基于 AOP 的 Java 程序性能跟踪系统的设计与实现

陈琪[1,2]，王晶[1,2]，李炜[1,2]

（1. 北京邮电大学网络与交换技术国家重点实验室，北京 100876;
2. 东信北邮信息技术有限公司，北京 100191）

摘要：Java 语言的免费、跨平台、开发效率高等特点，使得 Java 语言已经广泛应用到软件开发的各个领域，但是 Java 软件体系结构决定了 Java 的执行效率低，这成为了制约 Java 语言进一步发展的最重要原因。因此，对 Java 程序进行性能优化是十分必要的。对程序的性能改善是依赖于运行时性能数据的，没有可靠的数据基础而更改应用或环境会导致更差的结果。本文提出了 Java 程序性能跟踪系统的设计与实现，可以对 Java 运行时的数据进行实时的采集，提供细粒度的性能分析。本文还采用了 AOP (Aspect—Oriented Programming) 的编程方式，将性能跟踪功能从主要业务功能代码中剥离出来，提高系统的可维护性与可扩展性。

关键词：计算机软件与理论；AOP；性能跟踪；Java 程序
中图分类号：TP311.1

Design and Implementation of Java Program's Performance Tracking System Based on AOP

Chen Qi[1,2], Wang Jing[1,2], Li Wei[1,2]

(1. State Key Lab of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876;
2. EBUPT Information Technology Co.,Ltd., Beijing 100191)

Abstract: The Java language's advantages, free、cross-platform、high efficient development、make it has been widely useed in all areas of software development. However, Java's system structure determines its low operation efficiency, which limits further expansion of Java. Therefore, it is necessary to optimize Java program's performance. To improve the performance of the program, it depends on the runtime performance data. Without reliable data, if we change the application or environment, it can cause worse results. This paper describes the design and implementation of Java performance tracking system, which can collect Java runtime data, for fine-grained analysis of performance. Meanwhile, in order to reduce code coupling and improve maintainability and scalability, this paper uses AOP (Aspect—Oriented Programming) to strip the tracking functions off the main business functions.

Keywords: Computer Software and Theory; AOP; performance tracking; Java program

0 引言

Java 语言由于其跨平台特性、动态性、安全性、健壮性以及开发效率高等特点，已经得到了越来越广泛的应用。但由于 Java 软件架构决定了 Java 语言执行效率较慢，这已经成为 Java 语言最主要的缺点[1]。因此，对 Java 语言进行有效的性能优化是十分重要的。

常用的 Java 性能优化，主要是调整 JVM（Java Virtual Machine）参数、系统参数、或者扩展硬件设备等方式实现，这些优化方式都是从系统周边环境的角度进行考虑的，成本过

基金项目：国家 973 计划项目（No. 2012CB315802）；国家自然科学基金（No. 61072057，60902051，61101119）；长江学者和创新团队发展计划资助；国家科技重大专项（No. 2011ZX03002-001-01，移动互联网总体架构研究）；中央高校基本科研业务费专项资金（BUPT2009RC0505）
作者简介：陈琪，（1987-），男，硕士，主要研究方向：业务网络智能化。
通信联系人：王晶，（1974-），女，副教授，主要研究方向：业务网络智能化. E-mail: wangjing@bupt.edu.cn

高且效果有限。其实造成性能低下的很大一部分原因是由程序结构和算法的不合理，对程序结构和算法进行有效的优化，将极大的提高软件的性能。如果不了解 Java 代码在 JVM 中是如何运行的，又或者各执行的请求是如何进行处理的，其处理的过程及处理步骤处于什么状态下，就盲目开展优化工作，往往起到事倍功半的效果。

常用的性能分析手段包括以下两种：

1、 使用HPROF、Jprofiler等性能跟踪工具，但是这些工具的额外开销过大，且性能跟踪的粒度仍然不够细。

2、 Java程序员最常用的性能跟踪手段是通过向标准输出打印或者日志等手段，记录下当前运行时间或者状态。但这种方式有以下两个缺点：

- ✓ 这是一个手动添加代码的过程，需要开发人员去分析在哪儿记录代码；需要重新编译、部署、运行和分析结果；随着需求的不断改变，又要不断的重复以上这个过程，非常繁琐。
- ✓ 对应用程序各部分的执行情况仍然没有提供足够细粒度的观察。

如果能够灵活自动的插入记录代码，上述第二种方式的不足就不存在了，采用 AOP(Aspect—Oriented Programming)技术可以很好的解决这类问题，利用 AOP 将性能跟踪这个横切关注点从业务逻辑核心关注点中分离出来[2]，然后各自独立的实现这些关注点，形成相应的功能组件。最后利用 AOP 编译器将两者重新编织到一起。

1 背景介绍

1.1 AOP 简介

AOP 是 Spring 框架中的一个重要内容[3]，其核心思想如图 1 所示。

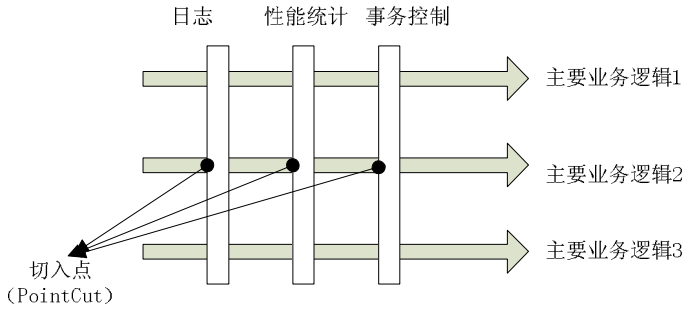


图 1 AOP 概念图

将日志、性能统计等功能独立出来，抽象成一个个切面（Aspect），最后再通过一组组切入点（PointCut）将切面的功能织入到主要业务逻辑里。通过这种方式，软件开发人员将更多的精力放在主要业务逻辑上，同时提高程序的可维护性与可扩展性。

1.2 Java Instrument

AOP 是一个概念，并没有设定具体语言的实现。在 Java 语言里，可以通过 Java Instrument 的接口来实现 AOP。Java Instrumentation 是 Java 1.5 开始提供的合法 AOP 方式。开发者可以构建一个独立于应用程序的代理程序（Agent），用来监测和协助运行在 JVM 上的程序，甚至能够替换和修改某些类的定义。有了这样的功能，开发者就可以实现更为灵活的运行时虚拟机监控和 Java 类操作了，这样的特性实际上提供了一种虚拟机级别支持的 AOP 实现方式，使得开发者无需对程序做任何升级和改动，就可以实现某些 AOP 的功能了。

但 Java Instrumentation 是直接对 Java 的类字节码进行操作的，由于字节码的晦涩难懂，类字节码的修改将会十分的麻烦。因此，可以使用 ASM[4]这个工具来对字节码进行读取和

修改。ASM 是一个 Java 字节码操控框架，它能够在类被加载入 Java 虚拟机之前动态改变类的字节码，从而改变类的行为，并对外提供了 API 供开发者修改字节码。相比于其他修改字节码的工具，ASM 具有占用额外空间小，运行速度更快等优点，因此在本课题中，将采用 Java Instrumentation+ASM 的方式来实现 AOP。

2 设计框架

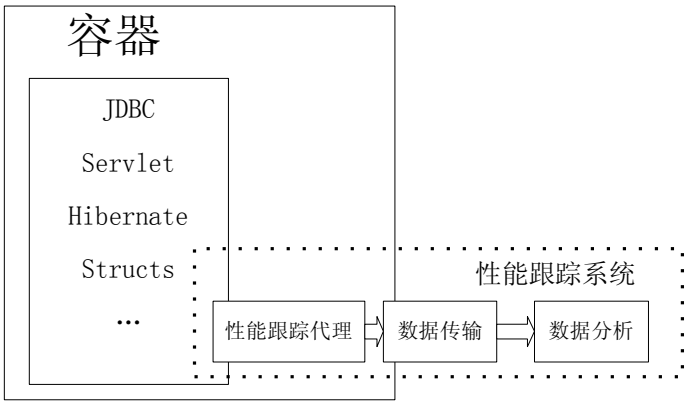


图 2 设计框架图

性能跟踪系统与容器内的一个或多个应用程序交互，捕捉性能数据，然后把性能跟踪代理采集到的性能数据传输到数据分析子系统。

2.1 类加载流程设计

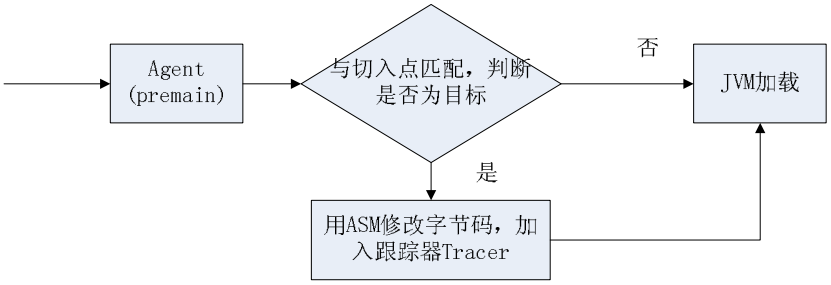


图 3 类加载流程设计

如图 3 所示，使用 Java Instrument 后，所有的字节码在加载过程中，都会先通过 Java Instrument 创建的一个包含指定 premain()函数的代理类对象。所以在这个代理类对象中，可以获取到每个类的字节码，并与切入点(PointCut)匹配，以判断是否为跟踪目标。如果获取到的类字节码信息符合跟踪目标的切入点，则通过 ASM 工具修改字节码，在切入点织入一个跟踪器(Tracer)，并指定目标类或方法在运行时将数据记录在 Tracer 对象中；如果类字节码信息不匹配目标的切入点，则不经过修改，直接交由 JVM 加载。

3 具体实现

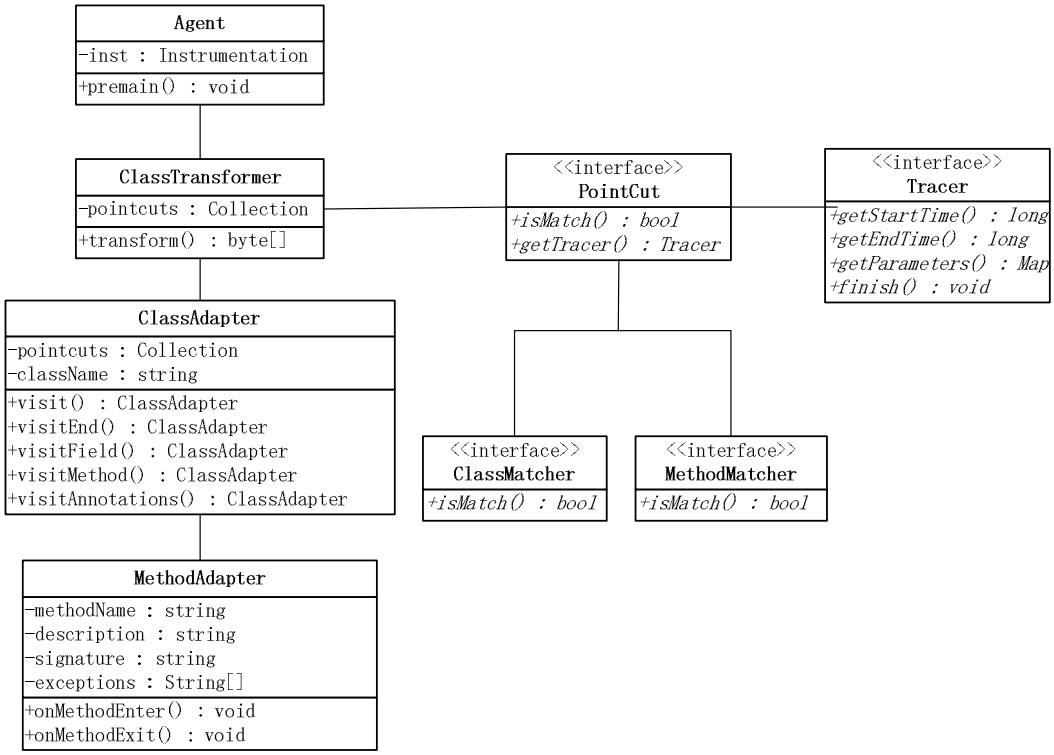


图 4 主要类图

- Agent，是整个程序的入口，虚拟机在启动时，会在加载应用之前调用 Agent.premain()，然后 Agent.premain()中实例化了一个定制的 ClassFileTransformer 即 ClassTransformer，并通过 inst.addTransformer()负责把代理类 ClassTransformer 载入虚拟机。

- ClassTransformer，代理类，应用类在加载时，都会调用 ClassTransformer 的 transform()方法，并以字节数组的形式传入类字节码；返回值类型为一个字节数组，作为类字节码交由 JVM 加载。所以，ClassTransformer 的 transform()方法是负责读取和修改类字节码的入口。

- PointCut，即 AOP 概念中的切入点，是程序中需要编织切面代码的点的集合，在本课题中，是需要进行性能跟踪的点的集合。它主要包含两个方法，isMatch()和 getTracer()。isMatch()用来判断是否需要进行性能跟踪，主要是通过类信息和方法信息(如类名、方法名、方法参数、返回值类型等)与 PointCut 对象中所包含的信息进行匹配，如果匹配成功，则进行性能跟踪。每个 PointCut 对象中都包含跟踪目标的信息，同时可以根据跟踪目标创建不同的 Tracer 对象，以记录对应的目标数据。所以 getTracer()用来返回与 PointCut 实例对应的 Tracer，以记录相关信息。除了直接实现 PointCut 接口创建切入点外，还可以直接通过 xml 配置文件生成 PointCut 对象，这样即提高了 PointCut 对象创建的灵活性，又简化了创建过程。

- ClassMatcher，类匹配器，可以有多种实现，例如子类匹配器，用来匹配某一类的子类；接口匹配器，用来匹配是否为某一接口的实现类；精确匹配器，根据类的全限名来匹配是否为某个类的字节码；组合匹配器，多种类匹配器的组合。

- MethodMatcher，方法匹配器，也可以有多种实现。

- Tracer，性能跟踪器，用于在性能跟踪过程中记录数据，其核心思想是在某方法开始时记录一次数据，在方法结束时再记录一次数据。getStartTime()方法，用来记录方法开始调用时的时间；getEndTime()方法，用来记录方法结束时的时间；将需要跟踪的类字段状态、方法参数状态记录在一个 Map 对象里，通过 getParameters()方法此 Map 对象。

- ClassAdapter，类适配器，用来读取和修改类字节码信息。一个类适配器可能有会有多个切入点，将所有的切入点放到一个集合变量 pointcuts 中；visit()方法用来访问类字节码的头部；visitEnd()用来访问类字节码的尾部；visitField()用来访问类的成员变量；visitMethod()用来访问方法实体；visitAnnotation()用来访问类的注解信息。

- MethodAdapter，方法适配器，用来读取和修改类字节码中方法的信息。一个方法体可以通过方法名 methodName、参数类型 description、返回值类型 signature、抛出异常类型 exceptions 等变量来唯一标识；在一个方法开始之前，会调用 onMethodEnter()，结束时会调用 onMethodExit()，基本上所有的性能跟踪数据操作都是在这两个方法中完成。

4 测试验证

4.1 验证可用性

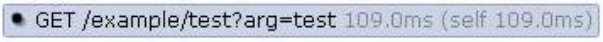
为了验证整个性能跟踪系统能够正常工作，我们可以通过一个配置文件定义一个 Servlet 的 PointCut，对 Servlet 的 service()方法进行性能跟踪。

配置文件示例：

```
<PointCut name=“ServletPointCut”>
    <class name=“javax.servlet.http. HttpServlet”>
    <method name=“service”
    parameter=“Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;”
    returnType=“”>
    <tracer=“monitor.tracer.ServletTracer”>
</ PointCut >
```

在此配置文件中，指定了 Servlet 切入点的类名、方法名、参数类型、返回值等，此外还指定了与 Servlet 相对应的 Tracer，即 monitor.tracer.ServletTracer，可以对 Servlet 的执行时间、URL 等参数进行性能跟踪。

测试结果：



Summary

URL	http://localhost:8080/example/test?arg=test
From Cache	false
Method	GET
Http Status	200
Mime-type	text/plain
Total Bytes	4 bytes

图 5 测试结果

如图 5 所示，整个性能跟踪系统抓取到了一个 Servlet 请求的详细状态：

- ✓ Servlet执行的时间是109ms；
- ✓ Servlet请求的路径是“/example/test”，携带的参数是“arg=test”
- ✓ 响应的结果不是来自于缓存；
- ✓ Servlet请求的方式是GET；
- ✓ 返回的状态码是200，表示此次WEB请求响应成功；
- ✓ 返回的文件类型是一个纯文本；
- ✓ 返回的文本内容长度是4个字节；

根据 Servlet 以上的状态数据，就可以对 Servlet 做合理的分析，以确定是否需要改进。

4.2 性能对比

为了验证整个性能跟踪系统不会因为额外的开销对服务器产生太大的影响，作者用 Tomcat+Mysql 搭建了一个小型的网站，并分别在使用性能跟踪系统和不使用性能跟踪系统的情况下，模拟 HTTP 请求消息，对网站的性能进行了测试，结果如表 1。

表 1 使用性能跟踪系统前后对比

	不使用性能跟踪系统	使用性能跟踪系统
平均响应时间（ms）	94	110
吞吐量（每秒请求数）	38.9	36
CPU 占用率	73%	81%
内存占用率	18.5%	9.2%

从表 1 中可以看出，使用性能跟踪系统后，平均响应延时仅增加了百分之十，吞吐量几乎没有变化，虽然 CPU 占用率高了 8 个百分点，但是内存占用率却低了 9 个百分点。之所以会 CPU 占用稍高，而内存占用率却降低的情况，是为了防止 ThreadLocal 可能会引发的内存泄漏现象，因此在每个 WEB 请求处理线程结束后，都会调用 ThreadLocal 的 remove()方法，触发一次 Java 虚拟机的垃圾回收[5]，不仅回收了 ThreadLocal 变量，也回收了其他已经无用的对象。因为垃圾回收的频率高了一些，所以 CPU 占用率较高，而内存占用率却降了下来。

从测试数据来看，性能跟踪系统对服务器的整体性能影响很小。

5 结束语

只有提供细粒度的性能跟踪数据，才能有针对性的对 Java 程序进行优化，提高程序效率。同时为了能让性能跟踪功能与原有的业务逻辑尽可能的解耦，引入了 AOP 技术，并阐述了实现 AOP 的方案。最后，对基于 AOP 的性能跟踪系统进行测试验证，实验结果说明此系统是完全可行的。

本系统与传统的 Java 运行时性能分析工具相比，具有以下优势：

1、 传统性能分析工具，会在运行时产生很多额外的性能开销。而本系统在运行时的额外开销很小，只是在类加载时会有少量时间的延迟。
2、 采用了 AOP 技术，使得性能跟踪功能从业务逻辑中独立出来，便于维护和升级。
3、 纯 Java 代码，具有平台无关性。

[参考文献] (References)

[1] Bill Venners. 深入 Java 虚拟机（原书第二版）[M].曹晓钢，蒋靖译.北京：机械工业出版社，2003.
[2] 朱晓民,温瑜,廖建新.SCE 研究进展及其在移动智能网中的实现[J].计算机工程，2005，31（6）：1 - 3.
[3] Craig Walls, Ryan Breidenbach.Spring in Action[M].李磊，程立，周悦虹.北京：人民邮电出版社，2006.

[4] ASM-Home Page.[EB/OL].http://asm.ow2.org/index.html
[5] Bruce Eckel.Java 编程思想（第四版）[M].陈昊鹏.北京：机械工业出版社，2007.