# 《JAVA 性能调优》

翻译：clark huang

Table of Contents

[《JAVA 性能调优》 1](#_Toc461098851)

[第一章 策略，处理，方法论。 4](#_Toc461098852)

[武装力量 4](#_Toc461098853)

[两种处理方式，由上而下和自下而上 6](#_Toc461098854)

[选择一个正确的平台，评估系统的参考资料 7](#_Toc461098855)

[第二章 操作系统监视器 8](#_Toc461098856)

[定义 8](#_Toc461098857)

[CPU使用 8](#_Toc461098858)

[CPU计划运行任务队列 8](#_Toc461098859)

[内存使用 8](#_Toc461098860)

[IO网络使用 8](#_Toc461098861)

[磁盘IO使用 8](#_Toc461098862)

[附带命令行工具 8](#_Toc461098863)

[在SPARC T系列系统CPU使用监控 8](#_Toc461098864)

[手册 8](#_Toc461098865)

[第三章 JVM总览 8](#_Toc461098866)

[热点VM高级别的架构 8](#_Toc461098867)

[热点VM运行时 8](#_Toc461098868)

[热点VM垃圾回收机制 8](#_Toc461098869)

[热点VMJIT编译器 8](#_Toc461098870)

[热点VM适应性调协 8](#_Toc461098871)

[引用 8](#_Toc461098872)

[第四章 JVM性能检测器 8](#_Toc461098873)

[定义 8](#_Toc461098874)

[垃圾回收 8](#_Toc461098875)

[JIT 编译器 8](#_Toc461098876)

[类加载器 8](#_Toc461098877)

[java 程序检测器 8](#_Toc461098878)

[引用 8](#_Toc461098879)

[第五章 java程序切面图 8](#_Toc461098880)

[专业术语 8](#_Toc461098881)

[Oracle Solaris Studio 性能解析器 8](#_Toc461098882)

[NetBeans 性能解析器 8](#_Toc461098883)

[引用 8](#_Toc461098884)

[第六章 Java程序切面图的一些好的建议和技巧 8](#_Toc461098885)

[表现机会 8](#_Toc461098886)

[系统以及内核CPU使用 8](#_Toc461098887)

[Lock锁的内容 8](#_Toc461098888)

[Volatile 使用 8](#_Toc461098889)

[数据结构尺寸重定义 8](#_Toc461098890)

[增加并行 8](#_Toc461098891)

[高CPU使用 8](#_Toc461098892)

[其他的一些分析器的使用 8](#_Toc461098893)

[引用 8](#_Toc461098894)

[第七章 调优JVM 9](#_Toc461098895)

[方法论 9](#_Toc461098896)

[程序的系统要求 9](#_Toc461098897)

[系统级别要求 9](#_Toc461098898)

[选择JVM部署模型 9](#_Toc461098899)

[选择JVM运行 9](#_Toc461098900)

[GC调优基础 9](#_Toc461098901)

[决定内存足迹 9](#_Toc461098902)

[调整潜在因素和职责 9](#_Toc461098903)

[调整程序生产量 9](#_Toc461098904)

[特别事件 9](#_Toc461098905)

[另外在命令行表现选项 9](#_Toc461098906)

[引用 9](#_Toc461098907)

[第八章 Java 程序的基本守则 9](#_Toc461098908)

[对基准点的挑战 9](#_Toc461098909)

[设计实验 9](#_Toc461098910)

[使用静态方法 9](#_Toc461098911)

[引用 9](#_Toc461098912)

[攻略 9](#_Toc461098913)

[第九章 多应用的基准点 9](#_Toc461098914)

[基准点的挑战 9](#_Toc461098915)

[企业级基准点的考量 9](#_Toc461098916)

[程序服务监测 9](#_Toc461098917)

[监测企业级程序 9](#_Toc461098918)

[引用 9](#_Toc461098919)

[第十章 web程序的表现 9](#_Toc461098920)

[web程序的基准点 9](#_Toc461098921)

[web容器的组件 9](#_Toc461098922)

[web容器检测器以及调优 9](#_Toc461098923)

[联系 9](#_Toc461098924)

[引用 9](#_Toc461098925)

[第十一章 web services 的表现 9](#_Toc461098926)

[xml表现 9](#_Toc461098927)

[验证 9](#_Toc461098928)

[获取外部实体 9](#_Toc461098929)

[对xml文件的部分处理 9](#_Toc461098930)

[选择正确的API 9](#_Toc461098931)

[JAX－WS对栈的实现 9](#_Toc461098932)

[Web services 基准点 9](#_Toc461098933)

[影响web service表现的因素 10](#_Toc461098934)

[表现实践 10](#_Toc461098935)

[引用 10](#_Toc461098936)

[第十二章 Java 持久层遗迹企业级别java bean的表现形式 10](#_Toc461098937)

[EJB编程模型 10](#_Toc461098938)

[JAVA持久层API以及相关引用实现 10](#_Toc461098939)

[监测和调试EJB容器 10](#_Toc461098940)

[事务隔离级别 10](#_Toc461098941)

[最佳实践企业级java bean 10](#_Toc461098942)

[最佳实践对java持久化 10](#_Toc461098943)

[引用 10](#_Toc461098944)

## 第一章 策略，处理，方法论

在java性能调试的过程中，正如其他很多的事情一样，你需要一个行动的计划，一个处理入口，一种策略。并且需要一些背景知识来保证成功。为了在java性能调优中能够成功，你需要从“我不知道我不知道什么”的阶段，进入“我知道什么我不知道”，或者已经到了“我已经知道我需要知道什么”的阶段。

如果你发现自己不太清楚这三个阶段的定义，下面有详细的解释。

* **我不知道我不知道什么。**有时候你被分派一个任务涉及到了一个你从未涉略的领域。第一个挑战就是理解这个新的领域，尽可能的学习这个新领域你需要解决的问题，因为你对这个领域可能知道的很少。在这个新领域中可能有很多的细枝末节你不太明白，或者说你不知道什么是重要的需要掌握的东西，因此说，我不知道我有什么不知道。
* **我知道我有什么不知道。**通常来说当你进入新的领域的时候，或者你知道很少的关于这个领域的时候，你最中到达了一个点，找到了这个领域需要了解的很重要的几个点。但是对于这几个点中具体的细节不是很明白。这个阶段称之为“我知道我不知道什么”
* **我已经知道了我需要知道的。**另一个事件你被分配了一个你很熟悉的问题。这个时候你已经学习了这类问题的基本上所有的解决方案。这个时候我们说你“已经知道你需要知道的”

看你是正在看这本书，或者是考虑买这本书，你基本上过了我不知道我不知道什么的阶段。如果你正在这个阶段，那么这个章节可能活帮助你发现你的问题。哪些在我知道我不知道什么阶段的同学可能获得这个章节十分的有用。

这个章节首先从武装力量着手去讨论性能调优，并且建议在软件开发过程中从高级别的角度进行调性能。然后看另外的两个话题，自下而上，以及自上而下

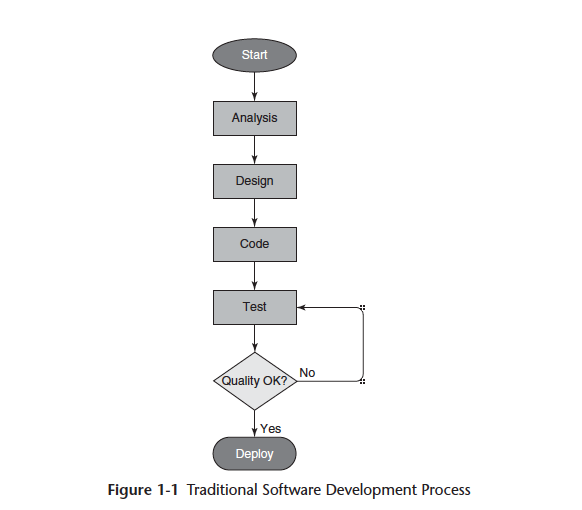
### 武装力量

传统的软件开发流程包含以下的四个步骤：需求分析，设计，编码，测试。这几个阶段的流程图可以看图1-1.

需求分析阶段是评估需求，权衡架构选择，高级别的抽象。设计阶段，衔接第一阶段的架构以及抽象，然后将这些抽象变成具体的实现概念。编码阶段，这个阶段完成设计阶段的是设计目的。测试阶段，完成需求测试。通常情况下的功能测试一点用都是没有的。程序是否完成了指定的任务，以及它执行了他需要执行的程序了吗。一旦测试用例完成后，程序就会交给客户。

大部分的传统软件开发商很少对性能以及可扩展性有所考虑在程序被交接的时候，或者是在前期的测试阶段的时候。威尔逊以及克斯曼在他们的书《java性能》这本书中介绍了在传统流程的基础上添加一个额外的分支“性能接受度”。如果性能和扩展性满足要求，那么才是一个可递交的程序。否则，工作流就需要向后转移。转移的程序根据调优的活动决定。换句话说，性能输出标志着性能问题。威尔逊以及克斯曼的流程图如1-2.

为了完成测试阶段的性能关键性问题，威尔逊以及克斯曼提出在分析阶段提供具体的使用案例用于对特殊性能的需要的界定。但是在实际过程中却常常没有被很好的贯彻执行。如果你接手的需求分析没有对性能一块做出规范你应当主动去要求。



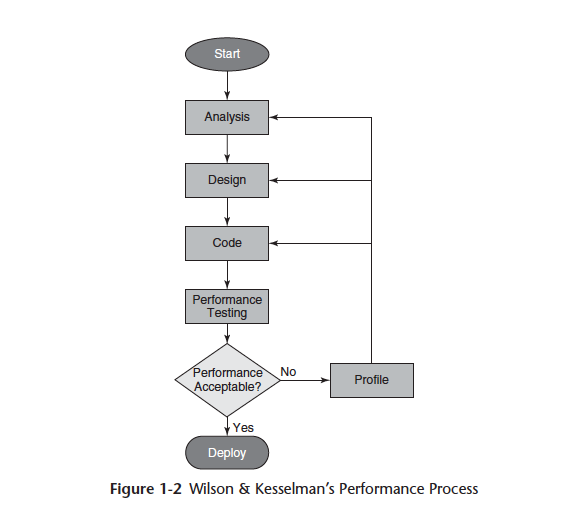
例如，你应该询问生产能力以及潜在的需求。下面是一个列表，你可能需要问到的问题。

* 这个程序期望的生产能力是怎样的？
* 对于刺激物以及反应刺激物的潜在需求是什么？
* 多少并发用户以及并发任务这个程序需要支持？
* 最大并发量／用户情况下的生产能力以及潜在需求是什么？
* 对程序的最大容忍程序？
* gc的回收频率的最大容忍度？

对于上述问题的回答可以作为完成需求以及性能测试的和开发的基准点，从而使得开发出来的程序满足表现能力以及扩展能力。这些基准点以及表现测试应该作为性能测试的一个重要环节来执行。当你在评估使用案例的有一些可能被考虑成是高风险的。这些可能很难遇到。高风险的案例应当被放缓，在完成分析阶段之前。通过一些原型以及基准点，以及超小基准点。这种处理方式让你能够攥住一些很疼的决定这些昂贵的决定一单在分析阶段决定了在开发阶段修改起来就十分的昂贵了。众所周知，越是往后的阶段发现问题，那么就该起来的代价也就越大。延缓高风险的用户用例帮助避免这些风险。

今天许多程序都会创建自动化的构建以及测试程序。作为结果，强化的软件开发流程进一步被提升。加入了自动化的测试作为自动构建以及测试活动领域。自动化测试工具能够通过email的形式将测试结果发送到指定人员通知它们测试结果。自动化测试程序也能够监测以及跟踪系统状态自动化的监测数据以及表现。

加入自动化测试程序能够让性能问题尽快的显示出来。另外一个值得考虑的因素将自动化测试集成道系统中的就是实用金泰方法和自动规划静态变量的分析。可以提高性能效果结果的反馈。



### 

### 两种处理方式，由上而下和自下而上

有两种常见的被普遍接受的方式进行性能分析：从上而下或者从下而上。从上而下，正如条件暗示的那样，集中在从高级别看引用程序，并且向下挖掘存在的问题以及优化的可能性。相反，从下而上，从最底层软件栈中开始分析，从CPU级别看数据，例如CPU缓存损失，对CPU的未有效利用，然后整理出软件中使用的那些结构体以及功能。从上而下的方式通常被程序开发者使用到。从下而上通常被使用在对性能表现在具体的环境中被要求的时候，比如不同平台架构，不同系统不同JVM中的程序的表现形式。

自上而下方式

对于性能调优最为常见的使用方式。这种方式也同在在你有能力从最该级别修改代码的时候使用。

通过这种方式，通常是在发现了一个性能问题的时候开始的。也有可能是在持续的对引用监测的过程中，性能逐渐变弱。也可能是因为需求对性能以及扩展能力的要求增加所以开始的。

不论是哪种原因导致的性能调优。将程序运行起来是自上而下的第一步。这种性能监测活动肯跟或观察操作系统级别的数据。JVM数据，JAVA EE 容器的数据。或者是程序提供的数据。然后根据这些监测数据建议你开始一步步的进行调优。JVM 的GC JVM的命令行选项调试操作系统，或者是调试程序。调优程序可能会导致一些实现的改变，识别出一些第三方库的非有效性。或者对识别出某个方法或者类的效率低下问题。

自下而上方式

这种方式最长用于对性能有特殊具体要求的当你想要特深表现对一个平台但是它和底层的CPU有关联，CPU的架构，CPU的数量。自下而上的方式也备用放你想要将程序适用于不同的操作系统的时候，

这种方式中，对表现数据的收集和监测活动开始在最底层CPU。数据监测在CPU级别，可能包含CPU数量（路径长度），CPU缓存丢失的数量。这两个点的查询是自下而上分析最为常见的点。通过提神CPU数量以及减少CPU缓存丢失是提高程序效率的很好的做法。因为CPU缓存丢失导致浪费CPU等待获取资源从内存中。

通过自上而下的方式提高性能主要是可以在不改变程序的条件下完成性能的提升。当然程序也是有肯跟被修改的。这些修改包含程序源码的修改例如：删除频繁的数据获取，很近的操作可以从CPU缓存中获取，从而减少从内存中获取数据的等待，这样的一个改变能够减少CPU缓存丢失因此减少CPU等待时间用于从内存中的数据获取。

Java程序执行在JVM中并且与之相关的还有一个叫做JIT的编译器，可能导致实现的优化问题。例如：可以创造更有效的机器码通过内存获取模型通过程序去重构。这些能够被设定通过操作系统级别，也可通过调整CPU的计划任务算法。或者对特定程序的执行时间就行修改。等待时间修改等等。

如果你发现自己的当前环境使用自下而上的方式更有效的话。你就可以开始手机系统数据和JVM数据。监测这些数据提供暗示。

### 选择一个正确的平台，评估系统的参考资料

有时，一个性能要求被提出来，发现程序运行在不合适的CPU系统架构上。CPU架构和系统在这些年都有很大的提升一个CPU多核心的和多哥CPU多核的。选择一个正确的平台和CPU架构对于一个程序的表现是十分明显的要求的。

## 第二章 操作系统监视器

对于客户来说，知道程序出现表现能力以及扩展能力不如预期是十分重要的。因为这个是作为客户指定服务计划的很重要的依据。因此，知道如何进行监测，用什么工具进行监测，在软件栈中的那个地方进行监测都是十分关键的。本章描述在操作系统级别有哪些是需要进行监测的，以及使用什么工具进行监测。另外来说，常见的用法也给出了用于解决常见的性能问题。本章对于操作系统包括windows，linux，oracle solaris。对于监测工具，只是针对不同系统有选择性的介绍了一个适合的，并不是为了穷举所有的工具。

### 定义

我们在进行性能调优的时候，一般分为三个步骤，性能监测，性能分析，性能调试。

* 性能监测是通过无干扰的一些动作对一个运行的程序进行数据监测。监测通常是一种预防措施，通常在生产环境，质量评估环境，或者是开发环境进行。也可能是客户在爆出一个性能问题时候，但是没有找到具体原因的第一步。
* 性能分析是对程序的操作数据，出口量等等进行分析。性能分析和性能监测一样也可能是由客户发现问题推动的。并且相对于性能监测，可能关注点更加的集中。性能分析通常不是在生产环境做的，通常实在测试环境或者是开发环境进行的。并且通常是作为性能监测后发现问题的手段。
* 性能调优和性能监测和性能分析不同是一种对源代码配置属性的修改来改变程序的输出量。性能调优通常在前两个步骤的后面。

### CPU使用

对于一个程序要他达到它最佳的表现和扩展，我们不仅要最大限度的利用CPU周期，并且让CPU不至于陷入浪费。在多线程程序运行多核心多CPU的系统中还是很难优化的。另外，要注意一个程序对CPU的饱和使用不一定表示就已经尽可能的用尽了CPU的所有的性能的和扩展。为了鉴定CPU的使用，你应该操作系统级别监控CPU。CPU的使用报告通常在CPU使用手册或者内核泗洪上都有报告。用户CPU使用报告表示当前时间程序的使用。相反，内核或者是系统CPU报告表示执行系统内核代码程序的使用情况。高内核或者系统CPU的使用可能是由于大量的IO交互设备导致的。最佳的情况是对系统内核的CPU使用尽可能的少。因此减少内核以及系统对CPU的使用是提高CPU效率的有效手段。

对程序的监测可能相对来说比对CPU的使用的监测更加的重要。在计算密集型的系统中，进一步的对CPU数量，每次CPU周期的监测是需要的。

监测windows系统的CPU使用

### CPU计划运行任务队列

### 内存使用

### IO网络使用

### 磁盘IO使用

### 附带命令行工具

### 在SPARC T系列系统CPU使用监控

### 手册

## 第三章 JVM总览

### 热点VM高级别的架构

### 热点VM运行时

### 热点VM垃圾回收机制

### 热点VMJIT编译器

### 热点VM适应性调协

### 引用

## 第四章 JVM性能检测器

### 定义

### 垃圾回收

### JIT 编译器

### 类加载器

### java 程序检测器

### 引用

## 第五章 java程序切面图

### 专业术语

### Oracle Solaris Studio 性能解析器

### NetBeans 性能解析器

### 引用

## 第六章 Java程序切面图的一些好的建议和技巧

### 表现机会

### 系统以及内核CPU使用

### Lock锁的内容

### Volatile 使用

### 数据结构尺寸重定义

### 增加并行

### 高CPU使用

### 其他的一些分析器的使用

### 引用

## 第七章 调优JVM

### 方法论

### 程序的系统要求

### 系统级别要求

### 选择JVM部署模型

### 选择JVM运行

### GC调优基础

### 决定内存足迹

### 调整潜在因素和职责

### 调整程序生产量

### 特别事件

### 另外在命令行表现选项

### 引用

## 第八章 Java 程序的基本守则

### 对基准点的挑战

### 设计实验

### 使用静态方法

### 引用

### 攻略

## 第九章 多应用的基准点

### 基准点的挑战

### 企业级基准点的考量

### 程序服务监测

### 监测企业级程序

### 引用

## 第十章 web程序的表现

### web程序的基准点

### web容器的组件

### web容器检测器以及调优

### 联系

### 引用

## 第十一章 web services 的表现

### xml表现

### 验证

### 获取外部实体

### 对xml文件的部分处理

### 选择正确的API

### JAX－WS对栈的实现

### Web services 基准点

### 影响web service表现的因素

### 表现实践

### 引用

## 第十二章 Java 持久层遗迹企业级别java bean的表现形式

### EJB编程模型

### JAVA持久层API以及相关引用实现

### 监测和调试EJB容器

### 事务隔离级别

### 最佳实践企业级java bean

### 最佳实践对java持久化

### 引用