目录

[第二章：JVM结构 2](#_Toc13060947)

[2.1类文件格式 2](#_Toc13060948)

[2.2数据类型 2](#_Toc13060949)

[2.3基本类型和值 2](#_Toc13060950)

[2.3.1整型和值 3](#_Toc13060951)

[2.3.2 浮点型，值集，值 3](#_Toc13060952)

[2.3.3 returnAddress类型和值 5](#_Toc13060953)

[2.3.4 boolean类型 5](#_Toc13060954)

[2.4引用类型和值 6](#_Toc13060955)

[2.5运行时数据区 6](#_Toc13060956)

[2.5.1计数器PC 6](#_Toc13060957)

[2.5.2 Java虚拟机栈 7](#_Toc13060958)

[2.5.3 Head堆 7](#_Toc13060959)

[2.5.4 方法区 8](#_Toc13060960)

[2.5.5 运行时常量池 8](#_Toc13060961)

[2.5.6 本地方法栈 9](#_Toc13060962)

[2.6 栈帧 9](#_Toc13060963)

[2.6.1 局部变量表 10](#_Toc13060964)

[2.6.2 操作数栈 11](#_Toc13060965)

[2.6.3 动态链接 11](#_Toc13060966)

[2.6.4 。方法调用完成 12](#_Toc13060967)

[2.6.5 突然方法调用完成(可能异常？) 12](#_Toc13060968)

[2.7 对象表示 12](#_Toc13060969)

# 第二章：JVM结构

## 2.1类文件格式

由Java虚拟机执行的编译代码使用与硬件和操作系统无关的二进制格式表示，通常(但不一定)存储在文件中，称为类文件格式。类文件格式精确地定义了类或接口的表示形式，包括一些细节，比如字节顺序，这些在特定于平台的目标文件格式中可能被认为是理所当然的。

## 2.2数据类型

与Java编程语言一样，Java虚拟机操作两种类型:基本类型和引用类型。相应地，有两种类型的值可以存储在变量中，作为参数传递，由方法返回并操作:基本值和引用值。

Java虚拟机期望几乎所有类型检查都在运行时之前完成，通常由编译器完成，而不必由Java虚拟机本身完成。基本类型的值不需要被标记或以其他方式可检查以在运行时确定它们的类型，或者与引用类型的值区分。相反，Java虚拟机的指令集使用旨在对特定类型的值进行操作的指令来区分其操作数类型。例如，iadd，ladd，fadd和dadd都是Java虚拟机指令，它们添加两个数值并生成数值结果，但每个指令都专门用于其操作数类型：int，long，float和double。有关Java虚拟机指令集中类型支持的摘要，请参见§2.11.1。Java虚拟机包含对对象的显式支持。对象是动态分配的类实例或数组。对对象的引用被视为具有Java虚拟机类型引用。类型引用的值可以被认为是指向对象的指针。可能存在多个对象的引用。始终通过类型引用的值操作，传递和测试对象。

## 2.3基本类型和值

Java虚拟机支持的原始数据类型是数字类型，布尔类型（§2.3.4）和returnAddress类型（§2.3.3）。数字类型由整数类型（§2.3.1）和浮点类型（§2.3.2）组成

整数类型是：

•byte，其值为8位带符号的二进制补码整数，其默认值为零

•short，其值为16位有符号二进制补码整数，其默认值为零

•int，其值为32位带符号的二进制补码整数，其默认值为零

•long，其值为64位带符号的二进制补码整数，其默认值为零

•char，其值为16位无符号整数，使用UTF-16编码，其默认值为空码点('\u0000')



浮点数类型是：

•float，其值是浮点值集的元素，或者，如果支持，则为float-extended-exponent值集，其默认值为正零

•double，其值是double值集的元素，或者，如果支持，则为double-extended-exponent值集，其默认值为正零

布尔类型的值对真值true和false进行编码，默认值为false。

Java®虚拟机规范的第一版没有将布尔值视为Java虚拟机类型。但是，布尔值在Java虚拟机中的支持有限。Java®虚拟机规范的第二版通过将布尔值视为一种类型来澄清该问题。

returnAddress类型的值是指向Java虚拟机指令的操作码的指针。在基本类型中，只有returnAddress类型与Java编程语言类型没有直接关联。

### 2.3.1整型和值

JVM中整型的值：

• For byte, from -128 to 127 (-27 to 27 - 1), inclusive  
• For short, from -32768 to 32767 (-215 to 215 - 1), inclusive  
• For int, from -2147483648 to 2147483647 (-231 to 231 - 1), inclusive  
• For long, from -9223372036854775808 to 9223372036854775807 (-263 to 263 - 1), inclusive  
• For char, from 0 to 65535, inclusive

### 2.3.2 浮点型，值集，值

浮点类型是float和double，它们在概念上与32位单精度和64位双精度格式IEEE 754值和IEEE二进制浮点运算标准（ANSI / IEEE中规定的操作）相关联。Std.754-1985，New York）。

IEEE 754标准不仅包括正负符号幅度数，还包括正负零，正无穷大和负无穷大，以及特殊的非数字值（以下简称为“NaN”）。NaN值用于表示某些无效操作的结果，例如将零除以零。(主要：浮点型是IEEE754标准中值类型的一部分。IEEE754包含float、double，还包含float-extended-exponent、double-extended-exponent、NaN等)

Java虚拟机的每个实现都需要支持两组标准的浮点值，称为浮点值集和双值集。此外，Java虚拟机的实现可以选择支持两个扩展指数浮点值集中的一个或两个，称为float-extended-exponent值集和double-extended-exponent值集。在某些情况下，可以使用这些扩展指数值集代替标准值集来表示float或double类型的值。

任何浮点值集的有限非零值都可以表示为*s* ⋅*m* ⋅2(e - N + 1)，其中s为+1或-1，m为小

于2N的正整数， e是Emin = - (2K-1-2)和Emax = 2K-1-1之间的整数，其中N和K是取决于值集

的参数。有些值可以用多种方式表示在这种形式中;例如，假设值集中的值v可能使用某些

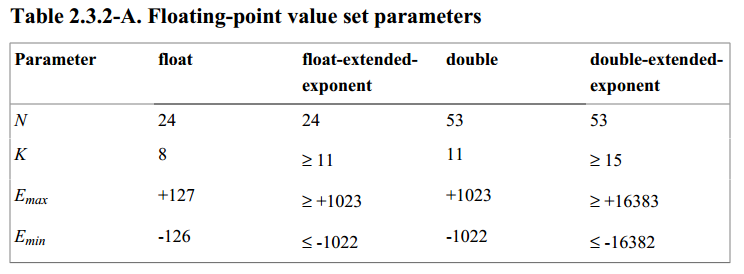
值以此形式表示s，m和e，那么如果m是偶数且e小于2K-1，则可以将m减半并将e增加

1以产生相同值v的第二表示。这种形式的表示是如果m≥2N-1则称为归一化;否则该表示被

称为非规范化。如果值集中的值不能以m≥2N-1的方式表示，则该值被称为非正规化值，因

为它没有归一化表示。

表2.3.2-A总结了对两个必需和两个可选浮点值集的参数N和K（以及导出参数Emin和Emax）的约束。



如果实现支持一个或两个扩展指数值集，则对于每个支持的扩展指数值集，存在特定的依赖于实现的常量K，其值受表2.3.2-A的约束;该值K反过来决定了Emin和Emax的值。四个值集中的每一个不仅包括上面归于它的有限非零值，还包括正零，负零，正无穷大，负无穷大和NaN的五个值。请注意，表2.3.2-A中的约束被设计为浮点值集的每个元素都必须也是float-extended-exponent值集，double值集和double-extended-exponent值的元素。组。同样，double值集的每个元素也必须是double-extended-exponent值集的元素。每个扩展指数值集的指数值范围都大于相应的标准值集，但没有更高的精度。

浮点值集的元素正是可以使用IEEE 754标准中定义的单浮点格式表示的值，除了只有一个NaN值（IEEE 754指定224-2个不同的NaN值）。双值集的元素恰好是可以使用IEEE 754标准中定义的双浮点格式表示的值，除了只有一个NaN值（IEEE 754指定253-2个不同的NaN值）。

但请注意，此处定义的float-extended-exponent和double-extended-exponent值集的元素分别不对应于可使用IEEE 754单扩展和双扩展格式表示的值。除了浮点值必须以类文件格式表示（§4.4.4，§4.4.5）之外，本规范不要求对浮点值集的值进行特定表示。

float，float-extended-exponent，double和double-extended-exponent值集不是类型。对于Java虚拟机的实现来说，使用float值集的元素来表示float类型的值总是正确的。但是，在某些上下文中可能允许实现使用float-extended-exponent值集的元素。类似地，对于实现来说，使用double值集的元素来表示double类型的值总是正确的;但是，在某些上下文中，实现可能允许使用双扩展指数值集的元素。

除NaNs外，浮点值集的值是有序的。当从小到大排列时，它们是负无穷大，负有穷值，正负零，正有穷值和正无穷大。

浮点正零和浮点负零比较相等，但还有其他操作可以区分它们;例如，将1.0除以0.0会产生正无穷大，但将1.0除以-0.0会产生负无穷大。

NaNs是无序的，因此如果其操作数中任何一个或两个都是NaN，则数值比较和数值相等测试的值为false。特别是，当且仅当值为NaN时，对值自身的数值相等性的测试具有值false。如果任一操作数是NaN，则对数值不等式的测试值为true。(Nan != NaN)

### 2.3.3 returnAddress类型和值

JVM的jsr，ret和jsr\_w指令（§jsr，§ret，§jsr\_w）使用returnAddress类型。returnAddress类型的值是指向Java虚拟机指令的操作码的指针。与数字基本类型不同，returnAddress类型不对应任何Java编程语言类型，并且不能由正在运行的程序修改。

### 2.3.4 boolean类型

虽然Java虚拟机定义了一个布尔类型，但JVM只为它提供了非常有限的支持。没有JVM指令专门用于操作布尔值。相反，Java编程语言中对布尔值进行操作的表达式被编译为使用Java虚拟机int数据类型的值。Java虚拟机直接支持布尔数组。它的newarray指令（§newarray）可以创建布尔数组。使用字节数组指令baload和bastore（§baload，§bastore）访问和修改类型为boolean的数组。

在oracle的JVM实现中，在java编程语言中的布尔数组被编译成JVM byte数据，一个布尔数据8bits

Java虚拟机编译布尔数组组件时，用1表示true，0表示false。其中Java编程语言布尔值由编译器映射到Java虚拟机类型int的值，编译器必须使用相同的编码。

## 2.4引用类型和值

有三种引用类型：class类型，array类型和interface类型。它们的值分别是对动态创建的类实例，数组或类实例或实现接口的数组的引用。

数组类型由具有单个维度的组件类型组成（其长度不是由类型给出的）。数组类型的组件类型本身可以是数组类型。如果从任何数组类型开始，考虑其组件类型，然后（如果它也是数组类型）该类型的组件类型，依此类推，最终必须达到不是数组类型的组件类型; 这称为数组类型的元素类型。数组类型的元素类型必须是基本类型，类类型或接口类型。

引用值也可以是特殊的空引用，对无对象的引用，这里将用null表示。null引用最初没有运行时类型，但可以转换为任何类型。引用类型的默认值为null。该规范没有强制要求编码null的具体值。

## 2.5运行时数据区

Java虚拟机定义了在程序执行期间使用的各种运行时数据区域。其中一些数据区域是在Java虚拟机启动时创建的，仅在Java虚拟机退出时销毁。其他数据区域是每个线程。线程数据区域是在线程退出时创建和销毁线程时创建的。

### 2.5.1计数器PC

Java虚拟机可以同时支持许多执行线程（JLS§17）。每个Java虚拟机线程都有自己的pc（程序计数器）寄存器。在任何时候，每个Java虚拟机线程都在执行单个方法的代码，即该线程的当前方法（第2.6节）。如果该方法不是本地方法，则pc寄存器包含当前正在执行的Java虚拟机指令的地址。如果线程当前正在执行的是本地方法，则Java虚拟机的pc寄存器的值是undefined。Java虚拟机的pc寄存器足够宽，可以在特定平台上保存returnAddress或本地指针。

### 2.5.2 Java虚拟机栈

每个Java虚拟机线程都有一个私有Java虚拟机栈，与线程同时创建。Java虚拟机栈存储栈帧（第2.6节）。Java虚拟机堆栈类似于传统语言的堆栈，例如C：它保存局部变量和部分结果，并在方法调用和返回中起作用。因为除了push和pop栈帧之外，JVM永远不会直接操作Java虚拟机栈，所以可以对栈进行堆分配 (???) 。Java虚拟机堆栈的内存不需要是连续的。

此规范允许Java虚拟机堆栈具有固定大小或根据计算的需要动态扩展和收缩。如果Java虚拟机堆栈具有固定大小，则可以在创建该堆栈时独立选择每个Java虚拟机堆栈的大小。

Java虚拟机实现可以为程序员或用户提供对Java虚拟机堆栈的初始大小的控制，以及在动态扩展或收缩Java虚拟机堆栈的情况下，控制最大和最小大小。

以下异常条件与Java虚拟机栈相关联：

• 如果线程中的计算需要比允许的更大的Java虚拟机栈，则Java虚拟机会抛出StackOverflowError。

• 如果可以动态扩展Java虚拟机栈，并且尝试进行扩展但可以使内存不足以实现扩展，或者可以使内存不足以创建初始Java如果可以动态扩展Java虚拟机栈， 尝试进行扩展，但可以使内存不足以实现扩展，或者如果可以使内存不足以为新线程创建初始Java虚拟机栈，则Java虚拟机会抛出OutOfMemoryError。

### 2.5.3 Head堆

Java虚拟机具有在所有Java虚拟机线程之间共享的堆。堆是运行时数据区，从中分配所有类实例和数组的内存。

堆是在虚拟机启动时创建的。对象的堆存储由自动存储管理系统（称为垃圾收集器）回收; 对象永远不会被显式释放。Java虚拟机假设没有特定类型的自动存储管理系统，可以根据实现者的系统要求选择存储管理技术。堆可以具有固定大小，或者可以根据计算的需要进行扩展，并且如果不需要更大的堆，则可以收缩。堆的内存不需要是连续的。(追求响应时间就serial、par new，追求吞吐量就parallel Scavenge；追求stw最小就G1)

Java虚拟机实现可以为程序员或用户提供对堆的初始大小的控制，以及如果可以动态扩展或收缩堆，则控制最大和最小堆大小。

以下异常条件与Java虚拟机堆相关联：

• 如果线程中的计算需要超过GC可用的堆，则Java虚拟机会抛出OutOfMemoryError。

### 2.5.4 方法区

Java虚拟机具有在所有Java虚拟机线程之间共享的方法区域。方法区域类似于传统语言的编译代码的存储区域或类似于操作系统进程中的“文本”段。它存储每类结构，例如运行时常量池，字段和方法数据，以及方法和构造函数的代码，包括类和接口初始化以及实例初始化（第2.9节）中使用的特殊方法。

方法区域是在虚拟机启动时创建的。虽然方法区域在逻辑上是堆的一部分，但是简单的实现可能选择不垃圾收集或压缩它。本规范未规定方法区域的位置或用于管理编译代码的策略。方法区域可以是固定大小的，或者可以根据计算的需要进行扩展，并且如果不需要更大的方法区域，则可以缩小方法区域。方法区域的内存不需要是连续的。逻辑上是连续的。

Java虚拟机实现可以为程序员或用户提供对方法区域初始大小的控制，在方法区域大小可变的情况下，还可以提供对最大和最小方法区域大小的控制。

以下异常条件与Java虚拟机方法区相关联：

• 如果方法区域中的内存不能满足分配请求，Java虚拟机将抛出OutOfMemoryError错误。

### 2.5.5 运行时常量池

运行时常量池是类文件中常量池表的每个类或每个接口的运行时表示(§4.4)。它包含几种常量，从编译时已知的数值常量到必须在运行时解析的方法和字段引用。运行时常量池的功能类似于传统编程语言的符号表，尽管它包含的数据范围比典型符号表更广。

每个运行时常量池都在Java虚拟机的方法区中进行分配(§2.5.4)。类或接口的运行时常量池是在Java虚拟机创建类或接口时构造的(§5.3)。

下列异常情况与类或接口的运行时常量池的构造有关:

* 在创建类或接口时，如果构建运行时常量池所需的内存超过Java虚拟机法

区域的可用内存，Java虚拟机将抛出OutOfMemoryError错误。

有关运行时常量池构造的信息，请参见§5(加载、链接和初始化)。

### 2.5.6 本地方法栈

Java虚拟机的实现可以使用传统的栈，俗称“C栈”，以支持本地方法（用Java编程语言以外的语言编写的方法）。本地方法栈也可以用于以诸如C语言的Java虚拟机的指令集的解释器的实现来使用。无法加载本地方法并且本身不依赖于传统堆栈的Java虚拟机实现不需要提供本地方法栈。如果提供，则通常在创建每个线程时为每个线程分配本地方法栈。

此规范允许本地方法栈具有固定大小或根据计算的需要动态扩展和收缩。如果本地方法栈具有固定大小，则可以在创建该堆栈时独立地选择每个本地方法栈的大小。

Java虚拟机实现可以为程序员或用户提供对本地方法栈的初始大小的控制，以及在不同大小的本地方法栈的情况下，控制最大和最小方法堆栈大小。

下列异常情况与本地方法栈相关:

• 如果线程中的计算需要比允许的更大的本地方法栈，则Java虚拟机会抛出StackOverflowError。

• 如果可以动态扩展本地方法栈，并尝试进行本地方法栈扩展但可用内存不足，或者如果可用内存不足以 为新线程创建初始本地方法栈，则Java虚拟机将抛出OutOfMemoryError异常。

## 2.6 栈帧

栈帧用于存储数据和部分结果，以及执行动态链接，返回方法的值和调度异常。每次调用方法时都会创建一个新栈帧。当方法调用完成时，无论该完成是。还是突然（它会抛出未捕获的异常），栈帧都将被销毁。栈帧是从创建栈帧的线程的JVM堆栈（第2.5.2节）中分配的。每个栈帧都有自己的局部变量数组（第2.6.1节），它自己的操作数堆栈（第2.6.2节），以及对当前方法类的运行时常量池（第2.5.5节）的引用。

可以使用附加的特定于实现的信息来扩展帧，例如调试信息。

局部变量数组和操作数堆栈的大小在编译时确定，并与与帧相关的方法的代码一起提供（第4.7.3节）。因此，栈帧数据结构的大小仅取决于Java虚拟机的实现，并且可以在方法调用上同时分配这些结构的存储器。

只有一个栈帧（执行方法的栈帧）在给定控制线程中的任何点处都是活动的。该栈帧被称为当前栈帧，并且其方法被称为当前方法。定义当前方法的类是当前类。局部变量和操作数堆栈的操作通常参考当前栈帧。

如果栈帧的方法调用另一个方法或其方法完成，则栈帧将不再是当前栈帧。调用方法时，会创建一个新栈帧，并在控制转移到新方法时变为当前栈帧。在方法返回时，当前栈帧将其方法调用的结果（如果有）传递回前一栈帧。然后当前一栈帧成为当前栈帧时丢弃当前栈帧。

请注意，由线程创建的栈帧对于该线程是本地的，并且不能被任何其他线程引用。

### 2.6.1 局部变量表

每个栈帧（第2.6节）包含一个称为局部变量的变量数组。栈帧的局部变量数组的长度在编译时确定，并以类或接口的二进制表示形式提供，同时提供与栈帧相关的方法的代码（第4.7.3节）。

单个局部变量可以包含boolean，byte，char，short，int，float，reference或returnAddress类型的值。一对局部变量可以包含long或double类型的值。

局部变量可以通过索引寻址找到。第一个局部变量的索引为零。当且仅当该整数在0到1之间且小于局部变量数组的大小时，整数才被认为是局部变量数组的索引。

long类型或double类型的值占用两个连续的局部变量。只能使用较小的索引来寻址处理这样的值。例如，存储在索引n的局部变量数组中的double类型的值实际上占用索引为n和n + 1的局部变量;但是，无法加载索引n + 1处的局部变量。它可以被存储到n + 1处。但是，这样做会使局部变量n的内容无效。

Java虚拟机不要求n是偶数。直观地说，long和double类型的值不需要在局部变量数组中进行64位对齐。实现者可以使用为该值保留的两个局部变量自由决定表示此类值的适当方法。

Java虚拟机使用局部变量在方法调用上传递参数。在类方法调用中，任何参数都在从局部变量0开始的连续局部变量中传递。在实例方法调用中，局部变量0总是用于传递对调用实例方法的对象的引用（在Java编程语言的this）。随后，任何参数都在从局部变量1开始的连续局部变量中传递。

### 2.6.2 操作数栈

每个栈帧（第2.6节）包含一个后进先出（LIFO）堆栈，称为其操作数栈。栈帧的操作数栈的最大深度在编译时确定，并与用于与栈帧相关的方法的代码一起提供（第4.7.3节）。

在上下文清楚的地方，我们有时会将当前栈帧的操作数栈简称为操作数栈。

当创建包含它的栈帧时，操作数栈为空。Java虚拟机提供指令以将局部变量或字段中的常量或值加载到操作数栈上。其他Java虚拟机指令从操作数栈中获取操作数，对它们进行操作，并将结果推回操作数栈。操作数栈还用于准备要传递给方法和接收方法结果的参数。

例如，iadd指令（§iadd）将两个int值相加。它要求要添加的int值是操作数栈的前两个值，由前面的指令推送到那里。两个int值都从操作数栈中弹出。它们被添加，它们的总和被推回到操作数栈上。子计算可以嵌套在操作数栈上，从而产生可以由包含计算使用的值。

操作数栈上的每个条目都可以包含任何Java虚拟机类型的值，包括long类型或double类型的值。

必须以适合其类型的方式操作操作数栈中的值。例如，不可能推送两个int值，然后将它们视为long或推送两个float值，然后使用iadd指令添加它们。少量Java虚拟机指令（dup指令（§dup）和swap（§swap））作为原始值在运行时数据区域上运行，而不考虑它们的特定类型;这些指令的定义方式使它们不能用于修改或分解单个值。操作数栈操作的这些限制是通过类文件验证（第4.10节）强制执行的。在任何时间点，操作数栈都具有相关联的深度，其中long或double类型的值对深度贡献两个单位，而任何其他类型的值贡献一个单位。

### 2.6.3 动态链接

每个栈帧（第2.6节）包含指向针对当前方法类型的运行时常量池（第2.5.5节）的引用，用于支持方法代码的动态链接。方法的类文件代码是指要调用的方法和要通过符号引用访问的变量。动态链接将这些符号方法引用转换为具体的方法引用，根据需要加载类以解析尚未定义的符号，并将变量访问转换为与这些变量的运行时位置相关联的存储结构中的适当偏移。

这种方法和变量的后期绑定使得在方法调用的其他类中的更改不太可能破坏此代码。

### 2.6.4 方法调用完成

如果调用不会直接从Java虚拟机或执行显式throw语句引发异常（第2.10节），则方法调用会。完成。 如果当前方法的调用。完成，则可以将值返回给调用方法。 当被调用的方法执行其中一个返回指令（第2.18.8节）时，就会发生这种情况，返回指令的选择必须适合于返回值的类型（如果有的话）。

在这种情况下，当前栈帧（第2.6节）用于恢复调用者的状态，包括其局部变量和操作数栈，调用者的程序计数器适当地递增以跳过方法调用指令。 然后执行在调用方法的帧中。继续，返回值（如果有）被推送到该帧的操作数堆栈。

### 2.6.5 终止方法调用完成(异常)

如果在方法中执行Java虚拟机指令导致Java虚拟机抛出异常（第2.10节），并且该异常未在该方法中处理，则方法调用会突然完成(个人认为叫执行突然终止更为恰当)。 执行athrow指令（§athrow）也会导致显式抛出异常，如果当前方法未捕获异常，则会导致突然的方法调用完成。 突然完成的方法调用永远不会向其调用者返回值。

## 2.7 对象表示

Java虚拟机不对对象的有任何特定内部结构要求。

在Oracle的一些Java虚拟机实现中，对类实例的引用是指向句柄的指针，该句柄本身是一对指针：一个指向包含对象方法的表和指向表示代表对象类型的Class对象的指针，另一个是从堆为对象数据分配的内存。

2.8 浮点运算

Java虚拟机包含IEEE二进制浮点运算标准（ANSI / IEEE Std.754-1985，New York）中规定的浮点运算的子集。

2.8.1 浮点运算和IEEE754

Java虚拟机支持的浮点运算与IEEE 754标准之间的主要区别是：

• Java虚拟机的浮点运算不会抛出异常，陷阱或以其他方式发出IEEE 754无效操作异常条件，除零，溢出，下溢或不精确的信号。 Java虚拟机没有信令NaN值。

• Java虚拟机不支持IEEE 754信令浮点比较。

• Java虚拟机的舍入操作始终使用IEEE 754舍入到最近模式。不精确的结果四舍五入到最接近的可表示值，并且连接到具有零最低有效位的值。这是IEEE 754默认模式。但Java虚拟机指令将浮点类型的值转换为整数类型的值，舍入为零。 Java虚拟机没有提供任何改变浮点舍入模式的方法。

• Java虚拟机不支持IEEE 754单扩展或双扩展格式，除非可以说double和double-extendedexponent值集支持单扩展格式。 float-extended-exponent和double-extended-exponent值集（可选择支持）与IEEE 754扩展格式的值不对应：IEEE 754扩展格式需要扩展精度以及扩展指数范围。

2.8.2 浮点模型

每个方法都有一个浮点模式，它可以是FP-strict，也可以不是FPstrict。方法的浮点模式由定义方法的method\_info结构（第4.6节）的access\_flags项的ACC\_STRICT标志的设置确定。设置该标志的方法是FP严格的;否则，该方法不是FP严格的。

请注意，ACC\_STRICT标志的这种映射意味着由JDK 1.1版或更早版本中的编译器编译的类中的方法实际上不是FP严格的。

当调用创建包含操作数堆栈的帧的方法具有浮点模式时，我们将操作数堆栈称为具有给定的浮点模式。类似地，当包含该指令的方法具有该浮点模式时，我们将Java虚拟机指令称为具有给定的浮点模式。

如果支持float-extended-exponent值集（第2.3.2节），则非FP-strict的操作数堆栈上float类型的值可以超出该值集，除非值集转换禁止（§2.8.3） ）。如果支持双扩展指数值集（第2.3.2节），则非FP严格的操作数堆栈上的double类型值可以超出该值集，除非值集转换禁止。

在所有其他上下文中，无论是在操作数堆栈上还是在其他位置，并且无论浮点模式如何，float和double类型的浮点值分别只能超出浮点值集和双值集。特别是，类和实例字段，数组元素，局部变量和方法参数只能包含从标准值集中提取的值。

2.8.3 值集转换

在特定情况下，允许或要求支持扩展浮点值集的Java虚拟机的实现，以在扩展值集和标准值集之间映射关联浮点类型的值。这样的值集转换不是类型转换，而是与相同类型关联的值集之间的映射。

在指示值集转换的情况下，允许实现对值执行以下操作之一：

•如果值的类型为float且不是float值集的元素，则它将值映射到float值集的最近元素。

•如果值的类型为double且不是double值集的元素，则会将值映射到double值集的最近元素。

此外，在指示值集转换的情况下，需要执行某些操作：

• 假设执行非FP-strict的Java虚拟机指令会导致float类型的值被推送到FP-strict的操作数堆栈，作为参数传递，或存储到本地变量，字段或数组的一个元素。如果该值不是浮点值集的元素，则将该值映射到浮点值集的最近元素。

• 假设执行非FP-strict的Java虚拟机指令会导致double类型的值被推送到FPstrict的操作数堆栈，作为参数传递，或存储到局部变量，字段或元素中一个数组。如果该值不是double值集的元素，则将值映射到double值集的最近元素。

在方法调用期间传递浮点类型的参数（包括本机方法调用）可能会发生这种所需的值集转换;从非FPstrict的方法返回浮点类型的值到FP-strict的方法;或者将浮点类型的值存储到非FP严格的方法中的局部变量，字段或数组中。

并非扩展指数值集中的所有值都可以精确映射到相应标准值集中的值。如果映射的值太大而无法准确表示（其指数大于标准值集允许的值），则将其转换为相应类型的（正或负）无穷大。如果映射的值太小而不能精确表示（其指数小于标准值集所允许的值），则将其舍入到最接近的可表示的非规范化值或相同符号的零。

值集转换会保留无穷大和NaN，并且不能更改正在转换的值的符号。值集转换对不是浮点类型的值没有影响。