 I/O 就是系统的性能瓶颈。

注意：根据木桶原理，系统的最终性能取决于系统中性能表现最差的组件。因此，为了提升系统整体性能，必须对系统中表现最差的组件进行优化，而不是对系统中表现良好的组件进行优化。

根据应用的特点不同，任何计算机资源都有可能成为系统瓶颈。其中，最有可能成为系统瓶颈的计算资源如下。

- 磁盘 I/O：由于磁盘 I/O 读写的速度要比内存慢很多，程序在运行过程中，如果需要等待磁盘 I/O 完成，那么低效的 I/O 操作会拖累整个系统。
- 网络操作：对网络数据进行读写的情况与磁盘 I/O 类似。由于网络环境的不确定性，尤其是对互联网上数据的读写，网络操作的速度可能比本地磁盘 I/O 更慢。因此，如不加特殊处理，也极可能成为系统瓶颈。
- CPU：对计算资源要求较高的应用，由于其长时间、不间断地大量占用 CPU 资源，那么对 CPU 的争夺将导致性能问题。如科学计算、3D 渲染等对 CPU 需求旺盛的应用。
- 异常：对 Java 应用来说，异常的捕获和处理是非常消耗资源的。如果程序高频率地进行异常处理，则整体性能便会有明显下降。
- 数据库：大部分应用程序都离不开数据库，而海量数据的读写操作可能是相当费时的。而应用程序可能需要等待数据库操作完成或者返回请求的结果集，那么缓慢的同步操作将成为系统瓶颈。
- 锁竞争：对高并发程序来说，如果存在激烈的锁竞争，无疑是对性能极大的打击。锁竞争将会明显增加线程上下文切换的开销。而且，这些开销都是与应用需求无关的系统开销，白白占用宝贵的 CPU 资源，却不带来任何好处。
- 内存：一般来说，只要应用程序设计合理，内存在读写速度上不太可能成为性能瓶颈。除非应用程序进行了高频率的内存交换和扫描，但这些情况比较少见。使内存制约系统性能的最可能的情况是内存大小不足。与磁盘相比，内存的大小似乎小的可怜，这意味着应用软件只能尽可能将常用的核心数据读入内存，这在一定程度上降低了系统性能。

1.1.4　Amdahl 定律

Amdahl 定律是计算机科学中非常重要的定律，它定义了串行系统并行化后加速比的计算公式和理论上限。

加速比定义：加速比=优化前系统耗时/优化后系统耗时

所谓加速比，就是优化前的耗时与优化后耗时的比值。加速比越高，表明优化效果越明显。

Amdahl 定律给出了加速比与系统并行度和处理器数量的关系。设加速比为 *Speedup*，系统内必须串行化的程序比重为 *F*，CPU 处理器数量为 *N*，则有：

$$Speedup \leqslant \frac{1}{F+\frac{1-F}{N}}$$

根据这个公式，如果 CPU 处理器数量趋于无穷，那么加速比与系统的串行化率成反比，如果系统中必须有 50%的代码串行执行，那么系统的最大加速比为 2。

假设有一程序分为以下步骤执行，每个执行步骤花费 100 个时间单位。其中，只有步骤 2 和步骤 5 可以进行并行，步骤 1、3、4 必须串行，如图 1.2 所示。在全串行的情况下，系统合计耗时 500 个时间单位。

图 1.2　串行工作流程

若将步骤 2 和步骤 5 并行化，假设在双核处理上，则有如图 1.3 所示的处理流程。在这种情况下，步骤 2 和步骤 5 的耗时将为 50 个是单位。故系统整体耗时为 400 个时间单位。根据加速比的定义有：

加速比=优化前系统耗时/优化后系统耗时=500/400=1.25

或者前文中给出的加速比公式。由于 5 个步骤中，3 个步骤必须串行，因此其串行化比重为 3/5=0.6，即 F=0.6，且双核处理器的处理器个数 N 为 2。代入公式得：

加速比=1/(0.6+(1-0.6)/2)=1.25

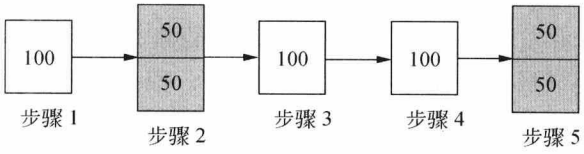


图 1.3　双核处理上的并行化

在极端情况下，假设并行处理器个数为无穷大，则有如图 1.4 所示的处理过程。步骤 2 和步骤 5 的处理时间趋于 0。即使这样，系统整体耗时依然大于 300 个时间单位。即加速比的极限为 500/300=1.67。

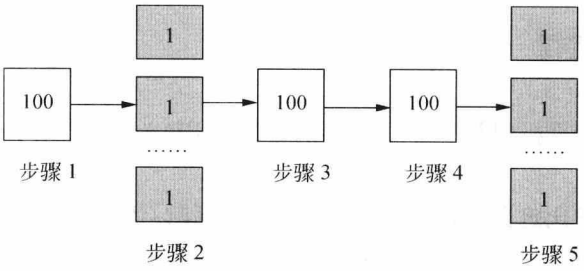


图 1.4　极端情况下的并行化

使用加速比计算公式，N 趋于无穷大，有 $Speedup$=1/F，且 F=0.6，故有 $Speedup$=1.67。

由此可见，为了提高系统的速度，仅增加 CPU 处理器的数量并不一定能起到有效的作用，需要从根本上修改程序的串行行为，提高系统内可并行化的模块比重，在此基础上，合理增加并行处理器数量，才能以最小的投入，得到最大的加速比。

注意：根据 Amdahl 定律，使用多核 CPU 对系统进行优化，优化的效果取决于 CPU 的数量以及系统中的串行化程序的比重。CPU 数量越多，串行化比重越低，则优化效果越好。仅提高 CPU 数量而不降低程序的串行化比重，也无法提高系统性能。