

O'REILLY®



Java Performance The Definitive Guide

Java 性能权威指南 (影印版)

東南大學出版社

Scott Oaks 著

Java性能权威指南 (影印版)

Java Performance: The Definitive Guide



Scott Oaks 著

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'REILLY®

O'Reilly Media, Inc. 授权东南大学出版社出版

南京 东南大学出版社

图书在版编目(CIP)数据

Java 性能权威指南: 英文/(美)奥克斯(Oaks, S.)
著. —影印本. —南京: 东南大学出版社, 2015.2
书名原文: Java Performance: The Definitive Guide
ISBN 978-7-5641-5383-0

I. ①J… II. ①奥… III. ①JAVA 语言—程序
设计—英文 IV. ①TP312

中国版本图书馆 CIP 数据核字(2014)第 294386 号

江苏省版权局著作权合同登记

图字: 10-2014-147 号

© 2014 by O'Reilly Media, Inc.

Reprint of the English Edition, jointly published by O'Reilly Media, Inc. and Southeast University Press, 2015. Authorized reprint of the original English edition, 2014 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2014。

英文影印版由东南大学出版社出版 2015。此影印版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有, 未得书面许可, 本书的任何部分和全部不得以任何形式重制。

Java 性能权威指南(影印版)

出版发行: 东南大学出版社

地 址: 南京四牌楼 2 号 邮编: 210096

出 版 人: 江建中

网 址: <http://www.seupress.com>

电子邮箱: press@seupress.com

印 刷: 常州市武进第三印刷有限公司

开 本: 787 毫米×980 毫米 16 开本

印 张: 26.75

字 数: 524 千字

版 次: 2015 年 2 月第 1 版

印 次: 2015 年 2 月第 1 次印刷

书 号: ISBN 978-7-5641-5383-0

定 价: 72.00 元

本社图书若有印装质量问题, 请直接与营销部联系。电话(传真): 025-83791830

Preface

When O'Reilly first approached me about writing a book on Java performance tuning, I was unsure. Java performance, I thought—aren't we done with that? Yes, I still work on performance of Java (and other) applications on a daily basis, but I like to think that I spend most of my time dealing with algorithmic inefficiencies and external system bottlenecks rather than on anything directly related to Java tuning.

A moment's reflection convinced me that I was (as usual) kidding myself. It is certainly true that end-to-end system performance takes up a lot of my time, and that I sometimes come across code that uses an $O(n^2)$ algorithm when it could use one with $O(\log N)$ performance. Still, it turns out that every day, I think about GC performance, or the performance of the JVM compiler, or how to get the best performance from Java Enterprise Edition APIs.

That is not to minimize the enormous progress that has been made in the performance of Java and JVMs over the past 15-plus years. When I was a Java evangelist at Sun during the late 1990s, the only real “benchmark” available was CaffeineMark 2.0 from Pendragon software. For a variety of reasons, the design of that benchmark quickly limited its value; yet in its day, we were fond of telling everyone that Java 1.1.8 performance was eight times faster than Java 1.0 performance based on that benchmark. And that was true—Java 1.1.8 had an actual just-in-time compiler, where Java 1.0 was pretty much completely interpreted.

Then standards committees began to develop more rigorous benchmarks, and Java performance began to be centered around them. The result was a continuous improvement in all areas of the JVM—garbage collection, compilations, and within the APIs. That process continues today, of course, but one of the interesting facts about performance work is that it gets successively harder. Achieving an eightfold increase in performance by introducing a just-in-time compiler was a straightforward matter of engineering, and even though the compiler continues to improve, we're not going to see an improvement like that again. Parallellizing the garbage collector was a huge performance improvement, but more recent changes have been more incremental.

This is a typical process for applications (and the JVM itself is just another application): in the beginning of a project, it's easy enough to find architectural changes (or code bugs) which, when addressed, yield huge performance improvements. In a mature application, finding such performance improvements is quite rare.

That precept was behind my original concern that, to a large extent, the engineering world might be done with Java performance. A few things convinced me I was wrong. First is the number of questions I see daily about how this or that aspect of the JVM performs under certain circumstances. New engineers come to Java all the time, and JVM behavior remains complex enough in certain areas that a guide to how it operates is still beneficial. Second is that environmental changes in computing seem to have altered the performance concerns that engineers face today.

What's changed in the past few years is that performance concerns have become bifurcated. On the one hand, very large machines capable of running JVMs with very large heaps are now commonplace. The JVM has moved to address those concerns with a new garbage collector (G1), which—as a new technology—requires a little more hand-tuning than traditional collectors. At the same time, cloud computing has renewed the importance of small, single-CPU machines: you can go to Oracle or Amazon or a host of other companies and very cheaply rent a single CPU machine to run a small application server. (You're not actually getting a single-CPU machine: you're getting a virtual OS image on a very large machine, but the virtual OS is limited to using a single CPU. From the perspective of Java, that turns out to be the same as single-CPU machine.) In those environments, correctly managing small amounts of memory turns out to be quite important.

The Java platform also continues to evolve. Each new edition of Java provides new language features and new APIs that improve the productivity of developers—if not always the performance of their applications. Best practice use of these language features can help to differentiate between an application that sizzles, and one that plods along. And the evolution of the platform brings up interesting performance questions: there is no question that using JSON to exchange information between two programs is much simpler than coming up with a highly optimized proprietary protocol. Saving time for developers is a big win—but making sure that productivity win comes with a performance win (or at least breaks even) is the real goal.

Who Should (and Shouldn't) Read This Book

This book is designed for performance engineers and developers who are looking to understand how various aspects of the JVM and the Java APIs impact performance.

If it is late Sunday night, your site is going live Monday morning, and you're looking for a quick fix for performance issues, this is not the book for you.

If you are new to performance analysis and are starting that analysis in Java, then this book can help you. Certainly my goal is to provide enough information and context that novice engineers can understand how to apply basic tuning and performance principles to a Java application. However, system analysis is a very broad field. There are a number of excellent resources for system analysis in general (and those principles of course apply to Java), and in that sense, this book will hopefully be a useful companion to those texts.

At a fundamental level, though, making Java go really fast requires a deep understanding about how the JVM (and Java APIs) actually work. There are literally hundreds of Java tuning flags, and tuning the JVM has to be more than an approach of blindly trying them and seeing what works. Instead, my goal is to provide some very detailed knowledge about what the JVM and APIs are doing, with the hope that if you understand how those things work, you'll be able to look at the specific behavior of an application and understand *why* it is performing badly. Understanding that, it becomes a simple (or at least simpler) task to get rid of undesirable (badly performing) behavior.

One interesting aspect to Java performance work is that developers often have a very different background than engineers in a performance or QA group. I know developers who can remember thousands of obscure method signatures on little-used Java APIs but who have no idea what the flag `-Xmn` means. And I know testing engineers who can get every last ounce of performance from setting various flags for the garbage collector but who could barely write a suitable "Hello, World" program in Java.

Java performance covers both of these areas: tuning flags for the compiler and garbage collector and so on, and best-practice uses of the APIs. So I assume that you have a good understanding of how to write programs in Java. Even if your primary interest is not in the programming aspects of Java, I do spend a fair amount of time discussing programs, including the sample programs used to provide a lot of the data points in the examples.

Still, if your primary interest is in the performance of the JVM itself—meaning how to alter the behavior of the JVM without any coding—then large sections of this book should still be beneficial to you. Feel free to skip over the coding parts and focus in on the areas that interest you. And maybe along the way, you'll pick up some insight into how Java applications can affect JVM performance and start to suggest changes to developers so they can make your performance-testing life easier.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at <https://github.com/ScottOaks/JavaPerformanceTuning>.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “*Java Performance: The Definitive Guide* by Scott Oaks (O’Reilly). Copyright 2014 Scott Oaks, 978-1-449-35845-7.”

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Safari® Books Online



Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world’s leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O’Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <http://oreil.ly/java-performance-tdg>.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

Acknowledgments

I would like to thank everyone who helped me as I worked on this book. In many ways, this book is an accumulation of knowledge gained over my past 15 years in the Java Performance Group at Sun Microsystems and Oracle, so the list of people who have provided positive input into this book is quite broad. To all the engineers I have worked with during that time, and particularly to those who patiently answered my random questions over the past year, thank you!

I would especially like to thank Stanley Guan, Azeem Jiva, Kim LiChong, Deep Singh, Martijn Verburg, and Edward Yue Shung Wong for their time reviewing draft copies and providing valuable feedback. I am sure that they were unable to find all my errors, though the material here is greatly improved by their input.

The production staff at O'Reilly was as always very helpful, and thanks to my editor Meg Blanchette for all your encouragement during the process. Finally, I must thank my husband James for putting up with the long nights and those weekend dinners where I was in a continual state of distraction.

Table of Contents

Preface.....	ix
1. Introduction.....	1
A Brief Outline	2
Platforms and Conventions	2
JVM Tuning Flags	4
The Complete Performance Story	5
Write Better Algorithms	5
Write Less Code	6
Oh Go Ahead, Prematurely Optimize	7
Look Elsewhere: The Database Is Always the Bottleneck	8
Optimize for the Common Case	9
Summary	10
2. An Approach to Performance Testing.....	11
Test a Real Application	11
Microbenchmarks	11
Macrobenchmarks	16
Mesobenchmarks	18
Common Code Examples	20
Understand Throughput, Batching, and Response Time	24
Elapsed Time (Batch) Measurements	24
Throughput Measurements	25
Response Time Tests	26
Understand Variability	29
Test Early, Test Often	33
Summary	36
3. A Java Performance Toolbox.....	37
Operating System Tools and Analysis	37

CPU Usage	38
The CPU Run Queue	41
Disk Usage	43
Network Usage	44
Java Monitoring Tools	46
Basic VM Information	47
Thread Information	50
Class Information	51
Live GC Analysis	51
Heap Dump Postprocessing	51
Profiling Tools	51
Sampling Profilers	52
Instrumented Profilers	54
Blocking Methods and Thread Timelines	55
Native Profilers	57
Java Mission Control	59
Java Flight Recorder	60
Enabling JFR	66
Selecting JFR Events	70
Summary	72
4. Working with the JIT Compiler.....	73
Just-in-Time Compilers: An Overview	73
Hot Spot Compilation	75
Basic Tunings: Client or Server (or Both)	77
Optimizing Startup	78
Optimizing Batch Operations	80
Optimizing Long-Running Applications	81
Java and JIT Compiler Versions	81
Intermediate Tunings for the Compiler	85
Tuning the Code Cache	85
Compilation Thresholds	87
Inspecting the Compilation Process	90
Advanced Compiler Tunings	94
Compilation Threads	94
Inlining	96
Escape Analysis	97
Deoptimization	98
Not Entrant Code	98
Deoptimizing Zombie Code	101
Tiered Compilation Levels	101

Summary	103
5. An Introduction to Garbage Collection.	105
Garbage Collection Overview	105
Generational Garbage Collectors	107
GC Algorithms	109
Choosing a GC Algorithm	113
Basic GC Tuning	119
Sizing the Heap	119
Sizing the Generations	122
Sizing Permgen and Metaspace	124
Controlling Parallelism	126
Adaptive Sizing	127
GC Tools	128
Summary	131
6. Garbage Collection Algorithms.	133
Understanding the Throughput Collector	133
Adaptive and Static Heap Size Tuning	136
Understanding the CMS Collector	140
Tuning to Solve Concurrent Mode Failures	145
Tuning CMS for Permgen	148
Incremental CMS	149
Understanding the G1 Collector	150
Tuning G1	157
Advanced Tunings	159
Tenuring and Survivor Spaces	159
Allocating Large Objects	163
AggressiveHeap	171
Full Control Over Heap Size	173
Summary	174
7. Heap Memory Best Practices.	177
Heap Analysis	177
Heap Histograms	178
Heap Dumps	179
Out of Memory Errors	184
Using Less Memory	188
Reducing Object Size	188
Lazy Initialization	191
Immutable and Canonical Objects	196
String Interning	198

Object Lifecycle Management	202
Object Reuse	202
Weak, Soft, and Other References	208
Summary	221
8. Native Memory Best Practices	223
Footprint	223
Measuring Footprint	224
Minimizing Footprint	225
Native NIO Buffers	226
Native Memory Tracking	227
JVM Tunings for the Operating System	230
Large Pages	230
Compressed oops	234
Summary	236
9. Threading and Synchronization Performance	237
Thread Pools and ThreadPoolExecutors	237
Setting the Maximum Number of Threads	238
Setting the Minimum Number of Threads	242
Thread Pool Task Sizes	243
Sizing a ThreadPoolExecutor	244
The ForkJoinPool	246
Automatic Parallelization	252
Thread Synchronization	254
Costs of Synchronization	254
Avoiding Synchronization	259
False Sharing	262
JVM Thread Tunings	267
Tuning Thread Stack Sizes	267
Biased Locking	268
Lock Spinning	268
Thread Priorities	269
Monitoring Threads and Locks	270
Thread Visibility	270
Blocked Thread Visibility	271
Summary	275
10. Java Enterprise Edition Performance	277
Basic Web Container Performance	277
HTTP Session State	280
Thread Pools	283

Enterprise Java Session Beans	283
Tuning EJB Pools	283
Tuning EJB Caches	286
Local and Remote Instances	288
XML and JSON Processing	289
Data Size	290
An Overview of Parsing and Marshalling	291
Choosing a Parser	293
XML Validation	299
Document Models	302
Java Object Models	305
Object Serialization	307
Transient Fields	307
Overriding Default Serialization	307
Compressing Serialized Data	311
Keeping Track of Duplicate Objects	313
Java EE Networking APIs	316
Sizing Data Transfers	316
Summary	319
11. Database Performance Best Practices.....	321
JDBC	322
JDBC Drivers	322
Prepared Statements and Statement Pooling	324
JDBC Connection Pools	326
Transactions	327
Result Set Processing	335
JPA	337
Transaction Handling	337
Optimizing JPA Writes	340
Optimizing JPA Reads	342
JPA Caching	346
JPA Read-Only Entities	352
Summary	353
12. Java SE API Tips.....	355
Buffered I/O	355
Classloading	358
Random Numbers	362
Java Native Interface	364
Exceptions	366
String Performance	370

Logging	371
Java Collections API	373
Synchronized Versus Unsynchronized	373
Collection Sizing	375
Collections and Memory Efficiency	376
AggressiveOpts	378
Alternate Implementations	378
Miscellaneous Flags	379
Lambdas and Anonymous Classes	379
Lambda and Anonymous Classloading	381
Stream and Filter Performance	382
Lazy Traversal	383
Summary	385
A. Summary of Tuning Flags.....	387
Index.....	397

CHAPTER 1

Introduction

This is a book about the art and science of Java performance.

The science part of this statement isn't surprising; discussions about performance include lots of numbers and measurements and analytics. Most performance engineers have a background in the sciences, and applying scientific rigor is a crucial part of achieving maximum performance.

What about the art part? The notion that performance tuning is part art and part science is hardly new, but it is rarely given explicit acknowledgment in performance discussions. This is partly because the idea of "art" goes against our training.

Part of the reason is that what looks like art to some people is fundamentally based on deep knowledge and experience. It is said that magic is indistinguishable from sufficiently advanced technologies, and certainly it is true that a cell phone would look magical to a knight of the Round Table. Similarly, the work produced by a good performance engineer may look like art, but that art is really an application of deep knowledge, experience, and intuition.

This book cannot help with the experience and intuition part of that equation, but its goal is to help with the deep knowledge—with the view that applying knowledge over time will help you develop the skills needed to be a good Java performance engineer. The goal is to give you an in-depth understanding of the performance aspects of the Java platform.

This knowledge falls into two broad categories. First is the performance of the Java Virtual Machine (JVM) itself: the way in which the JVM is configured affects many aspects of the performance of a program. Developers who are experienced in other languages may find the need for tuning to be somewhat irksome, though in reality tuning the JVM is completely analogous to testing and choosing compiler flags during compilation for C++ programmers, or to setting appropriate variables in a *php.ini* file for PHP coders, and so on.

The second aspect is to understand how the features of the Java platform affect performance. Note the use of the word *platform* here: some features (e.g., threading and synchronization) are part of the language, and some features (e.g., XML parsing performance) are part of the standard Java API. Though there are important distinctions between the Java language and the Java API, in this case they will be treated similarly. This book covers both facets of the platform.

The performance of the JVM is based largely on tuning flags, while the performance of the platform is determined more by using best practices within your application code. In an environment where developers code and a performance group tests, these are often considered separate areas of expertise: only performance engineers can tune the JVM to eke out every last bit of performance, and only developers worry about whether their code is written well. That is not a useful distinction—anyone who works with Java should be equally adept at understanding how code behaves in the JVM and what kinds of tuning is likely to help its performance. Knowledge of the complete sphere is what will give your work the patina of art.

A Brief Outline

First things first, though: Chapter 2 discusses general methodologies for testing Java applications, including pitfalls of Java benchmarking. Since performance analysis requires visibility into what the application is doing, Chapter 3 provides an overview of some of the tools available to monitor Java applications.

Then it is time to dive into performance, focusing first on common tuning aspects: just-in-time compilation (Chapter 4) and garbage collection (Chapter 5 and Chapter 6). The remaining chapters focus on best practice uses of various parts of the Java platform: memory use with the Java heap (Chapter 7), native memory use (Chapter 8), thread performance (Chapter 9), Java Enterprise Edition APIs (Chapter 10), JPA and JDBC (Chapter 11), and some general Java SE API tips (Chapter 12).

Appendix A lists all the tuning flags discussed in this book, with cross-references to the chapter where they are examined.

Platforms and Conventions

This book is based on the Oracle HotSpot Java Virtual Machine and the Java Platform, Standard Edition (Java SE), versions 7 and 8. Within versions, Oracle provides update releases periodically. For the most part, update releases provide only bug fixes; they never provide new language features or changes to key functionality. However, update releases do sometimes change the default value of tuning flags. Oracle will doubtless provide update releases that postdate publication of this book, which is current as of Java 7 update 40 and Java 8 (as of yet, there are no Java 8 update releases). When an