TURING 图灵程序设计丛书

PEARSON

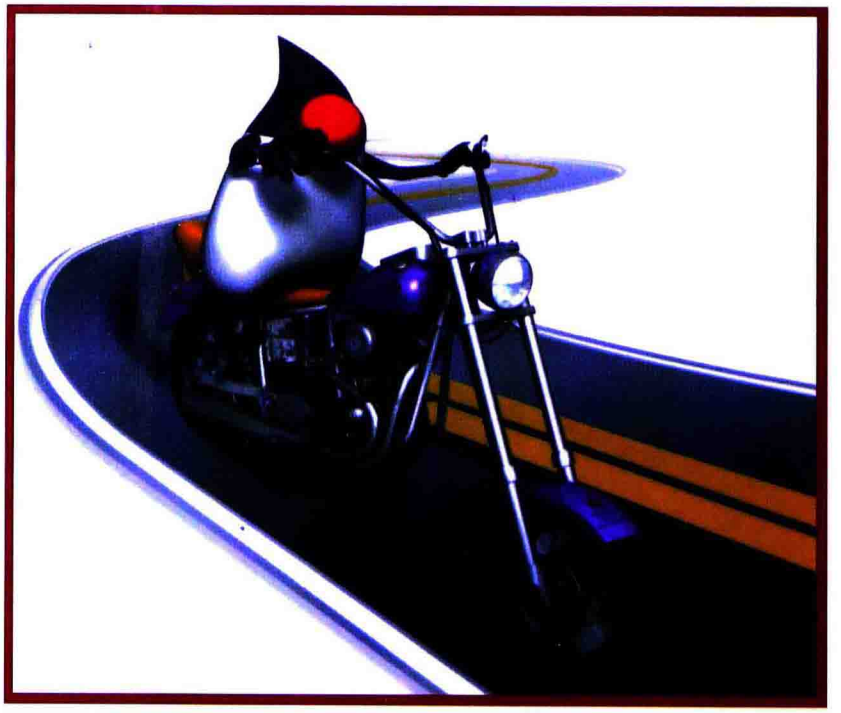
Java性能优化圣经！Java之父重磅推荐！

Java性能优化权威指南

Java Performance

[美] Charlie Hunt
Binu John 著

柳飞 陆明刚 译



人民邮电出版社
POSTS & TELECOM PRESS

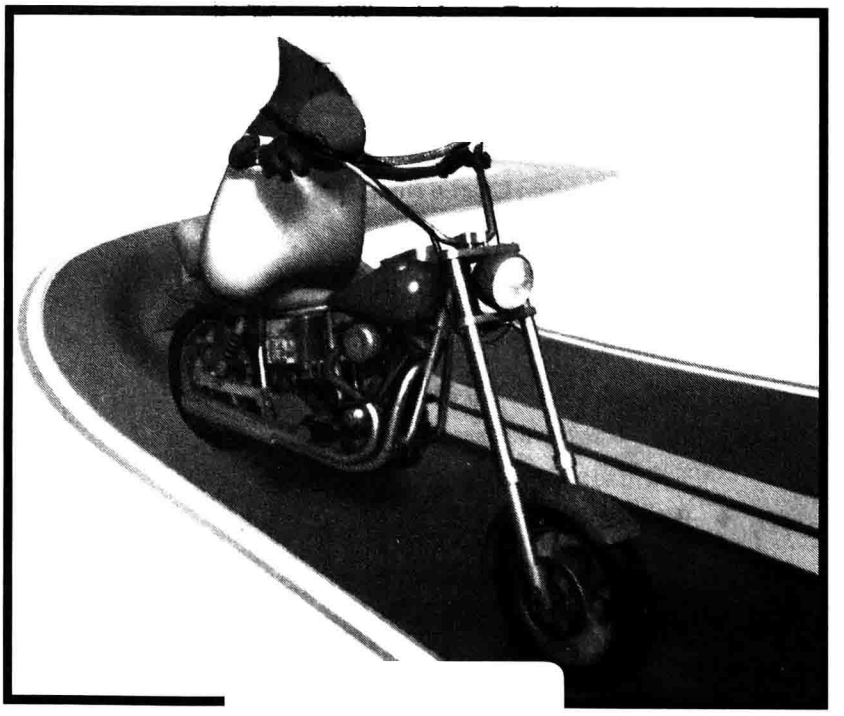
TURING 图灵程序设计丛书

Java 性能优化权威指南

Java Performance

[美] Charlie Hunt
Binu John ■ 著

柳飞 陆明刚 ■ 译



人民邮电出版社
北 京

图书在版编目（CIP）数据

Java性能优化权威指南 /（美）亨特（Hunt,C.），（美）约翰（John,B.）著；柳飞，陆明刚译. -- 北京：人民邮电出版社，2014.3
（图灵程序设计丛书）
书名原文：Java performance
ISBN 978-7-115-34297-3

Ⅰ. ①J… Ⅱ. ①亨… ②约… ③柳… ④陆… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2013)第317831号

内 容 提 要

本书主要为 Java SE 和 Java EE 应用的性能调优提供建议。主要包括以下几方面：性能监控、性能分析、Java HotSpot VM 调优、高效的基准测试以及 Java EE 应用的性能调优。

本书适合所有 Java 程序员、系统调优师和系统架构师阅读。

◆ 著 [美] Charlie Hunt Binu John
译 柳 飞 陆明刚
责任编辑 丁晓昀
责任印制 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000 1/16
印张：33.75 彩插 2
字数：799千字 2014年3月第1版
印数：1－4 000册 2014年3月河北第1次印刷
著作权合同登记号 图字：01-2011-7475号

定价：109.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第0021号

Gosling 序

现代大规模关键性系统中的 Java 性能调优，是一项富有挑战的任务。你需要关注各种问题，包括算法结构、内存分配模式以及磁盘和文件 I/O 的使用方式。性能调优最困难的是找出问题，即便是经验丰富的人也会被他们的直觉所误导。性能杀手总是隐藏在最意想不到的地方。

正如维基百科所言："科学（来自拉丁文 scientia，意思是'知识'）是以对世界可证实的解释和预见来构建和组织知识的系统。"性能调优正是这样一门实验科学，你需要构建和进行实验，然后根据实验结果建立理论假设。

所幸实验所用的性能监控工具在 Java 世界里随处可见，既有可独立运行的应用程序，也有开发环境内建的性能分析工具，还有操作系统提供的工具。综合运用这些工具，才能从数据汪洋中找出真相。

本书是 Java 应用性能调优的圣经，内容通俗易懂，介绍了大量的监控和测量工具，涉及各种硬件架构和操作系统。涵盖了如何构建实验、解释结果以及如何采取行动等技巧。如果你是一个细节控，那么这本书正适合你。

——James Gosling，Java 之父

Wilson 序

在当今世界，大型关键性计算系统的核心中总能见到 Java 的身影，而 1997 年我加入 Java 小组时，它还只是个刚刚流行起来的新平台。人们喜欢它简单的语法、可移植的字节码以及安全的垃圾收集（有别于其他系统常用 malloc/free 式的内存管理）。然而，伴随这些优秀特性的是 Java 运行比较慢，这限制了它在某些环境下的使用。

之后的几年里，我们一直致力于解决这个问题。我们相信，Java 慢并不是因为可移植性和安全性。我们重点关注了以下两方面。首先是提高 Java 平台的运行速度，通过引入高级 JIT 编译技术、并行垃圾收集和高级锁管理，核心 VM 有了巨大的改进。同时，重新整合类库使之更有效率。所有这些从根本上增强了 Java 在大型关键性系统中的运行能力。

其次，我们关注如何帮助大家编写更快的 Java 程序。虽然 Java 语法和 C 类似，但如何编写高效程序的技术却大相径庭。Jeff Kessleman 和我在 2000 年出版过一本书，这是最早关于 Java 性能调优的书籍之一。从那以后，涌现出了许多这方面的图书，有经验的开发人员也学会了如何在 Java 开发中避免一些常见的陷阱。

随着平台变得更快，开发人员掌握了如何使程序更快的技巧，Java 已经变成企业级软件平台的中流砥柱，并在一些重要的大型系统中得以应用。然而，随着 Java 应用越来越广泛，人们开始意识到它依然有所缺失，这就是**可监测性**（Observability）。当系统越来越大时，如何才能知道性能是否发挥到了极致呢？

早期 Java 的性能分析工具比较粗糙，虽然有用，但对代码的运行时性能有很大影响。如今，现代 JVM 自带监测工具，可以让人了解系统性能的关键所在，同时对系统的影响微乎其微。这意味着可以一直开启这些工具，即便在应用运行时也可以对它的许多方面进行检测。这再次改变了人们探索性能的方法。

本书的作者融合了所有这些概念，并就 Jeff 和我那本书以来十多年的所有进展做了更新和阐述。这是有史以来关于 Java 性能优化主题最雄心勃勃的一本书，包含了许多改善 Java 应用性能的技术，深入探讨了最新的 JVM 技术，连最时髦的垃圾收集算法都介绍了！你还将学会如何使用最新、最好的监控工具，包括 JDK 自带的工具和常见操作系统内嵌的重要工具。

所有这些最新进展都在推动平台持续改进，这真让人振奋。对于 Java 光辉的未来，我已迫不及待。

——Steve Wilson，Oracle 公司工程副总裁，Java 性能组创始成员，
《Java 平台性能：策略和技巧》合著者

试读结束，需要全本PDF请购买 www.ertongbook.com

前　言

欢迎翻开这本 Java 性能调优指南！

本书主要为 Java SE 和 Java EE 应用的性能调优提供建议。具体来说包括以下几方面：性能监控、性能分析、Java HotSpot VM（以下简称 HotSpot VM）调优、高效的基准测试以及 Java EE 应用的性能调优。虽然近些年出版过几本 Java 性能方面的书，但覆盖面像本书这样广的并不多见。本书的主题涵盖了诸如现代 Java 虚拟机的内部运作机制、垃圾收集的调优、Java EE 应用的性能调优以及如何编写卓有成效的基准测试。

通读本书后，读者可以深入了解 Java 性能调优的许多主题。读者也可以把本书作为参考，对于感兴趣的主题，直接跳到相应章节寻找答案。

对于 Java 性能调优的新手或者自认为初学的读者来说，最好先读前 4 章，然后可依据自己的 Java 性能调优问题，进一步阅读特定的主题或章节，这样收获最大。对于有经验的读者，他们知道基本的性能调优方法，了解 HotSpot VM 内部的基本原理，还会使用一些工具监控操作系统和 JVM 的性能，因此可以直接跳到与手头性能调优问题相关的章节，这样的效果更好。不过，即便是掌握 Java 性能调优高级技巧的读者，也仍然能从前 4 章中受益。

纵览本书，没有一招鲜式的性能调优秘笈或包罗万象的性能百科，能让你摇身一变成为老练的 Java 性能调优专家。相当数量的 Java 性能问题还需要专门的知识技能才能解决。性能调优在很大程度上是一门艺术。解决的 Java 性能问题越多，技艺才会越精湛。Java 性能调优技术仍在不断演变中，5 年前最普遍的 Java 性能问题，现在已经不是大家最关心的问题了。现代 JVM 持续演进，内建了更为成熟的优化技术、运行时技术和垃圾收集器。与此同时，底层的硬件平台和操作系统也在演化。本书包含了至编写时为止的最新内容，阅读和理解这些内容可以大大增强读者的 Java 性能调优能力，为调优艺术的登堂入室奠定基础。有了坚实的基础，性能调优的功力就会像日新月异的硬件平台、操作系统和 JVM 一样突飞猛进。

下面简单介绍一下各章的主要内容。

第 1 章“策略、方法和方法论”，介绍了 Java 性能调优实践中的各种方法、策略和方法论，并对传统软件开发过程提出了改进建议，即在软件开发中应该提前考虑软件应用的性能和可扩展性。

第 2 章“操作系统性能监控”讨论了操作系统的性能监控，介绍了操作系统中重要的监控统计信息，以及如何用工具监控这些统计信息。本章涉及的操作系统包括 Windows、Linux 及 Oracle Solaris。在其他基于 Unix 的系统（例如 Mac OS X）上监控性能统计信息时，可使用与 Linux 或 Oracle Solaris 相同或类似的命令。

第 3 章“JVM 概览”，高屋建瓴地介绍了 HotSpot VM，描述了现代 Java 虚拟机架构和运转的基本概念，并为后续的诸多章节奠定了基础。本章没有覆盖所有的 Java 性能调优问题，也没有提供 Java 性能问题所需的全部背景知识。但对于绝大多数与现代 Java 虚拟机内部机制密切相关的性能问题，本章提供了足够多的背景知识。结合第 7 章的内容，有助于你领会如何进行 HotSpot VM 调优，本章也有助于理解第 8、9 章的主题，即如何编写高效的基准测试。

第 4 章“JVM 性能监控”，顾名思义，涵盖了 JVM 性能监控的相关内容，介绍了重点需要监控的 JVM 统计数据，以及监控这些统计数据的工具。本章最后指出，这些工具扩展之后可以一并监控 JVM 和 Java 应用的统计数据。

第 5 章“Java 应用性能分析”与第 6 章“Java 应用性能分析技巧”讲述性能分析。这两章可看成第 2 章和第 4 章性能监控的补充。性能监控通常用来考察是否存在性能问题，或者为定位性能问题提供线索，告诉人们问题是出在操作系统、JVM、Java 应用程序还是其他地方。一旦发现性能问题，并进一步通过性能监控定位之后，通常就能进行性能分析了。第 5 章介绍分析 Java 方法和 Java 堆（内存）的基本技术，还推荐了一些免费工具来说明这几种性能分析技术背后所蕴藏的概念。本章提及的工具并不是性能分析仅有的手段，还有许多商业或者免费的工具也能提供类似的功能，其中一些工具的功能甚至超出了第 5 章涉及的技术范围。第 6 章提供了一些技巧，用来识别一些常见的性能分析模式，这些模式指示了一些特定类型的性能问题。本章所列的经验和技巧并不完整，却是作者在多年 Java 性能调优过程中经常碰到的。附录 B 中包含了第 6 章大部分示例的源代码。

第 7 章“JVM 性能调优入门”，涵盖了 HotSpot VM 性能调优的诸多方面，包括启动、内存占用、响应时间/延迟以及吞吐量。第 7 章介绍了调优的一系列步骤，包括选择哪个 JIT 编译器，选用何种垃圾收集器，怎样调整 Java 堆，以及如何改动应用程序以符合干系人设定的性能目标。对于大多数读者来说，第 7 章可能是本书中最有用和最值得参考的章节。

第 8 章“Java 应用的基准测试”和第 9 章“多层应用的基准测试”，探讨如何编写高效的基准测试。通常来说，基准测试是通过应用程序的功能子集来衡量 Java 应用的性能。这两章还将展示创建高效 Java 基准测试的艺术。第 8 章涵盖了与编写高效基准测试相关的较通用的主题，例如探讨现代 JVM 的一些优化方法，还介绍了如何在基准测试中运用统计方法以增强基准测试的准确性。第 9 章则重点关注如何编写高效的 Java EE 基准测试。

有些读者对 Java EE 应用的性能调优特别感兴趣，第 10 章“Web 应用的性能调优”、第 11 章“Web Service 的性能”及第 12 章“Java 持久化和 Enterprise Java Bean 的性能”，分别着重介绍了 Web 应用、Web Service、持久化及 Enterprise Java Bean 的性能分析。这 3 章会深入分析 Java EE 应用中常遇到的性能问题，并为常见的 Java EE 性能问题提供建议或解决方案。

本书还有两个附录。附录 A“重要的 HotSpot VM 选项”列举了本书所用到的 HotSpot VM 选项和其他重要的 HotSpot VM 性能调优选项，并描述了每个选项的含义，对何时可以使用这些选项给出了建议。附录 B“性能分析技巧示例源代码”包含了第 6 章示例的源代码，涉及减少锁竞争、调整 Java 集合（Collection）的初始容量以及增加并行性。

致　　谢

Charlie Hunt

如果没有那么多人的帮助，本书不可能问世。首先我要感谢合著者 Binu John，他为本书贡献了大量内容。Binu John 编写了本书中所有 Java EE 的内容。他是一名天资聪颖的 Java 性能工程师，是我的挚友。我还要感谢编辑 Greg Doech，本书从章节提纲的第一稿到我们提交全部手稿，历时将近三年，谢谢他如此耐心。感谢 Paul Hohensee 和 Dave Keenan，感谢他们的意见、激励和支持，感谢他们全面审阅了本书。感谢 Tony Printezis 和 Tom Rodriguez，感谢他们在 Java HotSpot VM 垃圾收集器和 JIT 编译器内部运作细节上的贡献。还要感谢 Java HotSpot VM 运行时团队的所有工程师，是他们编写了 HotSpot VM 各部分协作的详细文档。感谢 James Gosling 和 Steve Wilson 在百忙之中为本书作序。感谢 Peter Kessler 完整审阅了第 7 章“JVM 性能调优入门”。感谢以下人士，他们的见解和审阅提高了本书的品质，他们是 Darryl Gove、Marty Itzkowitz、Geertjan Wielenga、Monica Beckwith、Alejandro Murillo、Jon Masamitsu、Y. Srinivas Ramkakrishna（大家叫他 Ramki）、Chuck Rasbold、Kirk Pepperdine、Peter Gratzer、Jeanfrancois Arcand、Joe Bologna、Anders Åstrand、Henrik Löf 和 Staffan Friberg。感谢 Paul Ciciora 清楚地阐述了什么是“晋升失败”（losing the race，CMS 收集器无法释放更多空间以容纳新生代提升上来的对象）。另外还要感谢 Kirill Soshalskiy、Jerry Driscoll，他们是我在编写此书时的领导，还要感谢 John Pampuch（Oracle 公司 VM 技术主管），感谢他们的支持。特别感谢我的妻子 Barb 以及两个儿子，Beau 和 Boyd，感谢他们包容我的坏脾气，尤其是在我陷入写作困境时给予我的关爱。

Binu John

感谢我的合著者 Charlie Hunt，正因为他的眼界、决心和坚持不懈，本书才得以问世。他不仅编写了与 Java SE 相关的章节，还完成了出版所必要的所有其他工作。与他共事的日子里，我真的很愉快，也学会了很多。谢谢你，Charlie。特别感谢 Rahul Biswas 提供了 EJB 和 Java 持久化相关的内容，感谢他任劳任怨不厌其烦地审核并提出宝贵意见。我要感谢以下人士，是他们的帮助提升了本书的质量：感谢 Scott Oaks 和 Kim Lichong 的鼓励以及他们在 Java EE 性能诸多方面的宝贵见解；感谢 Bharath Mundlapudi、Jitendra Kotamraju 及 Roma Pulavarthi 在 XML 和 Web

服务方面的真知灼见；感谢 Mitesh Meswani、Marina Vatkina 及 Mahesh Kannan 在 EJB 和 Java 持久化上提供的帮助；感谢 Jeanfrancois Arcand 关于 Web 容器的解释、博客和评论。我的领导们都很支持这本书，我很荣幸为他们工作。谢谢我在 Sun 公司工作时的高级经理 Madhu Konda，工程、基础架构和运营的副总裁 Sef Kloninger；Ning 公司的工程与运营高级副总裁 Sridatta Viswanath。特别感谢我的孩子们 Rachael 和 Kevin 以及我美丽的妻子 Rita，感谢他们在整个过程中的支持和鼓励。

目　录

试读结束，需要全本PDF请购买 www.ertongbook.com

第1章 策略、方法和方法论

凡事预则立，不预则废，和许多事情一样，Java性能调优的成功，离不开行动计划、方法或策略以及特定领域的背景知识。为了在Java性能调优工作中有所成就，你得超越“花似雾中看”的状态，进入“悠然见南山”或者已然是“一览众山小”的境界。

这3个境界的说法可能让你有些糊涂吧，下面进一步解释。

- **花似雾中看**（I don't know what I don't know）。有时候你的任务会涉及你所不熟悉的问题域。理解陌生问题域首先面临的困难就是如何竭尽所能地了解它，因为你对它几乎一无所知。对于这类问题域，你有许多东西不了解，或者不知道重点。换句话说，这个问题域有哪些东西需要了解，你还傻傻看不清楚。这个阶段就是“花似雾中看”。
- **悠然见南山**（I know what I don't know）。刚进入不熟悉的问题域时，你对它知之甚少，随着时间的推移，你对它的许多重要方面都已有所认识，只是对重要的具体细节还缺乏了解。这时，你可以算是刚刚“见南山”。
- **一览众山小**（I already know what I need to know）。还有些时候，你对任务的问题域非常熟悉，或者已经具有该领域所必备的技能和知识，是这方面的专家。或者你对问题域足够了解，处理起来得心应手，比如你已经掌握了必要的知识，解决问题游刃有余。如果达到这个境界，那就意味着你已经是“一览众山小”了。

如果你已经或打算购买本书，说明可能还没有“一览众山小”，除非你只是想手边有本不错的参考手册。如果还在“花似雾中看”，本章将有助于你找到那些不了解的内容，以及应对Java性能问题的方法或策略。那些“悠然见南山”的读者也能从本章获得有用的信息。

本章首先讨论了性能问题的现状，并建议将性能调优集成到软件开发过程中，接着探讨了两种不同的性能调优方法——自顶向下和自底向上。

1.1 性能问题的现状

通常认为，传统的软件开发过程主要包括4个阶段：分析、设计、编码和测试，如图1-1所示。

分析是开发过程的第一步，用于评估需求、权衡各种架构的利弊以及构思高层抽象。设计则依据分析阶段的基本架构和高层抽象，进行更精细的抽象并着手考虑具体实现。编码自然就

是设计的实现。编码之后是测试，用以验证实现是否合乎应用需求。值得注意的是，测试阶段通常只包括功能测试，即检验应用的执行是否合乎需求规格。一旦测试完成，应用就可以发布给客户了。

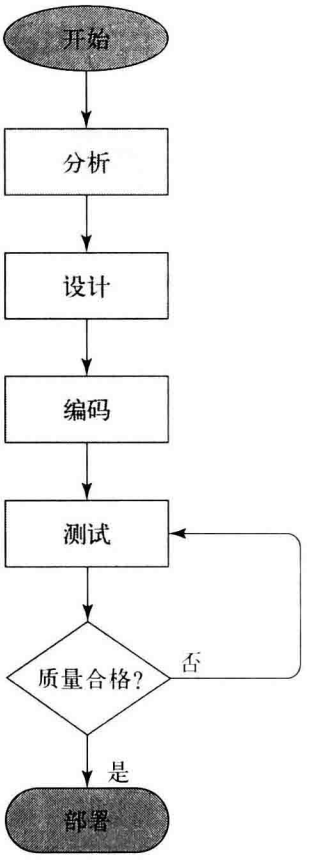


图1-1　传统软件开发过程

遵循这种传统软件开发过程的应用，通常要到测试或即将发布时才会关注性能或扩展性。为了解决这个问题，Wilson和Kesselman对传统软件开发过程做了些补充，即在传统开发模型基础上引入了性能测试分析阶段，参见他们的畅销书*Java Platform Performance*。他们建议在测试阶段之后增加性能测试，并将“性能测试是否通过”设定为产品是否发布的标准。如果达到性能和扩展性标准，应用就可以发布，否则就要转向性能分析，并依据分析结果回到之前的某个或者某些步骤。换句话说，通过性能分析来定位性能问题。Wilson和Kesselman添加的性能测试分析如图1-2所示。

对分析阶段提炼出来的性能需求，Wilson和Kesselman建议以用例的方式特别标识出来，这有助于在分析阶段制定性能评估指标。不过应用的需求文档中通常都不会明确描述性能或扩展性需求。如果你正在开发的应用还没有明确定义这些需求，那就应该想办法将它们挖掘出来。以吞吐量和延迟性需求为例，下面的清单列举了挖掘这些需求所要考虑的问题。

- 应用预期的吞吐量是多少？
- 请求和响应之间的延迟预期是多少？
- 应用支持多少并发用户或者并发任务？
- 当并发用户数或并发任务数达到最大时，可接受的吞吐量和延迟是多少？
- 最差情况下的延迟是多少？
- 要使垃圾收集引入的延迟在可容忍范围之内，垃圾收集的频率应该是多少？

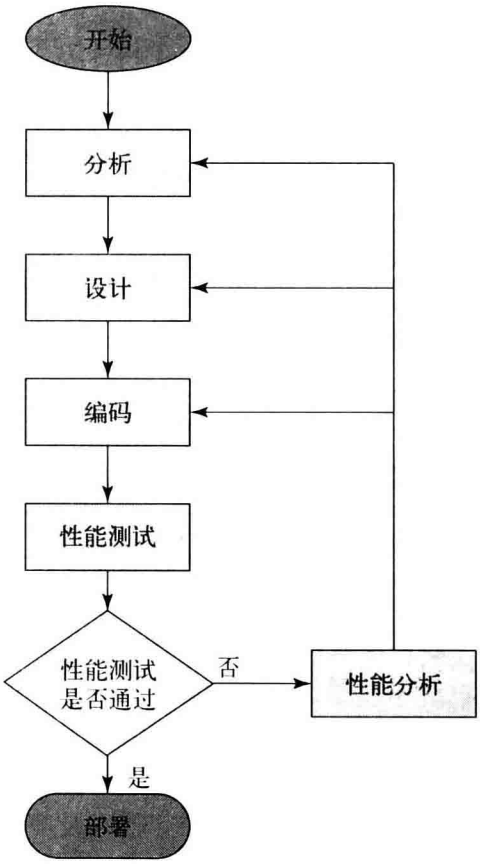


图1-2 Wilson和Kesselman添加性能测试分析之后的软件开发过程

需求和对应的用例文档应该回答上述问题，并以此制定基准测试和性能测试，确保应用能够满足性能和扩展性需求。基准测试和性能测试应该在性能测试阶段执行。有些用例在评估时发现风险很高，难以实现，就应该在结束分析阶段之前，通过一些原型、基准测试和微基准测试来降低这类风险。分析结束后再变更决策的代价非常高，这个方法可以让你事先对决策进行评估。软件开发周期中的软件缺陷、低劣设计和糟糕实现发现得越晚，修复的代价就越大，这是一条颠扑不破的金科玉律。降低用例的高风险有助于避免这些代价昂贵的错误。

现在许多应用在开发过程中都会使用自动构建和测试。Wilson和Kesselman建议改进软件开发过程，在自动构建或测试中进一步添加自动性能测试。自动性能测试可以发出通知，比如用电子邮件将性能测试结果（譬如性能是衰减还是改善、性能指标的达成度）发送给干系人。这个过

试读结束，需要全本PDF请购买 www.ertongbook.com