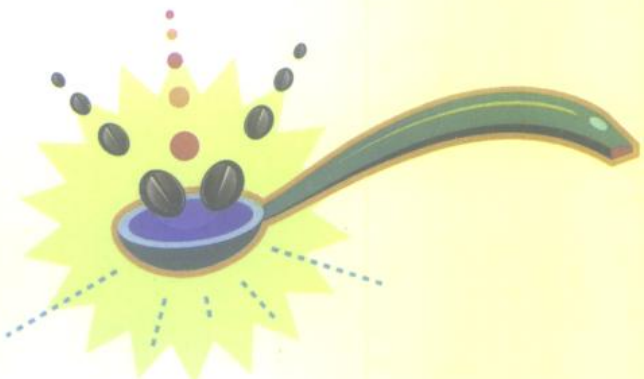
JAVA 系列丛书

JAVA虚拟机规范

JAVA 虚拟机规范

王克宏 主编

李京华 陈文光 编著



清华大学出版社

Java 系列丛书之二

Java虚拟机规范

王克宏　主编
李京华
陈文光　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书介绍了 Java 体系结构中的一个重要组成部分——Java 虚拟机。Java 虚拟机是 Java 的平台独立性和安全性的基础。本书先介绍了 Java 虚拟机的概貌，然后描述了 Java 虚拟机规范中的虚拟机体系结构，.class 文件(类文件)的格式，指令集。附录中还包括了 Sun 公司的 Java 虚拟机实现中使用的内部指令等内容。

本书不但为以 Java 虚拟机为目标机器的编译器的开发者和兼容的 Java 虚拟机的开发者提供了规范，而且为 Java 应用程序开发人员解决编程中的问题提供了新的途径。

图书在版编目(CIP)数据

Java 虚拟机规范/李京华，陈文光编著. —北京：清华大学出版社，1996.12
(Java 系列丛书/王克宏主编)
ISBN 7-302-02365-4

Ⅰ. J… Ⅱ. ①李… ②陈… Ⅲ. 虚拟处理机-规范 Ⅳ. TP338-65

中国版本图书馆 CIP 数据核字(96)第 22252 号

出 版 者：清华大学出版社(北京清华大学校内，邮政编码：100084)
责任编辑：徐培忠
印 刷 者：北京市丰台区丰华印刷厂印刷
发 行 者：新华书店总店北京科技发行所
开 本：787×1092 1/16 印张：6.75 字数：152.6 千字
版 次：1996 年 12 月第 1 版 1997 年 4 月第 2 次印刷
书 号：ISBN 7-302-02365-4/TP・1182
印 数：5001— 15000
定 价：10.50 元

序　言

近年来 Internet 以迅猛之势发展，已成为全世界规模最大的计算机网络，网上资源丰富，为世界各国千万用户所瞩目，我国也已于 1994 年 5 月正式入网，并已有成千上万使用者上网工作，如何充分、合理地利用 Internet 的网络与信息资源，为我国社会的各方面服务，是我们应予重视和研究的问题。

Java 的出现正迎合了 Internet 发展形势的需要，它所体现的简单、不依附于平台、面向对象、分布式、可靠性、安全性、可移植性、动态性、多线程等特性，为 Internet 的使用提供了一种良好的开发和运行环境，成为 Internet 适用、新型的编程语言。Java 出现后立即为世界各大公司所注目，纷纷购买 Java 使用权，并在剧烈的市场竞争中展开了大规模的研究与产品开发。Java 语言及其相关技术同样也引起了我国学术界、产业界和应用界的广泛关注和兴趣。因此 Java 丛书的出版发行，是符合时宜之举，必将受到大家的欢迎。

根据编著者的计划，Java 丛书将包括：《Java 语言入门》，《Java 虚拟机规范》，《Java 语言编程技术》，《Java 语言 Applet 编程技术》，《Java 语言 API 类库》，《Java 软件包的使用》，《Java 语言 SQL 接口》，《Java 语言调试技术》及《HotJava 使用指南》等 9 部书，约 100 余万字。详细深入地介绍 Java 语言及其相关技术并在短期内出齐，为我国广大读者研究和掌握 Java 提供了一种系统而全面的文献资料，无疑是一种十分有意义的事。Java 作为新生事物，尚在不断发展完善之中，因此我希望我们对 Java 及其相关技术不只停留于被动的学习、跟踪与使用，还应以积极主动的态度，通过应用实践和深入分析研究，参与开发创新，为计算机的网络应用做出我们自己的贡献。

清华大学计算机科学与技术系王克宏教授及其知识工程科研组，在 Internet 上进行多层次的研究开发工作已有一段时间，取得了阶段性成果，并曾在不久前召开的“中国计算机学会第九次全国学术大会”的全体会议上作报告，受到与会者的欢迎。相信他们在结合自己研究工作基础上编写的这套 Java 系列丛书，必将有助于我国信息科学技术的发展。我高兴地为丛书作序并祝他们成功！

张效祥

1996.8

开辟 Java 计算的新时代

Sun 公司发明的 Java 语言风靡全球。据 Forrester Rcsearch 公司对《幸福》杂志评出的世界 1000 家大型企业进行的调查，目前在它们中间已有 62%正在使用 Java 从事开发工作；有 42%已将 Java 纳入自己企业一年内的战略开发计划。这个调查结果表明，Java 已为大家所接收。

计算机自从 50 年代诞生以来，经历了几个发展时代：终端/主机计算、PC 计算、客户机/服务器计算，而现在已发展到一个新的阶段，即 Java 计算时代。这个时代的特点就是以 Java 为代表的网络计算。

Java 带来的是一场革命。这是第一个真正独立于平台的计算方案，它能充分发挥 Internet 的作用。Java 计算是实现“一个世界，一个网络”构想的关键。这样一个透明的、全球连接的和信息交换的网络，可将所有最新的计算技术、电话、出版/媒体和娱乐集成一体。过去妨碍这一构想成为现实的是计算平台不能兼容，而 Java 语言却以其许多优秀特性使之成为允许各类系统相互兼容和共享应用环境的桥梁。这使各类软件“一次写成，到处可用（write once/run anywhere）”，这样，相同的软件可在不同器件上运行，无论是 PC 机、苹果机、UNIX 计算机、还是顶置盒、PDA（个人数据助理）、移动电话、乃致智能元器件、无一例外。Java 的这一贡献将使全人类受益，因而给人们的生活方式带来极大的改变。

清华大学王克宏教授是最早研究 Java 语言的中国学者之一。在他的周围，许多热心于 Java 计算的青年学者、博士生、硕士生和本科生组成了一支朝气蓬勃、积极进取的生力军，他们思想活跃、工作勤奋、通过努力把 Java 语言的开发和应用提高到一个新的水平。据我所知，在中国的诸多大专院校和科研院所，包括北京航空航天大学、上海复旦大学，也都有许多学者致力于 Java 的研究和开发。由于他们的辛勤努力和积极探求，在 Java 平台上开发和使用的 Java 应用将给中国的信息业带来新的发展，使 Java 计算在中国进一步得到推广。

《Java 系列丛书》就是 Java 开发园中的一朵奇葩，它不俏不争，以其朴实无华、实用有效的风格展示于人。我们希望由王克宏教授主编的该套丛书能赢得广大读者的喜爱，希望能有更多的中国智者和用户在 Java 计算的时代熟悉 Java，掌握 Java，利用 Java 的诸多优点，编写出更多更美的应用软件，从而造福于人类。

谨以此愿祝贺 Java 系列丛书的出版发行。

李永起

1996.10

* 本文作者为 Sun 中国公司市场总监

前　言

本书描述了 Java 虚拟机 1.0 版及其指令集。本书的内容不但为以 Java 虚拟机为目标机器的编译器作者和希望实现兼容的 Java 虚拟机的作者提供了规范，而且为 Java 应用程序开发人员解决他们编程中的问题提供了新的途径。

本书内容的组织结构如下：

第 1 章是 Java 虚拟机综述。

第 2 章描述了 Java 虚拟机的体系结构。

第 3 章描述了 .class 文件(类文件)的结构。

第 4 章描述了字节码。

第 5 章是分析类文件结构和指令的一个例子。

附录 A 包含了 Sun 公司的 Java 虚拟机的实现中使用的一些内部指令。严格地说，这些内部指令不是 Java 虚拟机规范的一部分，在此可以作为我们的 Java 虚拟机实现的参考资料。随着更多的 Java 虚拟机实现的出现，附录 A 中的内容今后可能将不再需要。

附录 B 是指令集索引。将 Java 虚拟机的指令按其对应操作码的数字的大小顺序排列，以方便读者查找。

附录 C 是词汇表。列出了 Java 虚拟机领域中常见的英文专业词汇并给出了中文释义，相信会对读者阅读英文文献资料带来方便。当然这方面词汇的翻译国内尚无定例，我们的成果也谨供广大读者参考。

Sun 公司将对符合本规范的兼容的 Java 虚拟机发放 Java Virtue Machine 商标和图标的许可证。如果你想要实现自己的 Java 虚拟机，可以用 email 地址 java@java.sun.com 与 Sun 公司联系，使你能得到 Sun 公司的合作，以保证你的实现是 100%兼容的。

目　录

第 1 章　Java 虚拟机综述

1.1　什么是 Java 虚拟机

什么是 Java 虚拟机(Java Virtual Machine)呢? 也许这是你最为关心的问题。我们认为 Java 虚拟机可以定义为:运行经过编译的 Java 目标代码的计算机的实现。它能运行的 Java 程序,既包括独立的 Java 应用程序(application),也包括下载到诸如 Netscape Navigator, HotJava 等 Web 浏览器中的 applet。

一般认为,Java 虚拟机是建立在实际的处理器基础上的假想的计算机。在 Java 语言刚刚推出的一段时间里,Java 虚拟机都是通过软件仿真的方法实现的。经过编译的 Java 目标代码,我们称为 Java 字节码(byte-code),再经过本地计算机上的 Java 解释器的解释,或者经过编译器的编译,就可以运行了。

Java 虚拟机不但可以用软件实现,也可以用硬件实现。最近就有了硬件实现的 Java 虚拟机。1992 年 2 月,Sun Microelectronics 公司公布了他们的最新产品——专门为 Java 而优化设计的微处理器系列产品。包括性能价格比最高的、用于蜂窝电话、打印机等小型嵌入式设备的 picoJava;建立在 picoJava 的基础上、增加了 I/O、内存、通信、控制等功能的通用型的 microJava;高档的、工业界最快速的 Java 处理器 UltraJava 等。

Java 虚拟机可以是假想的计算机,也可以是像 Sun Microelectronics 公司实现的处理器那样的实际的计算机。当然,目前大多数的 Java 虚拟机还是用软件方法实现的。所以,为简便和通用起见,我们在综述中介绍的内容也大多基于这种软件实现。但是,本书中介绍的 Java 虚拟机规范,是无论哪种实现都要遵从的。

1.2　由 Java 程序生成(实际机器的)可执行代码的过程

许多程序设计语言通过对源程序进行编译和链接,直接生成可执行的代码。而 Java 语言和它们不同,它的这个过程包括 Java 程序的编译,字节码的装入、校验、解释或编译。了解 Java 语言的这个过程,对掌握 Java 虚拟机会有所帮助。

Java 程序首先经过编译,生成字节码。这个过程由程序开发者的 Java 编译程序完成。字节码的层次大致相当于汇编语言一级。当然,它并不是针对某种特定的计算机硬件平台的。对变量和方法(相当于函数)的引用,并不在编译过程中确定为数值引用,即通过具体的偏移量的值引用,而是将符号引用的信息保存在字节码中。

字节码的装入、校验、解释或编译,由本地计算机的解释器或编译器完成。首先,类装入器装入程序所需的所有代码,包括程序中调用(use)、包含(contain)、继承(inherit)的所有类

的代码。每个类都被装入一个独立的名字空间内，彼此之间只有通过符号引用才能互相作用。本地的类和外部的类在地址空间上是区分开的。所有的类都装入以后，可执行代码的内存布局就被确定。由符号引用到内存地址空间的查询表也建立起来了。然后，字节码校验器对装入的字节码进行校验，以排除错误和不安全的因素。最后，目前通常由解释器解释执行字节码。但是，解释执行的速度较慢，一般比 C 语言慢 15 倍左右。而如果编译执行，速度将基本能和 C 语言匹敌。截止到我们写稿时为止，Sun 公司还未推出 Java 字节码的编译器。但是可以预计，今后各方开发商将会逐渐推出 Java 编译器，以满足速度要求较高的程序的要求。

1.3 Java 虚拟机规范

为什么要为 Java 虚拟机制定规范呢？一个最明显的理由就是要保证 Java 代码在任何系统上都能够运行。凡是符合 Java 虚拟机规范的实现，都是百分之百兼容的。Java 虚拟机对其实现作出了具体的规定。

Java 虚拟机的体系结构直接支持 Java 语言的这些基本数据类型，包括 byte、short、int、long、float、double 和 char。它们的定义是独立于具体的平台的，这为 Java 程序的可移植性打下了基础。规范没有对 object 的内部结构作特殊的要求。由于在各种计算机体系结构中，寄存器的设置千差万别，所以 Java 虚拟机采用了面向堆栈的体系结构，只设置了数量很少的寄存器。这几个寄存器是程序计数器，栈顶指针，运行环境指针和局部变量指针。这种设置减少了在只有少量寄存器或非通用寄存器的计算机上实现的困难。而对于拥有较多寄存器的计算机，可以通过具体实现时的优化以充分利用计算机的资源。Java 虚拟机的局部变量和操作数栈都是 32 位的，long 和 double 类型需要占据两个 32 位空间。除了以上方面的内容，Java 虚拟机还对运行环境、无用单元收集堆、方法区、指令集和限制等方面作了规定。

Java 虚拟机的程序代码存储在类文件(.class)中。每个类文件包含了一个 Java 类或者 Java 接口的编译后的代码。Java 虚拟机对类文件的格式作出了规定。

Java 虚拟机的指令由操作码和操作数组成。操作码为 8 位，理论上最多能够设计 256 条指令。目前，只有其中的一部分(约 200 条)被使用。操作数的数目依指令不同而不同。除了表跳转指令 4 字节对齐以外，其他的指令都是字节对齐的。

1.4 采用 Java 虚拟机的意义

采用 Java 虚拟机，对 Java 的平台独立性和安全性有很大的意义。

平台独立性

在 Java 平台的结构中(图 1.1)，Java 虚拟机处在核心的位置。它的下方是移植接口，移植接口由依赖平台的和不依赖平台的两部分组成，其中依赖于平台的部分称为适配器。Java 虚拟机通过移植接口在具体的操作系统上实现。如果在 JavaOS 上实现，则不需要依赖于平台的适配器，因为这部分工作已由 JavaOS 完成。因此，对于 Java 虚拟机来说，操作系统和更

低层的硬件是透明的。(关于透明性的概念,我国计算机界有两种不同的定义。我们采用的是较为普遍的一种,即指的是对于 Java 虚拟机这一层次来说,操作系统和硬件好像不存在一样,因此也无需考虑。)在 Java 虚拟机的上方,是 Java 的类和 Java API(Java 应用程序接口)。在 Java API 上,我们可以编写 Java 应用程序和 applet。所以,对于 Java 应用程序和 applet 这一层次来说,操作系统和硬件就更是透明的了。我们编写的 Java 程序,可以在任何 Java 平台上运行而无需修改。

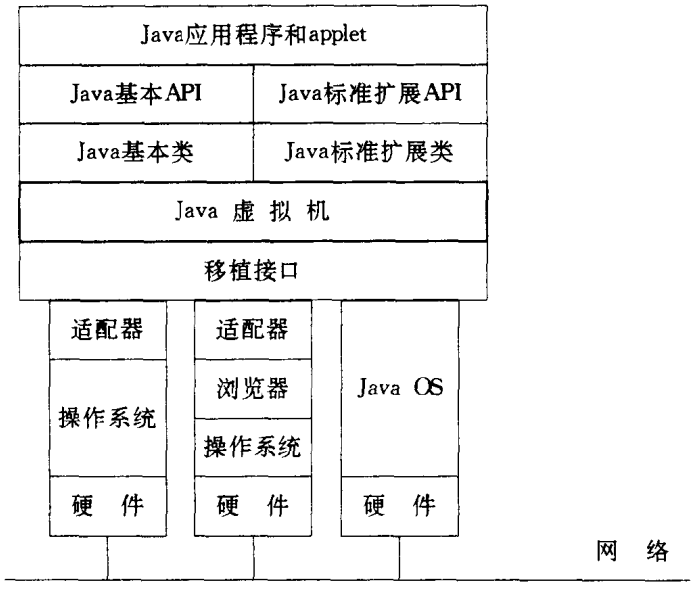


图 1.1

Java 虚拟机定义了独立于平台的类文件格式和字节码形式的指令集。在任何 Java 程序的字节码表示中,变量和方法的引用都是使用符号,而不是使用具体的数字。用数字引用替代符号引用是在运行时由解释器完成的。当第一次引用一个变量或方法的时候,用它的名字在查找表中查找,以确定它的数字偏移量。这种查找和替换只在第一次引用时需要,因此对速度影响不大。由于存储器的布局要在运行时才确定,所以对类的变量和方法的改变不会影响现存的字节码。例如,你的 Java 程序引用了其他系统中的某个类,该系统中那个类的更新,不会使你的程序崩溃。(如果是C++,你必须将你的程序中相关部分再次编译。)这也提高了 Java 的平台独立性。

安全性

作为适用于网络编程的语言,安全性是必须要保证的。Java 虚拟机在 Java 的安全体系中占有重要地位。至少有以下的 3 项安全性保证是和 Java 虚拟机有关的。

第 1 项是内存的布局。我们已经介绍过,在任何 Java 程序的字节码表示中,变量和方法的引用都是使用符号,而不是使用具体的数字。用数字引用替代符号引用在运行时由解释器完成。对于编程人员来说,内存分配是透明的,不可能在编写程序的时候就知道内存的布局。编程人员无法通过伪造指针访问任何的内存空间。

第 2 项是字节码的检验。虽然 Java 编译器保证源代码不违反安全规范,但是当从其他

地方下载 Java 字节码时，并不能认为它是安全的。因为有可能是生成这些字节码的编译器本身有安全性的问题，也有可能是有人故意制作了一个 Java 编译器，以侵入其他用户的系统。所以必须对字节码进行检验。字节码的一个重要特性是它的可解析性很强，能够对指令集进行分析，并对其以后的行为做出一定的推断，这就为检验打下了基础。通过检验的程序，可以确保不存在伪造的指针、不违反访问权限、不非法访问对象、不会导致操作数栈溢出，等等。

第 3 项是字节码装载器的安全检查。因为字节码采用符号引用，内存布局在运行时确定，所以可以在字节码装入时将本地的类和外来的类严格区别开。本地的类共有一个统一的名字空间，外来的类每个网络节点一个名字空间。外来的类不能以任何方式对本地的类进行操作。这种机制为本地的类建立起一道安全屏障。

1.5 为什么要学习 Java 虚拟机

也许很多读者会这样说：不错，Java 虚拟机的确在 Java 平台的体系结构中作用很大。但是我不会自己去做实现一个 Java 虚拟机或是写一个 Java 解释器这种依赖于硬件平台的工作，我的主要工作是编写 Java 应用程序。既然从这一层次上看，Java 虚拟机是透明的，那么我为什么要学习 Java 虚拟机呢？好，我们现在就来回答这个问题。

学习 Java 虚拟机的最重要的原因是，它能给你提供解决 Java 编程中的问题的其他方法。Java 的体系结构是非常开放的，只要你学习过 Java 虚拟机的基本规范，你就可以轻易地对 Java 添加扩展内容。由于 Java 虚拟机的可移植性，你只需编写一遍扩展内容，以后就都可以使用了。因此，如果你不喜欢 Java 语言的某方面的特性，为什么不自己制作一个扩展库，实现你想要的特性呢？要知道，这也许并没有你所想象的那么难！

例如，你的应用程序中有许多对矩阵的操作。你可能希望用简单的操作符，诸如＋、－、＊、/ 来对矩阵进行运算。而 Java 不允许对操作符的重载。怎么办呢？你可以编写一个简单的语法分析程序，将矩阵的表达式直接编译为 Java 字节码。然后，你就可以在任意的 Java 解释器下，在任意的 Java 程序中调用这个方法，实现所需的功能。又例如，你要写一个基于规则的应用程序，希望用简单明了的方式表达规则。编写一个 Java 虚拟机的接口就可以使你快速，高效地做到这一点。

使用 Java 虚拟机能够释放出 Java 的巨大能力。它使你可以开发出附加的语法，表达你想要解决的问题，还可以使你能够完全控制你的应用程序的性能。你可以超越 Java 语言的设计特性，甚至编写你自己的语言。已经有人开发出一种为 Java 虚拟机产生字节码的 ada 编译器。（ada 是一种计算机高级语言，在美国军方应用较广。）由于 Java 虚拟机的体系结构的特性，你可以简单、高效地做到这些，你的工作成果可以被几乎所有的计算机、所有的 Java 解释器、所有的 Java 应用程序访问。

你想在 Java 的竞争中领先吗？从 Java 虚拟机开始吧！

第 2 章　Java 虚拟机的体系结构

2.1　支持的数据类型

Java 虚拟机支持 Java 语言的基本数据类型：

```
byte：    // 1 字节有符号补码整数
short：   // 2 字节有符号补码整数
int：     // 4 字节有符号补码整数
long：    // 8 字节有符号补码整数
float：   // 4 字节 IEEE 754 单精度浮点数
double：  // 8 字节 IEEE 754 双精度浮点数
char：    // 2 字节无符号 Unicode 字符
```

几乎所有的 Java 类型检查都是在编译时完成的。上面列出的原始数据类型的数据在 Java 执行时不需要用硬件标记。操作这些原始数据类型的数据的字节码(指令)本身就已经指出了操作数的数据类型。例如 iadd,ladd,fadd 和 dadd 指令都是把两个数相加,其操作数类型分别是 int,long,float 和 double。

虚拟机没有给 boolean(布尔)类型设置单独的指令。boolean 型的数据是由整数指令,包括整数返回来处理的。boolean 型的数组则是用 byte 数组来处理的。

虚拟机规范使用 IEEE754 格式的浮点数,并支持逐步下溢。不支持 IEEE 格式的计算机在运行 Java 数值计算程序时可能会非常慢。

虚拟机支持的其他数据类型包括：

```
object   // 对一个 Java object(对象)的 4 字节引用
returnAddress   // 4 字节,用于 jsr/ret/jsr_w/ret_w 指令
```

注:Java 数组被当作 object 处理。

虚拟机的规范对于 object 内部的结构没有任何特殊的要求。在我们的实现中,对 object 的引用是一个句柄,其中包含一对指针:一个指针指向该 object 的方法表,另一个指向给该 object 的数据。其他的实现方法包括使用 inline caching 来代替方法表分派。后一种方法在即将出现的计算机硬件上可能会运行得更快。

用 Java 虚拟机的字节码表示的程序应该遵守类型规定。Java 虚拟机的实现应拒绝执行违反了类型规定的字节码程序。

Java 虚拟机由于字节码定义的限制似乎只能运行于 32 位地址空间的机器上。但是可以创建一个 Java 虚拟机,它自动地把字节码转换成 64 位的形式。对于这种变换的描述超出了本规范的范围。

2.2 寄存器

虚拟机在任何时候都是在执行一个单独方法的代码，pc 寄存器包含下一个要执行的字节码的地址。

每个方法都有一个为它分配的存储器空间，用于存放：

- 一个局部变量集，由 vars 寄存器引用，
- 一个操作数栈，由 optop 寄存器引用，以及
- 一个“运行环境”结构，由 frame 寄存器引用。

局部变量和操作数栈的大小在编译时就可以得到，“运行环境”结构的大小由解释器提供，因此为方法分配存储器空间是很方便的。

所有寄存器都是 32 位的。

2.3 局部变量

每个 Java 方法使用一个固定大小的局部变量集。它们按照与 vars 寄存器的字偏移量来寻址。局部变量都是 32 位的。

长整数和双精度浮点数占据了两个局部变量的空间，但是按照第一个局部变量的索引来寻址。(例如，一个具有索引 n 的局部变量如果是一个双精度浮点数，那么它实际占据了索引 n 和 n+1 所指向的存储空间。)虚拟机规范并不要求在局部变量中的 64 位的值是 64 位对齐的。因此，实现者可以自由地决定把长整数和双精度数分成两个字的合适的方法。

虚拟机提供了把局部变量中的值装载到操作数栈的指令，也提供了把操作数栈中的值存储到局部变量的指令。

2.4 操作数栈

机器指令只从操作数栈中取操作数，对它们进行操作，并把结果返回到栈中。选择栈结构的原因是：在只有少量寄存器或非通用寄存器的机器(如 Intel 486)上也能够高效地模拟虚拟机的行为。

操作数栈是 32 位的。它用于给方法传递参数并从方法接收结果，也用于支持操作的参数并保存操作的结果。

例如，iadd 指令将两个整数相加。相加的两个整数应该是操作数栈顶的两个字。这两个字是由先前的指令压进栈的。这两个整数将被从操作数栈中弹出、相加，并把结果压回到操作数栈中。

每个原始数据类型都有专门的指令对它们进行必需的操作。除了 long 和 double 型外，每个操作数在栈中需要一个存储位置，long 和 double 型则需要两个位置。

操作数只能适用于其类型的操作符所操作。例如，压入两个 int 类型的数，然后把它们当作是一个 long 类型的数是非法的。在 Sun 的虚拟机实现中，这个限制由字节码检验器强

制实行。但是，有少数操作(如 dupe 和 swap 操作)，用于对运行时数据区进行操作时是不考虑类型的。

在本书描述的虚拟机指令中，指令的执行对于操作数栈的影响是用文字表示的，栈的方向是从左到右，每一项表示一个 32 位的字。因此：

Stack:…,value1,value2 ⇒ …,value3

表示了一个操作开始时，栈顶为 value2，次栈顶为 value1。作为指令执行的结果，value1 和 value2 被弹出栈并被 value3 所代替。(value3 是指令计算的结果。) 用…表示的栈的其他部分，这些部分不受该指令执行的影响。

long 和 double 类型的数据在操作数栈中占有两个 32 位的字的位置：

stack:… ⇒ …,value-word1,value－word2

本规范没有说明如何从 64 位的 long 或 double 类型的值中选择这两个字；只要在虚拟机实现的内部保持一致即可。

2.5 运行环境

在运行环境中包含的信息用于动态链接、正常的方法返回以及异常传播。

动态链接

运行环境包括指向当前类和当前方法的解释器符号表的指针，用于支持方法代码的动态链接。方法的类文件代码在引用要调用的方法和要访问的变量时使用符号。动态链接把符号形式的方法调用翻译成实际方法调用，装载必要的类以解释还没有定义的符号，并把变量访问翻译成与这些变量运行时存储结构相适应的偏移地址。

动态链接方法和变量减小了修改其它类时对本程序代码的影响。

正常的方法返回

如果当前方法正常地结束了，在执行了一条具有正确类型的返回指令后，调用者会得到一个返回值。

执行环境在正常返回的情况下被用于恢复调用者的寄存器，并把调用者的程序计数器增加一个恰当的数值以跳过已执行过的方法调用指令。然后在调用者的执行环境中继续执行下去。

异常和错误传播

异常的情况，在 Java 中被称作 Error(错误)或 Exception(异常)，是 Throwable 类的子类。在程序中出现异常的原因是：

- 动态链接错，如无法找到所需的类文件。
- 运行时错，如对一个空指针的引用。
- 其他线程发生的异步事件，如由 Thread.Stop 引发的异步事件。

·程序使用了 throw 语句。

当异常发生时：

·检查与当前方法相联系的 catch 子句表。每个 catch 子句包含其有效指令范围，能够处理的异常类型，以及处理异常的代码地址。

·与异常相匹配的 catch 子句应该符合下面的条件：造成异常的指令在其指令范围之内，发生的异常类型是其能处理的异常类型的子类型。如果找到了匹配的 catch 子句的话，系统转移到指定的异常处理块处执行。如果没有找到异常处理块，重复寻找匹配的 catch 子句的过程，直到当前方法的所有嵌套的 catch 子句都已被检查过。

·由于虚拟机从第 1 个匹配的 catch 子句处继续执行，所以 catch 子句表中的顺序是很重要的。因为 Java 代码是结构化的，因此总可以把某个方法的所有的异常处理块都按顺序排列到一个表中，对任意可能的程序计数器的值，都可以用线性的顺序找到合适的异常处理块，以处理在该程序计数器值下发生的异常情况。

·如果找不到匹配的 catch 子句，那么当前方法得到一个“未截获异常”的结果，返回调用者。调用当前方法的调用者的运行状态从运行环境中恢复，并在其 catch 子句表中查找匹配的 catch 子句，就好像异常刚刚在调用者中发生一样。

附加的信息

运行环境可以扩展，以包含一些与虚拟机的实现相关的信息，如调试信息。

2.6 无用单元收集堆

Java 的堆是一个运行时的数据区，类的实例(对象)从中分配空间。Java 语言具有无用单元收集能力——它不给程序员显式释放对象的能力。Java 不规定使用某个特定的无用单元收集算法，可以根据系统的需求使用各种各样的算法。

2.7 方法区

方法区与传统语言中的编译后代码或是 UNIX 进程中的正文段类似。它保存方法代码(编译后的 Java 代码)和符号表。在当前的 Java 实现中，方法代码不包括在无用单元收集堆中。但计划在将来的版本实现。

2.8 Java 指令集

Java 指令集中的指令包含 1 个单字节的操作符，用于指定要执行的操作，还有 0 个或多个操作数，提供操作所需的参数或数据。许多指令没有操作数，仅由 1 个单字节的操作符构成。

虚拟机的内层循环的执行过程如下：

```
do {
```

取 1 个操作符字节
根据操作符的值执行 1 个动作
} while（程序未结束）

附加的操作数的数量和大小是由操作符决定的。如果附加的操作数比 1 个字节要大，那么它存储的顺序是高位字节优先。例如，1 个 16 位的参数以两个字节的形式存放，其值为：

第 1 个字节 * 256＋第 2 个字节

字节码指令流一般只是字节对齐的。指令 tableswitch 和 lookupswitch 是例外，在这两条指令内部要求强制的 4 字节边界对齐。

以上特点使得编译后的 Java 程序的虚拟机代码非常紧凑。这反映了以某种程度上可能的性能损失来换取紧凑性的有意的偏好。

2.9 限　制

一个类的常数池的最大表项数是 65535。这是对单个类的复杂性的内部限制。

每个方法的代码总长度不能超过 65535 字节。这个限制是由于异常表中，行号表中以及局部变量表中对代码的索引限制的。这个限制将在 1.0beta2 版中得到修正。

在一个方法调用中的参数所包含的字的个数不能超过 255 个。