JAVA 语言程序设计(连载五)

第五讲 Java 虚拟机

吴明行　杨乔林　(中科院计算所)

一、Java 虚拟机简介

1.概念、功能及特点

Java 虚拟机是一台面向对象、由软件模拟实现的计算机。被编译后的 Java 类文件被假想成在这台虚拟机上解释执行。Java 的类文件以"字节码"(Bytecode)形式组成。字节码的格式不依赖于特定的硬件体系结构、操作系统和窗口系统，任何一台机器只要配备了 Java 解释器，即可运行 Java 程序，因此具有平台独立性。

2.作用

Java 系统是由 Java 语言和 Java 虚拟机组成。作用如下图所示：

3.设计目的

设计规范：试图在技术范围内为读者提供一个抽象、逻辑的机器设计，创建一种可以在 Internet 上进行相互交换信息的语言工具。Java 设计者不希望让 Java 成为一种特定的语言，而是希望源代码可以作为工具被自由的使用，当确认 Java 的解释程序在不同的 Java 工具之间被可靠的共享后，方可得到网上运行者的权限。以上设计避免了 Java 虚拟机的定义是完全抽象的。下面我们将介绍机器的抽象设计和编译了的 Java 代码的具体转换格式。就其性质而言，它们还划分成两部分。

Java 的转换部分(即转换为 Java 虚拟机所需的代码规范)包括如下成分：

·指令集语法，包括操作码和操作数字节类型，校准和 endian 码

·指令集操作码的值

·指令或支持结构中的标号值(例如：类型标号等)

·编译过的 Java 码中支持结构布局(例如：常数池等)

·Java 对象格式

与转换无关的部分包括如下成分：

·运行数据域的布局和管理

·无用单元的回收算法、策略和限制

·编译器、开发环境和运行时间(除去生成和读取无效的类文件代码部分)

·对被接收的类文件代码进行优化

二、Java 虚拟机的组成

Java 虚拟机(简称 JVM)由以下几部分组成：

·一个指令集

·一组寄存器

·一个栈

·一个无用单元收集堆

·一个方法域

以上都是虚拟机的逻辑组成元素。任何基于这个虚拟机的 Java 系统，它们都以某种方式来使用。JVM 可通过利用各种传统技术实现，例如：bytecode 解释，native 或 silicon 码编译。

JVM 存储区不是存储区中事先特定的或相互关联的区域，JVM 存储区也不必由连续的存储区域构成。指令集、寄存器和存储区都是以指定的最小逻辑宽度值表示(例如：Java 栈是 32bits 宽)。对于这些问题下面将作进一步讨论。

1. Java 指令集

Java 指令集是与 Java 应用程序等价的集合语言。Java 程序被编译成 Java 的指令集合，如同 C 语言程序被编译成微处理器的指令集合。一条 Java 指令是由一个操作码(说明当前执行的运算)和 0 个或多个操作数(提供运算中要使用的参数或数据)组成，许多指令仅含一个操作码而不包含操作数。指令集操作码一般是单字

节长，而操作数是多字节。当操作数大于一个字节时他们以 big－endian 编码(高位字节在前)存储。例如：

一个 16 位参数以两字节形式存放 first-byte * 256 + second-byte。

大于 8 位的操作数一般由运行时的字节流构成，但不受保护。一个指令流是以字节为单元对齐而不是以字节块对齐，这个规则的一个例外是表转换和查找转换指令。这些规定是为了使编译程序的虚拟机器码紧凑，且有意的牺牲压缩方面某些性能。

2．**基本数据类型**

JVM 指令集把 JVM 运行数据域上的数据解释为一些基本类型。基本数据类型包括整型、长整型、单双精度浮点，字节和短型，所有数据类型是带符号的。无符号的短型只在字符(unicode) 中使用，计算中的 Java 对象被表示成对象类型。少数运算（例如：dup 指令)无相对的类型，但可作为带有宽度的未处理值形式在运行数据域上操作。基本数据类型由编译器管理，而不是由编译了的 Java 程序或在 Java 运行时管理。实际上基本数据在运行时，有带标志也有不带标志的。Java 指令集处理不同的操作码、不同数据类型的操作数，例如：iadd、ladd 、fodd 、dadd 指令都是两数相加，但分别是整数、长整数，单精度浮点和双精度浮点两数相加操作。

此外在 Java 中，还包括：object(对一个 Java Object 的 4 字节的引用)，returnAddress(4 字节)等数据类型。

3．**寄存器**

JVM 寄存器用于保留运算过程中机器的状态。它们与微处理器的寄存器相似。JVM 寄存器包括：

·pc —— Java 程序计数器

·optop —— Java 操作数栈顶端指针

·frame —— 当前运行方法的执行环境指针

·vars —— 当前运行方法的第 0 个局部变量的指针

JVM 定义每个寄存器都是 32 位。某些 Java 实现可以不使用全部寄存器，例如：从 Java 源代码到本地码的汇编器就不包含 pc。

JVM 是基于栈的操作，因此它不定义或使用寄存器来传递或获取参数，这是为了实现指令集精简和紧密，而采取在宿主处理器中不放入过多寄存器（例如 Intel 486)的措施。

4．**Java 栈**

为方便起见，这里分为局部变量、执行环境、异常三部分来叙述。

JVM 是一个基于栈的机器。利用 Java 栈提供操作参数、返回值、传递参数给方法等。Java 栈结构与传统编程语言的栈结构相同，它保存方法调用所需的相关信息，嵌套方法调用的一些结构则被压在方法调用栈中。

每个 Java 栈结构包括三部分即局部变量、运行环境和操作数栈，其中一个或多个部分可为空。局部变量和运行环境在方法调用时字节固定，而操作数栈则随方法运行时不同而不同。下面分别讨论：

(1) 局部变量

每个 Java 栈结构有一套局部变量。它们按变量寄存器顺序寻址，所以可看成是一个数组，局部变量都是 32 位。

长整和双精度浮点数被认为是占两个局部变量，但由第一个局部变量索引确定(例如：带索引号为第 n 的局部变量，若是一个双精度浮点数即按 n 和 n＋1 顺序存储)。64 位值在局部变量中实现时，是采用适当方法，分割长整和双精度浮点数为两个寄存器。JVM 通过指令把局部变量值存入操作数栈或从操作数栈中获取值放入局部变量。

(2) 运行环境

运行环境是用于保存解释器对 Java 字节码进行解释时所需的信息，它包含上次调用方法、局部变量的指针和操作数栈顶和底的指针，附加的预引用信息(例如 debugging)也属于运行环境。

(3) 操作数栈

操作数栈是一个 32 位宽的 FIFO 栈，被用来存储参数和 JVM 参数指令返回的值。例如 iadd 指令两个整数相加，被加的整数是操作数栈中顶端 2 个字，通过旧指令压入，两个整数由栈中弹出，相加后将它们的和压回操作栈。长整型和双精浮点数逻辑上是单个虚拟机操作数，但在操作数栈中占两个物理入口。每种指令的基本数据类型说明该指令的操作数类型，也就是说，操作数必须按它们的类型进行运算，如果压入两个整数却按长整数运算对待这将产生非法的操作信息。

在大多数情况下操作数栈顶和 Java 栈顶是一致的，所以我们只称从"栈"弹出或压入，上下文和操作中的数据也会清晰地体现这一点。

(4) 异常

每个 Java 方法有一个与 catch 语句相关的列表。每个 catch 语句描述指令活动范围，被处理的异常的类型和一块要处理的代码。当一个异常产生时，就从当前方法的接收列表中，搜寻匹配。如果在指令范围内有引发异常的指令，以及产生的异常是一个 catch 语句处理的异常类型的子类型，则异常匹配一个 catch 语句。如果一个匹配语句被找到，则系统转移到处理程序，如果没找到，则当前栈结构被弹出，再次重复以上操作，直到搜索完当前方法的所有嵌套的 catch 子句。

列表中接收语句的顺序很重要，解释器第一个匹配语句匹配后就转移到相应的处理程序。

5. 无用单元收集堆

Java 堆是一个运行时所需的数据域，类实例即创建对象时，从中分配空间。Java 语言被设计成具有收集无用单元的能力，程序员不能显式释放对象。Java 不预测无用单元的类别，根据不同的系统设备采用不同的回收算法。

Java 对象总是通过句柄，间接地进行查询和操作，句柄是在无用单元收集堆之外分配区域的指针。

6. 方法区

方法区类似于存储传统语言中编译代码的区域，或是 UNIX 进程中存储的正文段。它保存方法码(为编译过的 Java 码)和符号表等。

7. 常数池

和每个类相关的是一个常数池，常数池包含所有域名称、方法名和在类中方法所需的其他信息。在后面的类文件格式中，有一张包含常数池类型和它们的相关值的表，其含义是：当从存储区先读入类时，类结构与常数池相关的有 2 个域：

nconstants 域 表明在这个类组常数池中常数个数；

constant-info. constants-offset 域 包含一个从类起始处到类中描述常数数据的整型偏移量；

constant-pool[0] 可按任何目的使用；

constant-pool[1] .... constant-pooL[nconstouts-l] 以类对象中由 constant-info. constants. offset 为起始的字节序列。每个字节序列包含一个"类型"域，其后跟着 1 个或多个在特定域中描述更多细节的字类型相关的字节。

8. 局限性

JVM 在设计上对 Java 执行程序有如下约束条件：

·32 位的指针和栈限定 JVM 的内部地址最大为 4G。

·对于分支和跳转指令带符号位的 16 位偏移量，限制了 Java 方法的字节 数为 32K。

·无符号 8 位局部变量索引，限制了局部变量数，因此每个 Java 栈 frame 为 256 个。

·带符号位的 16 位索引的常数池，限制了每个方法常数池入口数为 32K。

9. 关于快指令：(优化时用)

·无符号 8 位偏移量，限定了对象在一个类中方法数为 256。

·无符号 8 位自变量计数，限定了回调方法的参数字节最大为 256×32bit。

(一个长整或双精度参数每个占 2 个字)

10. Java 指令集的解释器

JVM 的指令集的执行一般使用常规方法来实现，如编译本地码或解释方式。初级 Java 工具包括一个指令集解释器，解释器把编译过 Java 码看成 JVM 指令字节流，以指令为单位，逐条地解释运行。

解释器内部循环，用 do，while 语句表示为：

do{

取一个字节

根据字节含义执行一个动作

}while(是否继续循环)

JVM 指令格式，包括长、短指令描述和 虚拟栈的格式

①短指令语言格式

操作码 = number
操作数 1
操作数 2
.....

② 长指令语言格式

xbc ., value1, value2

xde ..., value3

更长的格式包括解释指令功能和指明在执行中被引

发的任何异常。在上述的 语法表中每条一般是 8 位宽。

⑵虚拟栈的格式

在操作数栈上,一条指令执行结果,若按以下文本描述,从左到右栈其深度也随之增长操作数栈上字全为 32 位数。

... , value1, value2 xde ... , value3

value2 是在栈顶 value1 在下面。例如对于作为 2 个 32 位数指令的执行结果, value1 和 value2 从栈中被弹出并且被由指令计算好的 value3 代替(这时 value3 处于栈顶位置)。栈其余部分(由省略号表示的) 不受指令执行影响。长整和双精浮点总是在操作数栈中占 2 个字,例如:

... xde ... , value - word1, value - word2

对于长整数和双精度浮点数,按一定方式把它们分成 word1 和 word2 二部分,并以适当的顺序完成相应的操作。

11. 协议

大多数 JVM 操作运算是从栈里取出它们的操作数,并且把其操作结果放回栈里。依照惯例,文本描述一般不涉及栈单元的属性,即说明它们是一个运算的源还是目标,只有需要时才被提及。例如:iload 指令是" Load integer from Local variable" 的简写, 含义是作为整数被装入栈, iadd 指令被描述成"整数和"等。一个计算控制流的指令可被用来把 VM 指针 pc 推进到下一指令的操作码, 只有那些确实影响控制流的指令,将会明确指出它们对 pc 的影响。

三、Java 虚拟机的类文件组成

类文件用来支持 Java 类和界面的编译版本。相应的 Java 解释器必须能处理与下面规范一致的类文件。

Java 的类文件由宽度为 8 bit 的字节流组成, 所有 16 和 32 位为单位的量均分别以 2 和 4 个 8 位字节组成。字节按 big-endian 格式编码。类文件格式以 C 语言结构类似, 但又不同于 C 语言的结构, 不同之处如下:

·结构的各部分之间没有组合(联合)

·结构的每个域都是可变字节

·数组也是可变字节, 这种情况数组前的区域给出数组大小, 分别用 U1、U2 和 U4 表示无符号的 1、2、和 4 字节数。

下列伪结构给出类文件格式的高级描述:

```
Class{
u4              magic;
u4              version;
u2              constant-pool-count;
cp-info         constant-pool[constant-pool-count - 1];
u2              access-flags;
u2              this-class;
u2              super-class;
u2              interfaces-count;
u2              interfaces[interfaces-count];
u2              fields-count;
field-info      fields[fields-count];
u2              methods-count;
method-info     methods[methods-count];
u2              attributes-count;
attribute-info  attributes[attribute-count];
}
```

四、Java 虚拟机与 Java 芯片

如前所述, 目前的 Java 语言的运行是基于 Java 虚拟机的 Java 解释程序,是用软件实现的。在速度上比 c/c++ 的慢, 为了进一步推广 Java 的应用, SUN 公司正在设计一系列的 Java 芯片, 也就是要在 VLSI 芯片, 用硬件直接运行 Java 虚拟机的指令系统。显然这样 Java bytecode 的执行会比软件模拟的 Java 虚拟机快得多。

Java 芯片计划开发三种: 最小的是 picoJava, 其次是 microJava, 功能最强的是 ultraJava 。Java 语言、Java OS 和 Java 芯片的有机结合对 Java 在各种应用领域中都有光辉前景。

(全文完)

参考资料

[1] SUN 公司 " HotJava(tm) Documentation " ,1995

[2] SUN 公司 " The Java Developers Kit " ,1995

[3] SUN 公司 " The Java Language: A White Paper " , 1995

[4] Symatech 公司 " cafe for Java " ,1995