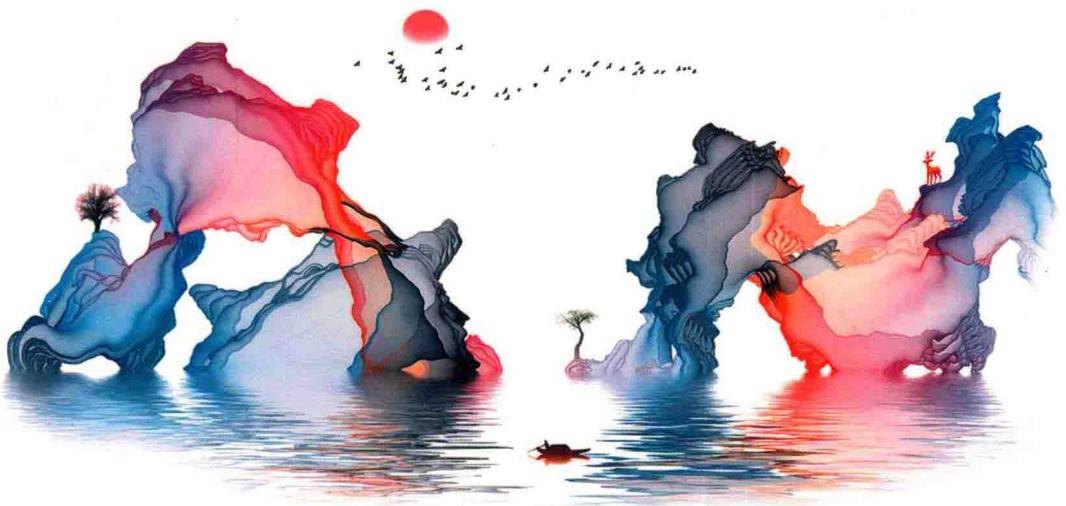
CMP BOOKS
机工IT

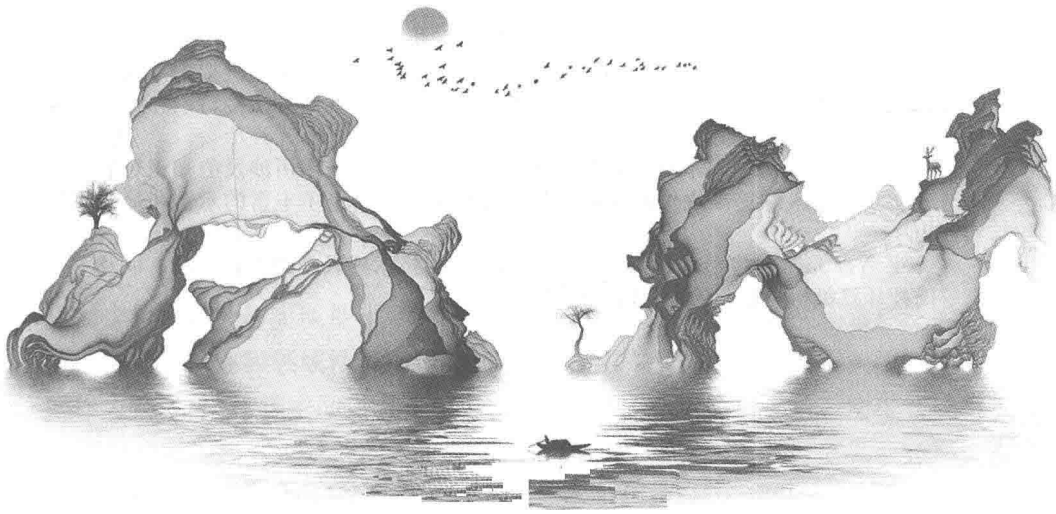
本书不仅关注对技术本身的介绍，还重点强调了这些技术所涉及的知识对读者进一步掌握工具和提高软件设计水平的重要作用，并给出了丰富的示例和最佳实践。



深入浅出 Java 虚拟机设计与实现

华保健 著

机械工业出版社
CHINA MACHINE PRESS



深入浅出 Java 虚拟机设计与实现

机械工业出版社
CHINA MACHINE PRESS

本书由国内编译器和虚拟机方面的资深研究者执笔，详细介绍了Java虚拟机设计与实现的各个方面，并给出了相关算法的实现。全书围绕虚拟机架构，讨论了虚拟机中的所有重要组件，包括类加载器、执行引擎、本地方法接口、异常处理、堆和垃圾收集、多线程及调试。

本书不仅关注对技术本身的介绍，还重点强调了这些技术所涉及的知识对读者进一步掌握工具和提高软件设计水平的重要作用，并给出了丰富的示例和最佳实践。

本书适合Java程序员、对编译器和虚拟机底层技术感兴趣的工程人员，以及高等院校计算机相关专业的学生阅读。

图书在版编目（CIP）数据

深入浅出：Java虚拟机设计与实现 / 华保健著. —北京：机械工业出版社，2020. 1

ISBN 978-7-111-64524-5

Ⅰ. ①深… Ⅱ. ①华… Ⅲ. ① Java语言－程序设计 Ⅳ. ① TP312. 8

中国版本图书馆CIP数据核字（2020）第002183号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）

策划编辑：孙 业 责任编辑：孙 业 赵小花

责任校对：张 力 责任印制：张 博

三河市国英印务有限公司印刷

2020年4月第1版 · 第1次印刷

169mm×239mm · 25印张 · 472千字

0001—2500册

标准书号：ISBN 978-7-111-64524-5

定价：99.00元

电话服务 网络服务

客服电话：010-88361066 机 工 官 网：www.cmpbook.com

010-88379833 机 工 官 博：weibo.com/cmp1952

010-68326294 金 书 网：www.golden-book.com

封底无防伪标均为盗版 机工教育服务网：www.cmpedu.com

前 言

虚拟机设计与实现是计算机科学中最古老、最成熟，也是应用最广泛的课题之一。许多通用性和领域性程序设计语言都使用某种与体系结构无关的中间语言格式作为编译目标，该中间语言在虚拟机上运行，因此虚拟机设计和实现就成为了支撑这类语言构建软件系统的关键与基础，而深入理解和掌握虚拟机设计和实现的基本原理和技术，也成为程序员必备的重要知识和技能。

但是，虚拟机的设计与实现所涉及的知识体系广而繁杂，和计算机科学的许多学科分支，如算法设计分析、程序设计语言、编译器、体系结构等，都有密切联系，并且，现代虚拟机已经发展得非常复杂，其中包含很多编程技巧和各种优化方法。虚拟机设计和实现的这些特点给初学者带来了很多困难：一方面，以小型的教学虚拟机入手研究，难以看到虚拟机设计与实现的全貌；另一方面，研究和学习工业级的虚拟机实现，又容易陷入繁复的实现细节。

本书讨论了一个典型 Java 虚拟机的设计原理和实现技术，其内容编排遵循了以下几个原则。

第一个原则是完整性。初学者在学习虚拟机设计与实现技术时，遇到的最大困难是在设计和实现一个虚拟机的过程中遇到问题的多样性，其中包括但不限于字节码文件格式、编译、动态加载和链接、执行引擎、堆和垃圾收集、本地方法接口、多线程与锁等广泛的内容。因此，若不能对这些相关技术进行全面介绍，读者就很难了解到虚拟机实现的全过程。基于此，本书完整地介绍了 Java6(JVMS2) 的实现过程，讨论了其中每个特性的实现原理和技术。

第二个原则是实践性。本书除了讨论虚拟机设计的基本原理和方法，还介绍了虚拟机的实现技术，对讨论的每个数据结构和算法都给出了类 C 语言的伪代码实现描述，这样，读者不仅能够深入理解虚拟机实现的基本原理，还能基于这些算法实现自己的虚拟机。

第三个原则是应用性，即本书特别强调了虚拟机设计和实现相关的理论和技

术对 Java 程序设计的指导作用。为此，书中结合对虚拟机实现技术的讨论，给出了较多的 Java 代码实例。一方面，通过对这些具体 Java 代码实例的讨论，读者可以更深入地理解虚拟机的运行原理；另一方面，理解虚拟机的设计与实现原理也有助于程序员构造更高质量的 Java 软件系统。笔者相信，虽然只有少量程序员会专门从事虚拟机的研究和开发工作，但理解包括虚拟机在内的底层系统的工作机理，是当前程序员知识栈中不可或缺的重要部分。

本书分为 8 章，第 1 章介绍 Java 虚拟机的整体架构。本章还讨论了一个简单的源语言——J 语言，其中包括对 J 语言语法、栈式计算机、J 字节码等方面的讲解，阐述了该源语言的程序从编译、加载到解释执行的整个过程，让读者对高级语言编译、字节码虚拟指令集、解释执行等虚拟机里的重要概念有一个全局的了解，也为后续章节中对 Java 虚拟机的深入讨论奠定基础。

第 2 章讨论了虚拟机类加载器的实现，主要内容有类的二进制定义、虚拟机方法区的设计，以及类加载的过程，包括类装载算法、类的验证、类的准备、类的解析、类的初始化和这些阶段的执行顺序。最后，本章还讨论了自定义类加载器的实现技术，并给出了自定义类加载器的两个典型应用：动态代理和热替换。

第 3 章讨论了执行引擎的设计与实现。主要内容包括：Java 运行栈的组织与数据结构设计、Java 方法调用规范与参数传递、Java 字节码执行引擎等。本章还简要讨论了本地方法执行引擎和可重入函数，以及一种常用的执行引擎实现加速技术——汇编模板。

第 4 章讨论了本地方法接口的实现技术。本章首先介绍了 Java 提供的标准本地方法接口 (Java Native Interface，JNI)，用于支持 Java 代码和本地代码的相互调用，然后讨论了二进制文件的加载、方法的静态注册和动态注册、本地方法的拦截，以及本地方法回调 Java 方法的技术。

第 5 章讨论了异常处理的实现方法和技术。本章首先给出了异常处理的两种最常用的实现技术——异常栈和异常表，讨论了这两种实现方式的优缺点，然后重点讨论了 Java 中使用的基于异常表的异常处理实现技术，包括异常表数据结构、栈回滚、本地方法异常等，最后讨论了异常处理中的一些其他重要问题，包括隐式异常、异常处理与多线程，以及异常的运行效率。

第 6 章讨论了堆和垃圾收集。Java 不支持动态内存的手工回收，而必须使用自动机制。本章讨论了 Java 堆数据结构、堆分配接口、对象的存储布局，并重点

讲解了基于 Cheney 算法的复制收集算法，另外，也介绍了和 Java 程序密切相关的根节点标记算法、终结和垃圾收集的触发机制。本章还讨论了对 Java 程序进行垃圾收集的一些关键问题，包括本地方法和垃圾收集、多线程与垃圾收集、无中断垃圾收集和类型标记等。

第 7 章讨论了多线程的实现技术。本章的主要内容有三个方面：第一，Java 多线程的语义模型，包括线程库中的主要线程方法、线程状态及线程中断；第二，管程的实现，包括管程数据结构、管程操作的接口与实现、管程与对象等；第三，多线程的实现，包括线程数据结构、创建线程对象、线程操作接口的支持等。本章还讨论了多线程与虚拟机其他子系统之间的交互。

第 8 章讨论了 Java 调试技术及其实现。本章内容包括 Java 调试器的整体架构、虚拟机端调试代理的设计与实现，以及 Java 调试在可调试性和安全性方面的问题。

限于篇幅并考虑读者的学习需求，本书略去了某些虚拟机实现技术，读者可以在其他著作中进一步学习。

本书是笔者在中国科学技术大学软件学院讲授的相关课程等资料基础上精心总结而成的，在此感谢中国科学技术大学相关老师和同学对课程的支持与建议。

由于作者水平和时间有限，错漏之处在所难免，敬请批评指正。

华保健

于中国科学技术大学软件学院

目 录

CONTENTS

第 1 章 虚拟机架构

任何一个复杂的软件系统都可以分解成若干模块来理解和实现，虚拟机也不例外。本章主要讨论 Java 虚拟机的整体架构和主要模块的划分，后续章节将分别讨论每个模块的具体实现技术。Java 虚拟机是一个典型的栈式计算机，本章首先给出在这类计算机架构上编译和解释执行程序的典型流程。为此，本章将结合一种简单的高级语言以及一个小的栈式计算机，来讨论如何把高级语言编译为栈式计算机的指令集，进而完成类加载、验证、解释执行等过程。尽管这种高级语言比 Java 简单得多，示例栈式计算机也比 Java 虚拟机简单得多，但这个过程可以帮读者建立很好的整体思路图，以便能快速理解栈式虚拟机的技术核心及各模块间的相互关系，为本书后续深入讨论 Java 语言及 Java 虚拟机奠定基础。

1.1 Java 与 Java 虚拟机

本节先对 Java 语言和 Java 虚拟机设计和实现的背景进行简要介绍，从而让读者对 Java 语言的特点和 Java 虚拟机的设计有一个全面的了解。

1.1.1 设计背景

20 世纪 90 年代初，SunMicrosystems 公司开始进行 Java 语言的设计与实现，当时的主要背景是：互联网正快速兴起，需要一种语言和平台能够对互联网上的编程及程序分发提供更好的支持。因此，在设计之初，Java 的设计者们就为这个新的语言和相应的执行平台确立了以下目标。

- 可移动：在一个平台上编写的程序代码，可以经由网络分发到其他的平台上运行 (当时主要以 Applets 的形式存在)。
- 跨平台：不同平台的体系结构和软硬件存在巨大差异，程序的执行需要做到与平台无关。
- 安全性：二进制代码必须能够独立于源代码进行检查和验证，以更好地保证二进制代码的安全性，这和传统上由 C/C++ 程序编译得到的二进制代码的执行

形成了鲜明的对比。

以上这些目标极大地影响了语言设计者所做的技术选型和设计思路，最终得到的新语言 Java 和新执行平台 Java 虚拟机都基本达到了这些最初的目标。

第一，为了提高可移动性，Java 字节码文件格式被设计为面向流，而且指令采用了栈式计算机字节码编码方式，不但使字节码文件在网络上的移动非常方便，而且由于栈式字节码隐式操作数的特点，代码的分发占用网络流量相对较小，因此代码分发的速度更快 (当然，今天的网络带宽已经和 20 世纪 90 年代初不可同日而语)。

第二，为了达到平台无关性，Java 程序不是由真实的物理机器执行，而是用 Java 虚拟机执行；Java 虚拟机屏蔽了平台的差异性：只要某个平台上实现了 Java 虚拟机，Java 程序就可以运行。尽管 Java 不是第一个采用虚拟指令和虚拟机执行平台的语言，而且有许多人，包括 C++ 之父 Bjarne Stroustup，都认为 Java 虚拟机不过是“又一个新的平台”，但 Java 的成功确实影响了后续许多语言的设计，采用虚拟机执行程序逐渐成为一个非常热门的方式，甚至有很多其他语言直接将 Java 虚拟机作为执行平台。

第三，在 Java 平台上，安全性被确立为非常重要的目标：Java 字节码二进制程序运行前，要首先经过字节码验证器的验证，验证未通过的程序会被拒绝执行，验证通过的程序还要在运行期间进行动态检查，对于含有不安全操作的程序，虚拟机会抛出运行时异常。通过引入这些机制，Java 实现了安全性的目标。这种设计理念是非常先进的，Java 也是较早全面采用类型安全的二进制代码的语言之一，又通过和垃圾收集等其他机制进行结合，避免了其他语言 (尤其是 C/C++) 程序中难以避免的数组越界、缓冲区溢出等安全问题。

1.1.2 Java 技术栈的组成要素

广义上讲，Java 技术发展至今，包括了四方面的核心内容：Java 语言、Java 类库、Java 字节码及字节码文件格式和 Java 虚拟机。

Java 语言指的是顶层语言自身，从 Java 1.0 开始，Java 语言就在不断地发展，陆续加入了泛型、函数式编程等越来越多的语言特征，演化成为目前复杂的集命令式、面向对象及函数式为一体的语言形态。

Java 类库通常指的是 Sun 公司 (已被 Oracle 收购) 伴随着 Java 语言和 Java 虚拟机发布的一套标准类库，广义上也包括所有的第三方类库。类库里提供了非

常丰富的数据结构、输入输出、操作系统接口、多线程支持等，大大提高了程序设计效率。同样，类库也在不断地演化，不断增加新的类和新的 API，也有一些类或 API 被废弃。

Java 字节码指的是 Sun 公司定义的一种低级别、类似汇编语言的程序设计语言，它是 Java 语言编译的目标语言。Java 字节码指令集是一种抽象的栈式计算机指令集，目前共包括 200 多条字节码指令。每条字节码指令的操作码部分都统一占用一个字节 (这也是“字节码”这个称谓的由来)，后面可跟多个操作数。相对 Java 语言及 Java 类库，Java 字节码的变动相对较小，20 多年来只对指令做了很小的修改。

Java 字节码文件格式指的是包含 Java 字节码程序的二进制可执行文件格式。严格来说，“文件格式”其实并非一定指的是磁盘文件，而是广义上任何符合文件格式标准的二进制流。随着 Java 新版本的发布，为了支持新的 Java 特性，Java 字节码文件格式也在不断变化。

最后，Java 虚拟机指的是能够读取和解析 Java 字节码文件、运行 Java 字节码程序的软件系统。除了 Sun 公司发布的“官方”Java 虚拟机 HotSpot 外，还有很多商业的和开源的 Java 虚拟机。本书要讲解的主要内容是 Java 虚拟机的设计与实现，但也和其他三部分内容有紧密联系。Java 虚拟机要依赖 Java 字节码及其文件格式进行理解，毕竟这是 Java 虚拟机的运行目标。Java 语言和 Java 类库与 Java 虚拟机的关系值得进一步讨论。

首先，Java 虚拟机是专门为 Java 语言设计的，Java 字节码中的部分指令就是为了支持 Java 语言的一些特性，例如，monitorenter 和 monitorexit 两条指令专门用来支持管程，而 invokeinterface 专门用来支持接口方法的调用，在实现这些指令时，必须熟悉 Java 语言中相关的机制。

其次，Java 虚拟机的实现也和 Java 类库密切相关，例如，类库 Object 中的大部分方法都是本地方法，如 getClass()、wait()、notify()、hashCode() 等。这些类库方法是本地方法的原因不难理解：它们都和虚拟机内部给对象分配的具体编码相关，因此，需要得到虚拟机的特殊支持。

由此也可以看到，深入学习 Java 虚拟机的运行机理与深入理解 Java 语言、Java 类库是相辅相成的。

1.1.3 Java 字节码

从编译的角度看，Java 字节码本质上定义了一种特定的“中间语言”，因为它

既不像 Java 语言这样处于顶层，也不是 X86、ARM 等汇编指令集那样的底层语言，抽象层次处于两者之间。一般来说，在编译的过程中，为某种高层语言引入恰当的中间语言，而不是将其直接编译为底层特定体系结构的指令集，有许多好处，其中最主要的好处是能够有效隔离高层语言和底层体系结构间的巨大差异，并屏蔽底层体系结构的细节，有助于实现平台无关性。

以 Java 字节码为例，除 Java 语言外，还有很多其他高级语言也可以编译成 Java 字节码，然后直接在 Java 虚拟机上执行。这种架构近年来非常流行，已经有 ML、Ruby、Python、Scala、Kotlin 等高级语言采用了这样的方案，Java 字节码也因此有了脱离 Java 技术范畴，发展成为更通用的中间语言的趋势。

实现 Java 虚拟机，离不开对字节码以及 Java 虚拟机的规范化。Java 虚拟机目前的官方规范是 Sun 公司发布的《Java 虚拟机规范》，它详细规定了所有 Java 虚拟机实现所必须遵循的规则，这些规则包括字节码文件格式、字节码文件合法性的校验规则、Java 类的初始化时机、每条字节码指令的执行语义等。这项规范是虚拟机的实现者必不可少的参考文件。

《Java 虚拟机规范》的新版本发布略落后于相应的 Java 语言规范发布及 Java 虚拟机的具体实现：Sun 公司在 20 世纪 90 年代初推出了最早期的 Java 虚拟机，接着将公司内部的虚拟机文档整理后，于 1996 年，发布了《Java 虚拟机规范》的第 1 版；在 1999 年，伴随 Java1.2 的发布，Sun 公司发布了《Java 虚拟机规范》(第 2 版)；2005 年，伴随着 Java6 的发布 (即 Java1.6)，Sun 公司发布了《Java 虚拟机规范》(第 3 版)；2013 年，Java7 的发布改变了 Java 虚拟机规范的命名规则，新的虚拟机规范被称为《Java 虚拟机规范》(Java SE7 版)；按新的命名规范，Oracle 又陆续于 2015 年和 2017 年发布了《Java 虚拟机规范》(Java SE8 版) 和《Java 虚拟机规范》(Java SE9 版)。

尽管每个版本的《Java 虚拟机规范》都难免受到 Sun (Oracle) 虚拟机具体实现的影响，甚至规范在许多地方都以 Sun(Oracle) 的具体虚拟机实现 HotSpot 为实例进行讲解，但本质上，《Java 虚拟机规范》仍然是一个较为松散的规定，在很多方面给虚拟机的实现留下了非常大的余地和空间。

以上的讨论，可以总结成两点：

(1) Java 字节码和 Java 虚拟机都和 Java 语言无关。尽管 Java 字节码和 Java 虚拟机最初都是为 Java 语言设计的，但目前已经有越来越多的其他高级语言可以

运行在 Java 虚拟机上，Java 虚拟机已经成为一个通用的运行平台。

(2)《Java 虚拟机规范》和 Java 虚拟机的具体实现无关。除了 Sun (Oracle) 的“官方”虚拟机外，不同的厂商、研究机构和个人，都可以按照规范的要求开发商用、研究或教学性质的 Java 虚拟机。

1.2 Java 虚拟机架构

《Java 虚拟机规范》给出了 Java 虚拟机的架构，整个架构比较复杂，但可以划分成图 1-1 所示的几个子系统：类加载子系统、堆存储子系统、执行引擎、本地方法接口、线程管理等。这些子系统通过适当的接口相互协作，共同实现 Java 虚拟机的功能。本节先简要介绍一下各个组成模块的功能以及相互间的接口。

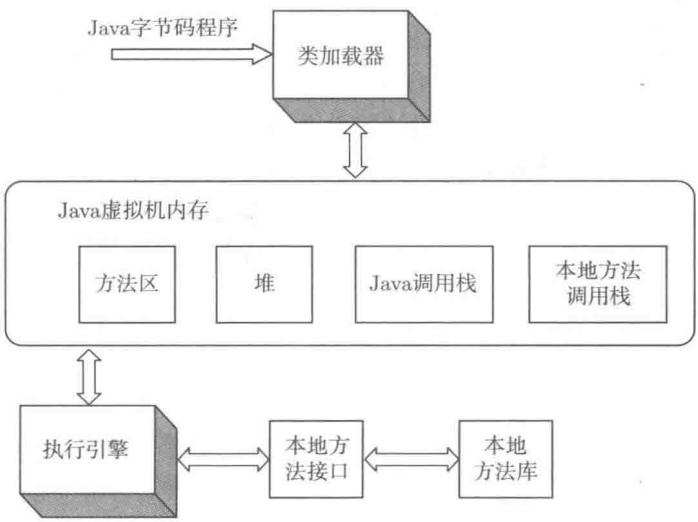


图 1-1 Java 虚拟机架构

类加载子系统负责把 Java 字节码文件加载到虚拟机内部，在虚拟机内部有专门的存储区来存放加载的类。在一些文献中，这些存放类的内部存储区被称为“方法区”(被称为“方法区”是因为历史原因，也许“类区”是更准确的名字)。加载完毕后，类加载子系统还要对类做进一步的处理，完成类的验证、准备、解析、初始化等操作，为执行类中的方法代码做好准备。Java 虚拟机类加载子系统采用了动态加载和动态链接的机制，即虚拟机在运行过程中会随时加载所需要的类，并把加载进来的类整合 (链接) 到虚拟机的内部数据结构中。这种机制增加了类加载子系