第六章

Java 虚拟机指令集

Java 虚拟机指令包括一个指定要执行的操作的操作码，后面是零个或多个包含要操作的值的操作数。本章详细介绍了每个 Java 虚拟机指令的格式及其执行的操作。

6.1 假设:“必须”的含义

每条指令的描述总是在满足§4(class 文件格式)的静态和结构约束的 Java 虚拟机代码的上下文中给出的。在单个 Java 虚拟机指令的描述中，我们经常声明某些情况“必须”或“不能”是这样的:“value2 必须是 int 类型。” §4(class 文件格式)的约束保证了所有这些期望实际上都将得到满足。如果指令描述中的某些约束(“必须”或“不能”)在运行时未得到满足，则 Java 虚拟机的行为是未定义的。

Java 虚拟机使用 class 文件验证器(§4.10)检查 Java 虚拟机代码在链接时是否满足静态和结构约束。因此，Java 虚拟机将只尝试执行来自有效 class 文件的代码。在链接时执行验证很有吸引力，因为检查只执行一次，大大减少了必须在运行时执行的工作量。如果它们符合 Java 语言规范，Java SE 19 版和 Java 虚拟机规范，Java SE 19 版，那么也可以采用其他实现策略。

6.2 保留的操作码

除了本章后面指定的指令的操作码，它们在 class 文件中使用(§4(class 文件格式))，还有三个操作码保留给 Java 虚拟机内部实现使用。如果将来扩展 Java 虚拟机的指令集，保证这些保留的操作码不会被使用。

其中两个保留操作码，编号 254 (0xfe)和 255 (0xff)，分别具有 impdep1 和 impdep2 的助记符。这些指令旨在为分别在软件和硬件中实现的特定于实现的功能提供“后门”或陷阱。第三个保留操作码是编号 202 (0xca)，具有助记断点，用于调试器实现断点。

尽管这些操作码是保留的，但它们只能在 Java 虚拟机实现中使用。它们不能出现在有效的 class 文件中。调试器或 JIT 代码生成器(§2.13)等工具可能直接与已经加载和执行的 Java 虚拟机代码交互，可能会遇到这些操作码。如果遇到任何这些保留指令，这些工具应该尝试优雅地执行。

6.3 虚拟机错误

当内部错误或资源限制阻止 Java 虚拟机实现本章描述的语义时，它抛出一个对象，该对象是 VirtualMachineError 类的子类的实例。该规范不能预测在哪里可能会遇到内部错误或资源限制，也没有明确规定何时可以报告这些错误。因此，下面定义的任何 VirtualMachineError 子类都可能在 Java 虚拟机操作期间的任何时候被抛出:

- InternalError: Java 虚拟机实现中出现了一个内部错误，这是由于实现虚拟机的软件中的错误、底层主机系统软件中的错误或硬件中的错误造成的。当检测到该错误时，该错误是异步传递的(§2.10)，并且可能发生在程序的任何位置。
- OutOfMemoryError: Java 虚拟机实现已耗尽虚拟内存或物理内存，自动存储管理器无法回收足够的内存来满足对象创建请求。
- StackOverflowError: Java 虚拟机实现耗尽了线程的栈空间，这通常是因为由于执行程序中的错误，线程正在执行无限数量的递归调用。
- UnknownError: 发生了异常或错误，但 Java 虚拟机实现无法报告实际的异常或错误。

6.4 指令说明的格式

在本章中，Java 虚拟机指令由如下所示形式的条目表示，按字母顺序排列，每个条目从一个新的页面开始。

mnemonic

操作	指令的简短描述
格式	mnemonic operand1 operand2 ...
形式	mnemonic = opcode
操作数栈	..., value1, value2→ ..., value3
描述	更长的描述详细描述了对操作数栈内容或常量池条目、执行的操作、结果类型等的约束。
链接异常	如果指令的执行可能引发任何链接异常，则按必须引发的顺序逐行设置这些异常。
运行时异常	如果指令的执行可以引发任何运行时异常，则按必须引发的顺序逐行设置这些异常。 除了为指令列出的链接和运行时异常(如果有)之外，该指令不得引发任何运行时异常，但 VirtualMachineError 或其子类的实例除外。
注释	严格意义上不属于指令规范的注释作为注释放在描述的末尾。

指令格式图中的每个单元表示一个 8 位字节。指令的助记符是它的名字。指令的助记符是它的名字。它的操作码是它的数字表示形式，以十进制和十六进制的形式给出。在 class 文件中的 Java 虚拟机代码中实际上只有数字表示形式。

请记住，有在编译时生成并嵌入 Java 虚拟机指令中的“操作数”，也有在运行时计算并在操作数栈上提供的“操作数”。尽管它们是由几个不同的区域提供的，但所有这些操作数都表示相同的东西:要由正在执行的 Java 虚拟机指令操作的值。通过隐式地从操作数栈中取出许多操作数，而不是在编译后的代码中显式地将它们表示为额外的操作数字节、寄存器号等，Java 虚拟机的代码保持了紧凑。

有些指令作为一组相关指令的成员呈现，共享单一的描述、格式和操作数栈图。因此，一组指令包括几个操作码和操作码助记符;只有族助记符出现在指令格式图中，并且有一个单

独的表单行列出所有成员助记符和操作码。例如，lconst_<l>指令族的表单行为该指令族中的两个指令(lconst_0 和 lconst_1)提供助记符和操作码信息，它们是:

lconst_0 = 9 (0x9)

lconst_1 = 10 (0xa)

在 Java 虚拟机指令的描述中，一条指令的执行对当前帧(§2.6)的操作数栈(§2.6.2)的影响以文本形式表示，栈从左向右增长，每个值分别表示。因此，

…，valuel, value2 T

…, result

显示一个操作，该操作开始时，value2 位于操作数栈的顶部，value1 位于它的下面。作为指令执行的结果，value1 和 value2 从操作数栈中弹出，并被指令计算出来的结果值所取代。操作数栈的其余部分，用省略号(…)表示，不受指令执行的影响。

long 和 double 类型的值由操作数栈上的单个条目表示。

在第一版的 Java 虚拟机规范中，long 和 double 类型的操作数栈上的值在栈图中分别用两个条目表示。

6.5 指令

aaload

操作	从数组加载引用
格式	aaload
形式	aaload = 50 (0x32)
操作数栈	..., arrayref, index → ..., value
描述	arrayref 必须具有类型引用，并且必须引用其组件具有类型引用的数组。index 必须是 int 类型的。arrayref 和 index 都是从操作数栈中弹出的。检索 index 处数组组件中的引用值，并将其压入操作数栈。
运行时异常	如果 arrayref 为空，aaload 抛出 NullPointerException 异常。 否则，如果 index 不在 arrayref 引用的数组的范围内，aaload 指令将抛出 ArrayIndexOutOfBoundsException 异常。

aastore

操作	存储到引用数组
格式	aastore
形式	aastore = 83 (0x53)
操作数栈	..., arrayref, index, value → ...
描述	arrayref必须具有类型引用，并且必须引用其组件具有类型引用的数组。index必须是int类型，value必须是reference类型。arrayref, index和value从操作数栈中弹出。 如果value为空，则该value存储为index处数组的组件。 否则，value为非空。如果value的类型与arrayref引用的数组组件的类型是赋值兼容的，则value存储为index处的数组组件。 以下规则用于确定非空value是否与数组组件类型赋值兼容。如果S是value引用的对象类型，T是数组组件的引用类型，则aastore确定赋值是否兼容，如下所示： • 如果S是类类型，则： – 如果T是类类型，则S必须是与T相同的类，或者S必须是T的子类； – 如果T是接口类型，则S必须实现接口T。 • 如果S是一个数组类型SC[]，即SC类型组件的数组，则： – 如果T是一个类类型，那么T必须是Object。 – 如果T是一种接口类型，那么T必须是数组实现的接口之一(JLS§4.10.3)。 – 如果T是数组类型TC[]，即由TC类型的组件组成的数组，则以下条件之一必须为true： ➢ TC和SC是相同的原生类型。 ➢ TC和SC是引用类型，类型SC可以通过这些运行时规则赋值给TC。
运行时异常	如果 arrayref 为空，aastore 抛出 NullPointerException 异常。 否则，如果index不在arrayref引用的数组的范围内，aastore指令将抛出ArrayIndexOutOfBoundsException异常。 否则，如果arrayref不为空，且非空值的实际类型与数组组件的实

	际类型不是赋值兼容，aastore将抛出ArrayStoreException。

aconst_null

操作	压入 null
格式	*aconst_null*
形式	*aconst_null* = 1 (0x1)
操作数栈	... → ..., `null`
描述	将空对象引用压入操作数栈。
注释	Java 虚拟机并不强制要求 null 的具体值。

aload

操作	从局部变量加载引用
格式	aload index
形式	aload = 25 (0x19)
操作数栈	... → ..., objectref
描述	index是一个无符号字节，它必须是当前帧的局部变量数组的索引(§2.6)。在index处的局部变量必须包含引用。index处的局部变量objectref被推入操作数栈。
注释	aload指令不能用于将returnAddress类型的值从局部变量加载到操作数栈。astore指令（§astore）的这种不对称是有意的。 aload操作码可与wide指令（§wide）结合使用，以使用两字节无符号索引访问局部变量。

aload_<n>

操作	从局部变量加载引用
格式	*aload_<n>*
形式	*aload_0* = 42 (0x2a) *aload_1* = 43 (0x2b) *aload_2* = 44 (0x2c) *aload_3* = 45 (0x2d)
操作数栈	... → ..., objectref
描述	<n>必须是当前帧的局部变量数组的索引（§2.6）。<n>处的局部变量必须包含引用。位于<n>的局部变量中的objectref被推送到操作数栈上。
注释	aload_<n>指令不能用于将returnAddress类型的值从局部变量加载到操作数栈。这种与相应的astore_<n>指令（§astore<n>）的不对称是有意的。 每个aload_ < n > 指令与带索引 < n > 的aload指令相同，只是操作数 < n > 是隐式的。

anewarray

操作	创建引用的新数组
格式	anewarray indexbyte1 indexbyte2
形式	anewarray = 189 (0xbd)
操作数栈	..., count → ..., objectref
描述	count必须是int类型。它从操作数栈中弹出。count表示要创建的数组的组件数。无符号的indexbyte1和indexbyte2用于构建到当前类的运行时常量池中的索引(§2.6)，其中索引值为(indexbyte1<<8)|indexbyte2。索引处的运行时常量池条目必须是对类、数组或接口类型的符号引用。解析命名的类、数组或接口类型(§5.4.3.1)。从垃圾回收堆分配一个具有该类型组件的新数组，长度为count，并将对此新数组对象的引用arrayref推送到操作数栈上。新数组的所有组件都被初始化为空，这是引用类型的缺省值(§2.4)。
链接异常	在解析类、数组或接口类型的符号引用时，可以抛出§5.4.3.1中记录的任何异常。
运行时异常	否则，如果count小于零，则anewarray指令将抛出NegativeArraySizeException。
注释	anewarray指令用于创建对象引用数组的单个维度或多维数组的一部分。

areturn

操作	从方法返回引用
格式	*areturn*
形式	*areturn* = 176 (0xb0)
操作数栈	..., *objectref*→ [empty]
描述	objectref必须具有类型引用，并且必须引用的对象类型与当前方法的返回描述符(§4.3.3)所表示的类型赋值兼容(JLS§5.2)。如果当前方法是一个同步方法，在调用该方法时进入或重新进入的监视器将被更新并可能退出，就像在当前线程中执行一个monitorexit指令(§monitorexit)一样。如果没有抛出异常，objectref从当前帧的操作数栈中弹出(§2.6)，并推入调用者帧的操作数栈中。当前方法的操作数栈上的任何其他值都将被丢弃。 解释器然后恢复调用程序的框架，并将控制权返回给调用程序。
运行时异常	如果Java虚拟机实现没有强制执行§2.11.10中描述的结构化锁定规则，那么如果当前方法是一个同步方法，并且当前线程不是调用该方法时进入或重新进入的监视器的所有者，则areturn抛出IllegalMonitorStateException异常。例如，如果同步方法在其同步的对象上包含monitorexit指令，但没有monitorenter指令，就会发生这种情况。 否则，如果Java虚拟机实现强制执行§2.11.10中描述的结构化锁定规则，并且如果在调用当前方法期间违反了这些规则中的第一条，则areturn将抛出一个IllegalMonitorStateException。

arraylength

操作	获取数组的长度
格式	arraylength
形式	arraylength = 190 (0xbe)
操作数栈	..., objectref→ ..., length
描述	arrayref必须是类型引用，并且必须引用数组。它从操作数栈中弹出。确定其引用的数组的长度。该长度作为一个int推送到操作数栈上。
运行时异常	如果arrayref为空，arraylength指令将抛出NullPointerException。

astore

操作	存储引用到局部变量
格式	astore index
形式	astore = 58 (0x3a)
操作数栈	..., objectref→ ...
描述	index是一个无符号字节，必须是当前帧的局部变量数组的索引(§2.6)。操作数栈顶部的objectref必须是类型returnAddress或类型reference。它从操作数栈中弹出，在index处的局部变量的值被设置为objectref。
注释	在实现Java编程语言的finally子句(§3.13)时，astore指令与returnAddress类型的objectref一起使用。 aload 指令(§loadad)不能用于将一个类型为returnAddress的值从局部变量加载到操作数栈中。这种与astore指令的不对称是有意为之的。 astore操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

astore_<n>

操作	存储引用到局部变量
格式	astore_<n>
形式	astore_0 = 75 (0x4b) astore_1 = 76 (0x4c) astore_2 = 77 (0x4d) astore_3 = 78 (0x4e)
操作数栈	..., objectref→ ...
描述	<n>必须是当前帧的局部变量数组的索引（§2.6）。操作数栈顶部的objectref必须是类型returnAddress或类型reference。它从操作数栈中弹出，并且在<n>处的局部变量的值被设置为objectref。
注释	在实现Java编程语言的finally子句时，astore_<n>指令与类型为returnAddress的objectref一起使用（§3.13）。 aload_<n>指令（§aload<n>）不能用于将returnAddress类型的值从局部变量加载到操作数栈。这种与相应的astore_<n>指令的不对称是有意的。 每个astore_ < n > 指令与带索引 < n > 的astore相同，只是操作数 < n > 是隐式的。

athrow

操作	抛出异常或错误
格式	athrow
形式	athrow = 191 (0xbf)
操作数栈	..., objectref→ objectref
描述	objectref必须具有类型引用，并且必须引用一个对象，该对象是Throwable类的实例或Throwable的子类的实例。它从操作数栈弹出。然后通过搜索当前方法(§2.6)中与objectref的类匹配的第一个异常处理程序来抛出objectref，就像§2.10中的算法给出的那样。如果找到与objectref匹配的异常处理程序，则它包含用于处理此异常的代码的位置。pc寄存器重置到该位置，清除当前帧的操作数栈，将objectref推回操作数栈，然后继续执行。 如果在当前帧中没有找到匹配的异常处理程序，则弹出该帧。如果当前帧表示一个同步方法的调用，在调用该方法时进入或重新进入的监视器将被退出，就像通过执行monitorexit指令(§monitorexit)一样。最后，如果存在这样的帧，则恢复其调用者的框架，并重新抛出objectref。如果不存在这样的帧，则当前线程退出。
运行时异常	如果objectref为空，athrow将抛出一个NullPointerException而不是objectref。 否则，如果Java虚拟机实现没有强制执行§2.11.10中描述的结构化锁定规则，那么如果当前帧的方法是一个同步方法，并且当前线程不是调用该方法时进入或重新进入的监视器的所有者，athrow抛出一个IllegalMonitorStateException而不是之前抛出的对象。例如，如果一个突然完成的同步方法在同步方法的对象上包含一个monitorexit指令，但没有monitorenter指令，就会发生这种情况。 否则，如果Java虚拟机实现强制执行§2.11.10中描述的结构化锁定规则，并且在当前方法调用过程中违反了第一条规则，则athrow抛出一个IllegalMonitorStateException而不是之前抛出的对象。
注释	athrow指令的操作数栈图可能具有误导性:如果在当前方法中匹配了此异常的处理程序，则athrow指令丢弃操作数栈上的所有值，然后将抛出的对象压入操作数栈。但是，如果在当前方法中没有匹配的处理程序，并且异常被抛出到方法调用链的更上层，那么

	处理异常的方法(如果有)的操作数栈将被清除，objectref将被推到空的操作数栈上。从抛出异常的方法到(但不包括)处理异常的方法的所有中间帧都将被丢弃。

baload

操作	从数组加载 byte 或 boolean
格式	baload
形式	baload = 51 (0x33)
操作数栈	..., arrayref, index→ ..., value
描述	arrayref必须是类型reference，并且必须引用其组件类型为byte或boolean的数组。index必须是int类型的。 arrayref和index都是从操作数栈中弹出的。检索index处数组组件中的byte值，符号扩展为int值，并将其压入操作数栈的顶部。
运行时异常	如果arrayref为空，baload抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的边界内，baload指令将抛出ArrayIndexOutOfBoundsException异常。
注释	baload指令用于从byte数组和boolean数组加载值。在Oracle的Java虚拟机实现中，boolean数组——即类型为T_BOOLEAN的数组(§2.2，§newarray)——被实现为8位值的数组。其他实现可能实现打包boolean数组;必须使用这种实现的baload指令来访问这些数组。

bastore

操作	存储到 byte 或 boolean 数组
格式	bastore
形式	bastore = 84 (0x54)
操作数栈	..., arrayref, index, value→ ...
描述	arrayref必须是类型reference，并且必须引用其组件类型为byte或boolean的数组。index和value必须都是int类型。arrayref、index和value从操作数栈中弹出。 如果arrayref引用的数组的组件类型为byte，那么int值将被截断为byte，并存储为按index索引的数组的组件。 如果arrayref引用的数组的组件类型为boolean，则int值将通过value和1进行按位与进行收窄;结果存储为按index索引的数组的组件。
运行时异常	如果arrayref为空，bastore抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的边界内，bastore指令将抛出ArrayIndexOutOfBoundsException异常。
注释	bastore指令用于将值存储到byte数组和boolean数组中。在Oracle的Java虚拟机实现中，boolean数组——即类型为T_BOOLEAN的数组(§2.2，§newarray)——被实现为8位值的数组。其他实现可能实现打包boolean数组;在这样的实现中，bastore指令必须能够将boolean值存储到打包boolean数组中，以及将byte值存储到byte数组中。

bipush

操作	压入 byte
格式	bipush byte
形式	bipush = 16 (0x10)
操作数栈	...→ ...,value
描述	当前的byte被符号扩展为int值。该value被压入操作数栈。

caload

操作	从数组加载 char
格式	caload
形式	caload = 52 (0x34)
操作数栈	..., arrayref, index→ ...,value
描述	arrayref必须是类型reference，并且必须引用其组件类型为char的数组。index必须是int类型的。arrayref和index都是从操作数栈中弹出的。检索index处数组的组件，并且0扩展为int值。该值被压入操作数栈。
运行时异常	如果arrayref为空，则caload抛出一个NullPointerException。 否则，如果index不在arrayref引用的数组的边界内，则caload指令抛出ArrayIndexOutOfBoundsException。

castore

操作	存储到 char 数组
格式	castore
形式	castore = 85 (0x55)
操作数栈	..., arrayref, index, value→ ...
描述	arrayref必须是类型reference，并且必须引用其组件为char类型的数组。index和value都必须是int类型。arrayref、index和value从操作数栈中弹出。int值被截断为char，并存储为按index索引的数组的组成部分。
运行时异常	如果arrayref为空，castore抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的范围内，castore指令将抛出ArrayIndexOutOfBoundsException异常。

checkcast

操作	检查对象是否具有给定的类型
格式	checkcast indexbyte1 indexbyte2
形式	checkcast = 192 (0xc0)
操作数栈	..., objectref→ ..., objectref
描述	objectref必须是类型reference。无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对类、数组或接口类型的符号引用。 如果objectref为空，则操作数栈不变。 否则，命名的类、数组或接口类型将被解析(§5.4.3.1)。如果objectref可以强制转换为解析的类、数组或接口类型，则操作数栈不变;否则，checkcast指令抛出ClassCastException。 以下规则用于确定是否可以将非空的objectref强制转换为已解析类型。如果S是objectref引用的对象类型，T是解析的类、数组或接口类型，则checkcast确定objectref是否可以按如下方法转换为类型T: • 如果S是一个类类型，那么: – 如果T是一个类类型，那么S必须是与T相同的类，或者S必须是T的子类; – 如果T是一个接口类型，那么S必须实现接口T。 • 如果S是一个数组类型SC[]，即SC类型组件的数组，则: – 如果T是一个类类型，那么T必须是Object。 – 如果T是一种接口类型，那么T必须是数组实现的接口之一(JLS§4.10.3)。 – 如果T是数组类型TC[]，即由TC类型的组件组成的数组，则以下条件之一必须为true: ➢ TC和SC是相同的原生类型。 ➢ TC和SC是引用类型，类型SC可以通过这些规则的递归应用转换为TC。

链接异常	在解析类、数组或接口类型的符号引用时，可以抛出§5.4.3.1中记录的任何异常。
运行时异常	否则，如果objectref不能强制转换为解析的类、数组或接口类型，则checkcast指令会抛出ClassCastException。
注释	checkcast指令非常类似于instanceof指令(§instanceof)。它对null的处理、测试失败时的行为(checkcast抛出异常，instanceof推入结果代码)以及对操作数栈的影响都有所不同。

d2f

操作	转换 double 为 float
格式	d2f
形式	d2f = 144 (0x90)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须为double类型。它从操作数栈中弹出，并使用四舍五入策略转换为float结果(§2.8)。结果被推入操作数栈。一个太小而不能用float表示的有限值被转换为具有相同符号的零;一个太大而不能用float表示的有限值将被转换为具有相同符号的无穷大。double NaN被转换为float NaN。
注释	d2f指令执行收窄原生转换(JLS§5.1.3)。它可能会失去关于value的整体大小的信息，也可能会失去精确度。

d2i

操作	转换 double 为 int
格式	d2i
形式	d2i = 142 (0x8e)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须为double类型。它从操作数栈中弹出并转换为int结果。结果被推入操作数栈: • 如果值为NaN，则转换结果为int 0。 • 否则，如果该值不是无穷大，则会使用四舍五入策略(§2.8)将其舍入为整数值V。如果这个整数值V可以表示为int，那么结果就是int值V。 • 否则，要么该值太小(大幅的负值或负无穷大)，结果是int类型的最小可表示值，要么该值太大(大幅的正值或正无穷大)，结果是int类型的最大可表示值。
注释	d2i指令执行收窄原生转换(JLS§5.1.3)。它可能会失去关于value的整体大小的信息，也可能会失去精确度。

d2l

操作	转换 double 为 long
格式	d2l
形式	d2l = 143 (0x8f)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须为double类型。它从操作数栈中弹出并转换为long。结果被推入操作数栈: • 如果值为NaN，则转换结果为long 0。 • 否则，如果该值不是无穷大，则会使用向零舍入策略(§2.8)将其舍入为整数值V。如果这个整数值V可以表示为long，那么结果就是long值V。 • 否则，要么该值太小(大幅的负值或负无穷大)，结果是long类型的最小可表示值，要么该值太大(大幅的正值或正无穷大)，结果是long类型的最大可表示值。
注释	d2l指令执行收窄原生转换(JLS§5.1.3)。它可能会失去关于value的整体大小的信息，也可能会失去精确度。

dadd

操作	double 加法
格式	dadd
形式	dadd = 99 (0x63)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须为double类型。值从操作数栈弹出。double结果是value1 + value2。结果被推入操作数栈。 dadd指令的结果受IEEE 754算术规则的控制: · 如果value1或value2中的一个为NaN，则结果为NaN。 · 相反符号的两个无穷大之和为NaN。 · 同一符号的两个无穷大之和就是该符号的无穷大。 · 无穷大和任何有限值之和等于无穷大。 · 符号相反的两个零的和为正零。 · 同一符号的两个零的和就是该符号的零。 · 零和非零有限值之和等于非零值。 · 两个大小相同、符号相反的非零有限值之和为正零。 · 在其余情况下，如果操作数都不是无穷大、零或NaN，且值具有相同符号或不同大小，则使用四舍五入舍入策略计算和并舍入到最接近的可表示值（§2.8）。如果大小太大，不能用double表示，我们说操作溢出;结果就是一个带适当符号的无穷大。如果大小太小，不能用double表示，我们就说操作下溢；结果是一个带适当符号的零。 Java虚拟机需要支持逐步下溢。尽管可能会发生溢出、下溢或丢失精度的情况，但执行dadd指令绝不会引发运行时异常。

daload

操作	从数组加载 double
格式	daload
形式	daload = 49 (0x31)
操作数栈	..., arrayref, index→ ..., value
描述	arrayref必须是类型reference，并且必须引用一个组件类型为double的数组。index必须是int类型的。arrayref和index都是从操作数栈中弹出的。检索index处数组组件中的double值，并将其压入操作数栈。
运行时异常	如果arrayref为空，daload抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的范围内，daload指令将抛出ArrayIndexOutOfBoundsException异常。

dastore

操作	存储到 double 数组
格式	dastore
形式	dastore = 82 (0x52)
操作数栈	..., arrayref, index, value→ ...
描述	arrayref必须是类型reference，并且必须引用一个组件类型为double的数组。index必须是int类型，value必须是double类型。arrayref、index和value从操作数栈中弹出。double值存储为按index索引的数组的组件。
运行时异常	如果arrayref为空，则dastore抛出NullPointerException。 否则，如果index不在arrayref引用的数组的范围内，则dastore指令将抛出ArrayIndexOutOfBoundsException。

dcmp<op>

操作	比较 double
格式	dcmp<op>
形式	dcmpg = 152 (0x98) dcmpl = 151 (0x97)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须为double类型。从操作数栈中弹出值并执行浮点比较: • 如果value1大于value2，则将整型值1压入操作数栈。 • 否则，如果value1等于value2，则将int值0压入操作数栈。 • 否则，如果value1小于value2，则将int值-1压入操作数栈。 • 否则，value1或value2中至少有一个是NaN。dcmpg指令将int值1推入操作数栈，而dcmpl指令将int值-1推入操作数栈。 浮点比较是按照IEEE 754执行的。除NaN之外的所有值都是有序的，负无穷小于所有有限值，正无穷大于所有有限值。正零和负零被认为是相等的。
注释	dcmpg和dcmpl指令的区别仅仅在于它们对涉及NaN的比较的处理。NaN是无序的，因此如果它的一个或两个操作数都是NaN，则任何double比较都失败。在dcmpg和dcmpl都可用的情况下，无论对非NaN值的比较失败还是因为遇到NaN而失败，都可以编译任何double比较，将相同的结果推入操作数栈。更多信息请参见§3.5。

dconst_<d>

操作	压入 double
格式	dconst_<d>
形式	dconst_0 = 14 (0xe) dconst_1 = 15 (0xf)
操作数栈	...→ ..., <d>
描述	将double常量<d>（0.0或1.0）推送到操作数栈上。

ddiv

操作	double 除法
格式	ddiv
形式	ddiv = 111 (0x6f)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是double类型。值从操作数栈中弹出。 double结果是value1/value2。结果被推送到操作数栈上。 ddiv指令的结果由IEEE 754算术规则控制: • 如果value1或value2中的一个为NaN，则结果为NaN。 • 如果value1和value2都不是NaN，如果两个值具有相同的符号，则结果的符号为正，如果两个值具有不同的符号，则结果的符号为负。 • 用无穷大除以无穷大得到NaN。 • 用一个无穷大除以一个有限值得到一个有符号的无穷大，前面已经给出了产生符号的规则。 • 用有限值除以无穷大得到有符号的零，上面给出了产生符号的规则。 • 0除以0得到NaN;用零除以任何其他有限值得到一个有符号的零，上面给出了产生符号的规则。 • 用一个非零的有限值除以一个零，得到一个有符号的无穷大，前面给出了产生符号的规则。 • 在其余的情况下，如果操作数都不是无穷大、零或NaN，则使用四舍五入舍入策略(§2.8)计算商并舍入到最近的double。如果大小太大，不能用double表示，我们说操作溢出;结果就是一个带适当符号的无穷大。如果大小太小，不能用double表示，我们就说操作下溢;结果是一个带适当符号的零。 Java虚拟机需要支持逐步下溢。尽管可能发生溢出、下溢、除零或丢失精度的情况，但执行ddiv指令绝不会抛出运行时异常。

dload

操作	从局部变量加载 double
格式	dload index
形式	dload = 24 (0x18)
操作数栈	...→ ..., value
描述	index是无符号字节。index和index+1都必须是当前帧的局部变量数组的索引(§2.6)。在index处的局部变量必须包含double类型。index处的局部变量的值被推入操作数栈。
注释	dload操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

dload_<n>

操作	从局部变量加载 double
格式	dload_<n>
形式	dload_0 = 38 (0x26) dload_1 = 39 (0x27) dload_2 = 40 (0x28) dload_3 = 41 (0x29)
操作数栈	...→ ..., value
描述	<n>和<n>+1都必须是当前帧的局部变量数组的索引（§2.6）。<n>处的局部变量必须包含double。<n>处的局部变量的值被推送到操作数栈上。
注释	每个dload_ < n > 指令与带索引 < n > 的dload相同，只是操作数 < n > 是隐式的。

dmul

操作	double 乘法
格式	dmul
形式	dmul = 107 (0x6b)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是double类型。值从操作数栈中弹出。 double结果是value1*value2。结果被推送到操作数栈上。 dmul指令的结果由IEEE 754算术规则控制: • 如果value1或value2中的一个为NaN，则结果为NaN。 • 如果value1和value2都不是NaN，那么如果两个值具有相同的符号，则结果的符号为正;如果两个值具有不同的符号，则结果的符号为负。 • 无穷大乘以零得到NaN。 • 用一个无穷大乘以一个有限值会得到一个有符号的无穷大，上面给出了产生符号的规则。 • 在其余的情况下，既不涉及无穷大也不涉及NaN，则使用四舍五入策略(§2.8)计算乘积并舍入到最接近的可表示值。如果大小太大，不能用double表示，我们说操作溢出;结果就是一个带适当符号的无穷大。如果大小太小，不能用double表示，我们就说操作下溢;结果是一个带适当符号的零。 Java虚拟机需要支持逐步下溢。尽管可能会发生溢出、下溢或丢失精度的情况，但执行dmul指令绝不会抛出运行时异常。

dneg

操作	double 取反
格式	dneg
形式	dneg = 119 (0x77)
操作数栈	..., value→ ..., result
描述	该值必须为double类型。它从操作数栈弹出。double结果是value的算术取反。结果被推入操作数栈。 对于double值，取反与从零减不一样。如果x是+0.0，那么0.0-x等于+0.0，而-x等于-0.0。一元减号只是把double的符号颠倒。 有关的特殊例子: • 如果操作数是NaN，则结果是NaN(记住NaN没有符号)。 Java虚拟机还没有采用2019年版IEEE 754标准中的更强要求，即对所有输入（包括NaN）的符号位进行反求。 • 如果操作数是无穷大，则结果是相反符号的无穷大。 • 如果操作数为零，则结果为符号相反的零。

drem

操作	double 取余
格式	drem
形式	drem = 115 (0x73)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是double类型。值从操作数栈中弹出。计算double结果并将其推送到操作数栈上。 由于Java虚拟机中舍入策略的选择，drem指令的结果与IEEE 754定义的余数运算的结果不同（§2.8）。IEEE 754余数运算通过舍入除法计算余数，而不是截断除法，因此其行为与通常的整数余数运算符的行为不同。相反，Java虚拟机将drem定义为以类似于整数余数指令irem和lrem的方式运行，并使用向零舍入策略进行隐含除法；这可以与C库函数fmod进行比较。 drem指令的结果由以下规则控制，这些规则与IEEE 754算法匹配，但隐含除法的计算方式除外： • 如果value1或value2中的一个为NaN，则结果为NaN。 • 如果value1和value2都不是NaN，则结果的符号等于被除数的符号。 • 如果被除数是无穷大或除数是零或两者都是，结果是NaN。 • 如果被除数是有限的，除数是无穷大的，结果等于被除数。 • 如果被除数是零，除数是有限的，结果等于被除数。 • 在其余情况下，如果操作数都不是无穷大、零或NaN，则由被除数value1和除数value2得到的浮点余数结果由数学关系result=value1-（value2*q）定义，其中q是一个整数，仅当value1/value2为负时为负，仅当value1/value2为正时为正，并且其大小尽可能大而不超过value1和value2的真实数学商的大小。 尽管可能会发生除零的情况，但drem指令的求值永远不会抛出运行时异常。不会发生溢出、下溢或精度损失。
注释	IEEE 754余数运算可以由库例程Math.IEEEremainder或StrictMath.IEEEremainder来计算。

dreturn

操作	从方法返回 double
格式	dreturn
形式	dreturn = 175 (0xaf)
操作数栈	..., value→ [empty]
描述	当前方法必须有返回类型double。该值必须为double类型。如果当前方法是一个同步方法，在调用该方法时进入或重新进入的监视器将被更新并可能退出，就像在当前线程中执行一个monitorexit指令(§monitorexit)一样。如果没有抛出异常，则从当前帧的操作数栈中弹出值(§2.6)，并推入调用者帧的操作数栈中。 当前方法的操作数栈上的任何其他值都将被丢弃。 解释器然后将控制返回给方法的调用者，恢复调用者的框架。
运行时异常	如果Java虚拟机实现没有强制执行§2.11.10中描述的结构化锁定规则，那么如果当前方法是一个同步方法，并且当前线程不是调用该方法时进入或重新进入的监视器的所有者，则dreturn抛出IllegalMonitorStateException异常。例如，如果同步方法在其同步的对象上包含monitorexit指令，但没有monitorenter指令，就会发生这种情况。 否则，如果Java虚拟机实现强制执行§2.11.10中描述的结构化锁定规则，并且如果在调用当前方法期间违反了这些规则中的第一条，则dreturn抛出一个IllegalMonitorStateException。

dstore

操作	存储 double 到局部变量
格式	dstore index
形式	dstore = 57 (0x39)
操作数栈	..., value→ ...
描述	index是一个无符号字节。index和index+1都必须是当前帧的局部变量数组的索引(§2.6)。操作数栈顶部的值必须为double类型。它从操作数栈弹出。index和index+1处的局部变量被设置为value。
注释	dstore操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

dstore_<n>

操作	存储 double 到局部变量
格式	dstore_<n>
形式	dstore_0 = 71 (0x47) dstore_1 = 72 (0x48) dstore_2 = 73 (0x49) dstore_3 = 74 (0x4a)
操作数栈	..., value→ ...
描述	<n>和<n>+1都必须是当前帧的局部变量数组的索引（§2.6）。操作数栈顶部的值必须为double类型。它从操作数栈中弹出。<n>和<n>+1处的局部变量设置为value。
注释	每个dstore_<n>指令与带索引<n>的dstore相同，只是操作数<n>是隐式的。

dsub

操作	double 减法
格式	dsub
形式	dsub = 103 (0x67)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须为double类型。值从操作数栈弹出。double结果是value1 - value2。结果被推入操作数栈。 对于double减法，a-b总是产生与a+(-b)相同的结果。然而，对于dsub指令，从零减去与求反不同，因为如果x为+0.0，则0.0-x等于+0.0，但-x等于-0.0。 Java虚拟机需要支持逐渐下溢。尽管可能发生溢出、下溢或精度损失，但执行dsub指令不会引发运行时异常。

dup

操作	复制顶部操作数栈值
格式	dup
形式	dup = 89 (0x59)
操作数栈	..., value→ ..., value, value
描述	复制操作数栈顶部的值，并将复制的值推送到操作数栈。 dup指令不能使用，除非value是种类1计算类型的值(§2.11.1)。

dup_x1

操作	复制顶部操作数栈值并向下插入两个值
格式	dup_x1
形式	dup_x1 = 90 (0x5a)
操作数栈	..., value2, value1→ ..., value1, value2, value1
描述	在操作数栈上复制顶部的值，并将复制的值插入到操作数栈下面的两个值。 除非value1和value2都是种类1计算类型的值(§2.11.1)，否则不能使用dup_x1指令。

dup_x2

操作	复制顶部操作数栈的值并向下插入两个或三个值
格式	dup_x2
形式	dup_x2 = 91 (0x5b)
操作数栈	形式 1: ..., value3, value2, value1→ ..., value1, value3, value2, value1 其中 value1、value2 和 value3 都是种类 1 计算类型的值(§2.11.1)。 形式 2: ..., value2, value1→ ..., value1, value2, value1 其中 value1 是种类 1 计算类型的值，value2 是种类 2 计算类型的值(§2.11.1)。
描述	在操作数栈上复制顶部的值，并将复制的值插入到操作数栈下面的两三个值中。

dup2

操作	复制最上面的一个或两个操作数栈的值
格式	dup2
形式	dup2 = 92 (0x5c)
操作数栈	形式 1: ..., value2, value1→ ..., value2, value1, value2, value1 其中 value1 和 value2 都是种类 1 计算类型的值(§2.11.1)。 形式 2: ..., value→ ..., value, value 其中 value 是种类 *2* 计算类型的值 (§2.11.1)。
描述	复制操作数栈上最上面的一个或两个值，并按原始顺序将复制的值推回到操作数栈上。

dup2_x1

操作	复制最上面的一个或两个操作数堆栈值，并向下插入两个或三个值
格式	dup2_x1
形式	dup2_x1 = 93 (0x5d)
操作数栈	形式 1: ..., value3, value2, value1→ ..., value2, value1, value3, value2, value1 其中 value1、value2 和 value3 都是种类 1 计算类型的值(§2.11.1)。 形式 2: ..., value2, value1→ ..., value1, value2, value1 其中 value1 是种类 2 计算类型的值，value2 是种类 1 计算类型的值(§2.11.1)。
描述	复制操作数栈上最上面的一到两个值，并按原来的顺序插入复制的值，在操作数栈中原始值的下面插入一个值。

dup2_x2

操作	复制最上面的一个或两个操作数堆栈值，并向下插入两个、三个或四个值
格式	dup2_x2
形式	dup2_x2 = 94 (0x5e)
操作数栈	形式 1: ..., value4, value3, value2, value1→ ..., value2, value1, value4, value3, value2, value1 其中 value1、value2、value3 和 value4 都是种类 1 计算类型的值(§2.11.1)。 形式 2: ..., value3, value2, value1→ ..., value1, value3, value2, value1 其中 value1 是种类 2 计算类型的值，value2 和 value3 是种类 1 计算类型的值(§2.11.1)。 形式 3: ..., value3, value2, value1→ ..., value2, value1, value3, value2, value1 其中value1和value2都是种类1计算类型的值，value3是种类2计算类型的值(§2.11.1)。 形式4: ..., value2, value1→ ..., value1, value2, value1 其中value1和value2都是种类2计算类型的值(§2.11.1)。
描述	复制操作数栈上最上面的一个或两个值，并将复制的值按原始顺序插入到操作数栈中。

f2d

操作	将float转换为double
格式	f2d
形式	f2d = 141 (0x8d)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须为float类型。它从操作数栈中弹出并转换为double结果。结果被推入操作数栈。
注释	f2d指令执行扩宽原生转换(JLS§5.1.2)。

f2i

操作	将float转换为int
格式	f2i
形式	f2i = 139 (0x8b)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须为float类型。它从操作数栈中弹出并转换为int结果。结果被推入操作数栈: • 如果值为NaN，则转换结果为int 0。 • 否则，如果该值不是无穷大，则会使用向零舍入策略(§2.8)将其舍入为整数值V。如果这个整数值V可以表示为int，那么结果就是int值V。 • 否则，要么该值太小(大幅的负值或负无穷大)，结果是int类型的最小可表示值，要么该值太大(大幅的正值或正无穷大)，结果是int类型的最大可表示值。
注释	f2i指令执行收窄原生转换(JLS§5.1.3)。它可能会失去关于value的整体大小的信息，也可能会失去精确度。

f2l

操作	将float转换为long
格式	f2l
形式	f2l = 140 (0x8c)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须为float类型。它从操作数栈中弹出并转换为long结果。结果被推入操作数栈: • 如果值为NaN，则转换结果为long 0。 • 否则，如果该值不是无穷大，则会使用向零舍入策略(§2.8)将其舍入为整数值V。如果这个整数值V可以表示为long，那么结果就是long值V。 • 否则，要么该值太小(大幅的负值或负无穷大)，结果是long类型的最小可表示值，要么该值太大(大幅的正值或正无穷大)，结果是long类型的最大可表示值。
注释	f2l指令执行收窄原生转换(JLS§5.1.3)。它可能会失去关于value的整体大小的信息，也可能会失去精确度。

fadd

操作	float 加法
格式	fadd
形式	fadd = 98 (0x62)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须为float类型。值从操作数栈弹出。float结果是value1 + value2。结果被推入操作数栈。 fadd指令的结果受IEEE 754算术规则的控制: · 如果value1或value2中的一个为NaN，则结果为NaN。 · 相反符号的两个无穷大之和为NaN。 · 同一符号的两个无穷大之和就是该符号的无穷大。 · 无穷大和任何有限值之和等于无穷大。 · 符号相反的两个零的和为正零。 · 同一符号的两个零的和就是该符号的零。 · 零和非零有限值之和等于非零值。 · 两个大小相同、符号相反的非零有限值之和为正零。 · 在其余情况下，如果操作数都不是无穷大、零或NaN，且值具有相同符号或不同大小，则使用四舍五入舍入策略计算和并舍入到最接近的可表示值（§2.8）。如果大小太大，不能用float表示，我们说操作溢出;结果就是一个带适当符号的无穷大。如果大小太小，不能用float表示，我们就说操作下溢；结果是一个带适当符号的零。 Java虚拟机需要支持逐步下溢。尽管可能会发生溢出、下溢或丢失精度的情况，但执行fadd指令绝不会引发运行时异常。

faload

操作	从数组加载 float
格式	faload
形式	faload = 48 (0x30)
操作数栈	..., arrayref, index→ ...,value
描述	arrayref必须是类型reference，并且必须引用其组件类型为float的数组。index必须是int类型的。arrayref和index都是从操作数栈中弹出的。检索index处数组组件中的float值，并将其压入操作数栈。
运行时异常	如果arrayref为空，则faload抛出一个NullPointerException。 否则，如果index不在arrayref引用的数组的边界内，则faload指令抛出ArrayIndexOutOfBoundsException。

fastore

操作	存储到 float 数组
格式	fastore
形式	fastore = 81 (0x51)
操作数栈	..., arrayref, index, value→ ...
描述	arrayref必须是类型reference，并且必须引用一个组件类型为float的数组。index必须是int类型，value必须是float类型。arrayref、index和value从操作数栈中弹出。float值存储为按index索引的数组的组件。
运行时异常	如果arrayref为空，则fastore抛出NullPointerException。 否则，如果index不在arrayref引用的数组的范围内，则fastore指令将抛出ArrayIndexOutOfBoundsException。

fcmp<op>

操作	比较 float
格式	fcmp<op>
形式	fcmpg = 150 (0x96) fcmpl = 149 (0x95)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须为float类型。从操作数栈中弹出值并执行浮点比较: • 如果value1大于value2，则将整型值1压入操作数栈。 • 否则，如果value1等于value2，则将int值0压入操作数栈。 • 否则，如果value1小于value2，则将int值-1压入操作数栈。 • 否则，value1或value2中至少有一个是NaN。fcmpg指令将int值1推入操作数栈，而fcmpl指令将int值-1推入操作数栈。 浮点比较是按照IEEE 754执行的。除NaN之外的所有值都是有序的，负无穷小于所有有限值，正无穷大于所有有限值。正零和负零被认为是相等的。
注释	fcmpg和fcmpl指令的区别仅仅在于它们对涉及NaN的比较的处理。NaN是无序的，因此如果它的一个或两个操作数都是NaN，则任何float比较都失败。在fcmpg和fcmpl都可用的情况下，无论对非NaN值的比较失败还是因为遇到NaN而失败，都可以编译任何float比较，将相同的结果推入操作数栈。更多信息请参见§3.5。

fconst_<f>

操作	压入 float
格式	fconst_<f>
形式	fconst_0 = 11 (0xb) fconst_1 = 12 (0xc) fconst_2 = 13 (0xd)
操作数栈	...→ ..., <f>
描述	将float常量<f>（0.0,1.0或2.0）推送到操作数栈上。

fdiv

操作	float 除法
格式	fdiv
形式	fdiv = 110 (0x6e)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是float类型。值从操作数栈中弹出。float结果是value1/value2。结果被推送到操作数栈上。 fdiv指令的结果由IEEE 754算术规则控制: • 如果value1或value2中的一个为NaN，则结果为NaN。 • 如果value1和value2都不是NaN，如果两个值具有相同的符号，则结果的符号为正，如果两个值具有不同的符号，则结果的符号为负。 • 用无穷大除以无穷大得到NaN。 • 用一个无穷大除以一个有限值得到一个有符号的无穷大，前面已经给出了产生符号的规则。 • 用有限值除以无穷大得到有符号的零，上面给出了产生符号的规则。 • 0除以0得到NaN;用零除以任何其他有限值得到一个有符号的零，上面给出了产生符号的规则。 • 用一个非零的有限值除以一个零，得到一个有符号的无穷大，前面给出了产生符号的规则。 • 在其余的情况下，如果操作数都不是无穷大、零或NaN，则使用四舍五入舍入策略(§2.8)计算商并舍入到最近的float。如果大小太大，不能用float表示，我们说操作溢出;结果就是一个带适当符号的无穷大。如果大小太小，不能用float表示，我们就说操作下溢;结果是一个带适当符号的零。 Java虚拟机需要支持逐步下溢。尽管可能发生溢出、下溢、除零或丢失精度的情况，但执行fdiv指令绝不会抛出运行时异常。

fload

操作	从局部变量加载 float
格式	fload index
形式	fload = 23 (0x17)
操作数栈	...→ ..., value
描述	index是一个无符号字节，必须是当前帧的局部变量数组的索引(§2.6)。index处的局部变量必须包含一个float。index处的局部变量的值被推入操作数栈。
注释	fload操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

fload<n>

操作	从局部变量加载 float
格式	fload_<n>
形式	fload_0 = 34 (0x22) fload_1 = 35 (0x23) fload_2 = 36 (0x24) fload_3 = 37 (0x25)
操作数栈	...→ ..., value
描述	<n>必须是当前帧的局部变量数组的索引（§2.6）。<n>处的局部变量必须包含float。<n>处的局部变量的值被推送到操作数栈上。
注释	每个fload_ < n > 指令与带索引 < n > 的fload相同，只是操作数 < n > 是隐式的。

fmul

操作	float 乘法
格式	fmul
形式	fmul = 106 (0x6a)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是float类型。值从操作数栈中弹出。float结果是value1*value2。结果被推送到操作数栈上。 fmul指令的结果由IEEE 754算术规则控制: • 如果value1或value2中的一个为NaN，则结果为NaN。 • 如果value1和value2都不是NaN，那么如果两个值具有相同的符号，则结果的符号为正;如果两个值具有不同的符号，则结果的符号为负。 • 无穷大乘以零得到NaN。 • 用一个无穷大乘以一个有限值会得到一个有符号的无穷大，上面给出了产生符号的规则。 • 在其余的情况下，既不涉及无穷大也不涉及NaN，则使用四舍五入策略(§2.8)计算乘积并舍入到最接近的可表示值。如果大小太大，不能用float表示，我们说操作溢出;结果就是一个带适当符号的无穷大。如果大小太小，不能用float表示，我们就说操作下溢;结果是一个带适当符号的零。 Java虚拟机需要支持逐步下溢。尽管可能会发生溢出、下溢或丢失精度的情况，但执行fmul指令绝不会抛出运行时异常。

fneg

操作	double 取反
格式	fneg
形式	fneg = 118 (0x76)
操作数栈	..., value→ ..., result
描述	该值必须为float类型。它从操作数栈弹出。float结果是value的算术取反。结果被推入操作数栈。 对于float值，取反与从零减不一样。如果x是+0.0，那么0.0-x等于+0.0，而-x等于-0.0。一元减号只是把float的符号颠倒。 有关的特殊例子: • 如果操作数是NaN，则结果是NaN(记住NaN没有符号)。Java虚拟机还没有采用2019年版IEEE 754标准中的更强要求，即对所有输入（包括NaN）的符号位进行反求。 • 如果操作数是无穷大，则结果是相反符号的无穷大。 • 如果操作数为零，则结果为符号相反的零。

frem

操作	float 取余
格式	frem
形式	frem = 114 (0x72)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是float类型。值从操作数栈中弹出。计算float结果并将其推送到操作数栈上。 由于Java虚拟机中舍入策略的选择，frem指令的结果与IEEE 754定义的余数运算的结果不同（§2.8）。IEEE 754余数运算通过舍入除法计算余数，而不是截断除法，因此其行为与通常的整数余数运算符的行为不同。相反，Java虚拟机将frem定义为以类似于整数余数指令irem和lrem的方式运行，并使用向零舍入策略进行隐含除法；这可以与C库函数fmod进行比较。 frem指令的结果由以下规则控制，这些规则与IEEE 754算法匹配，但隐含除法的计算方式除外： • 如果value1或value2中的一个为NaN，则结果为NaN。 • 如果value1和value2都不是NaN，则结果的符号等于被除数的符号。 • 如果被除数是无穷大或除数是零或两者都是，结果是NaN。 • 如果被除数是有限的，除数是无穷大的，结果等于被除数。 • 如果被除数是零，除数是有限的，结果等于被除数。 • 在其余情况下，如果操作数都不是无穷大、零或NaN，则由被除数value1和除数value2得到的浮点余数结果由数学关系result=value1-（value2*q）定义，其中q是一个整数，仅当value1/value2为负时为负，仅当value1/value2为正时为正，并且其大小尽可能大而不超过value1和value2的真实数学商的大小。 尽管可能会发生除零的情况，但frem指令的求值永远不会抛出运行时异常。不会发生溢出、下溢或精度损失。
注释	IEEE 754余数运算可以由库例程Math.IEEEremainder或StrictMath.IEEEremainder来计算。

freturn

操作	从方法返回 float
格式	freturn
形式	*freturn* = 174 (0xae)
操作数栈	..., value→ [empty]
描述	当前方法必须有返回类型float。该值必须为float类型。如果当前方法是一个同步方法，在调用该方法时进入或重新进入的监视器将被更新并可能退出，就像在当前线程中执行一个monitorexit指令(§monitorexit)一样。如果没有抛出异常，则从当前帧的操作数栈中弹出值(§2.6)，并推入调用者帧的操作数栈中。当前方法的操作数栈上的任何其他值都将被丢弃。 解释器然后将控制返回给方法的调用者，恢复调用者的框架。
运行时异常	如果Java虚拟机实现没有强制执行§2.11.10中描述的结构化锁定规则，那么如果当前方法是一个同步方法，并且当前线程不是调用该方法时进入或重新进入的监视器的所有者，则freturn抛出IllegalMonitorStateException异常。例如，如果同步方法在其同步的对象上包含monitorexit指令，但没有monitorenter指令，就会发生这种情况。 否则，如果Java虚拟机实现强制执行§2.11.10中描述的结构化锁定规则，并且如果在调用当前方法期间违反了这些规则中的第一条，则freturn抛出一个IllegalMonitorStateException。

fstore

操作	存储 float 到局部变量
格式	fstore index
形式	fstore = 56 (0x38)
操作数栈	..., value→ ...
描述	index是一个无符号字节，必须是当前帧局部变量数组的索引（§2.6）。操作数栈顶部的值必须为float类型。它从操作数栈中弹出，index处的局部变量的值设置为value。
注释	fstore操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

fstore_<n>

操作	存储 float 到局部变量
格式	fstore_<n>
形式	fstore_0 = 67 (0x43) fstore_1 = 68 (0x44) fstore_2 = 69 (0x45) fstore_3 = 70 (0x46)
操作数栈	..., value→ ...
描述	<n>必须是当前帧的局部变量数组的索引（§2.6）。操作数栈顶部的值必须为float类型。它从操作数栈中弹出，并且 < n > 处的局部变量的值设置为value。
注释	每个fstore_<n>指令与带索引<n>的fstore相同，只是操作数<n>是隐式的。

fsub

操作	float 减法
格式	fsub
形式	fsub = 102 (0x66)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须为float类型。值从操作数栈弹出。float结果是value1 - value2。结果被推入操作数栈。 对于float减法，a-b总是产生与a+(-b)相同的结果。然而，对于fsub指令，从零减去与求反不同，因为如果x为+0.0，则0.0-x等于+0.0，但-x等于-0.0。 Java虚拟机需要支持逐渐下溢。尽管可能发生溢出、下溢或精度损失，但执行fsub指令不会引发运行时异常。

getfield

操作	从对象获取字段
格式	getfield indexbyte1 indexbyte2
形式	getfield = 180 (0xb4)
操作数栈	..., objectref→ ..., value
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是字段的符号引用(§5.1)，它给出了字段的名称和描述符，以及字段所在类的符号引用。引用字段被解析(§5.4.3.2)。 objectref必须是类型reference，但不是数组类型，它从操作数栈中弹出。objectref中引用字段的值被获取并推入操作数栈。
链接异常	在解析对字段的符号引用时，任何与字段解析相关的错误(§5.4.3.2)都可能被抛出。 否则，如果解析字段是静态字段，则getfield将抛出IncompatibleClassChangeError。
运行时异常	否则，如果objectref为空，则getfield指令将抛出NullPointerException。
注释	getfield指令不能用于访问数组的长度字段。使用arraylength指令（§arraylength）。

getstatic

操作	从类获取静态字段
格式	getstatic indexbyte1 indexbyte2
形式	getstatic = 178 (0xb2)
操作数栈	...,→ ..., value
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对字段的符号引用(§5.1)，它给出了字段的名称和描述符，以及对字段所在的类或接口的符号引用。引用字段被解析(§5.4.3.2)。 成功解析该字段后，如果类或接口尚未初始化，则声明该解析字段的类或接口将被初始化(§5.5)。 获取类或接口字段的值并将其压入操作数栈。
链接异常	在解析类或接口字段的符号引用时，可以抛出任何与字段解析相关的异常(§5.4.3.2)。 否则，如果解析的字段不是静态(类)字段或接口字段，getstatic抛出IncompatibleClassChangeError。
运行时异常	否则，如果执行这个getstatic指令导致被引用的类或接口初始化，getstatic可能抛出§5.5中详细描述的Error。

goto

操作	分支跳转
格式	goto branchbyte1 branchbyte2
形式	goto = 167 (0xa7)
操作数栈	没变化
描述	无符号字节branchbyte1和branchbyte2被用来构造一个带符号的16位的branchoffset，其中branchoffset为(branchbyte1 << 8) | branchbyte2。执行从该goto指令的操作码地址的偏移量开始。目标地址必须是包含该goto指令的方法中的指令的操作码的地址。

goto_w

操作	分支跳转(宽索引)
格式	goto branchbyte1 branchbyte2 branchbyte3 branchbyte4
形式	goto_w = 200 (0xc8)
操作数栈	没变化
描述	无符号字节branchbyte1、branchbyte2、branchbyte3和branchbyte4用于构造一个带符号的32位branchoffset，其中branchoffset为(branchbyte1 << 24) | (branchbyte2 << 16) | (branchbyte3 << 8) | branchbyte4。从这个goto_w指令的操作码地址的偏移量开始执行。目标地址必须是包含此goto_w指令的方法中指令的操作码的地址。
注释	尽管goto_w指令的分支偏移量为4字节，但其他因素会将方法的大小限制在65535字节(§4.11)。这个限制可能会在Java虚拟机的未来版本中提高。

i2b

操作	将int转换为byte
格式	i2b
形式	i2b = 145 (0x91)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是int类型的。它从操作数栈中弹出，被截断为byte，然后被符号扩展为int结果。结果被推入操作数栈。
注释	i2b指令执行收窄原生转换(JLS§5.1.3)。它可能会丢失关于value总体大小的信息。结果也可能没有与value相同的符号。

i2c

操作	将int转换为char
格式	i2c
形式	i2c = 146 (0x92)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是int类型的。它从操作数栈中弹出，被截断为char，然后被零扩展为int结果。结果被推入操作数栈。
注释	i2c指令执行收窄原生转换(JLS§5.1.3)。它可能会丢失关于value总体大小的信息。结果(它总是正的)也可能没有与value相同的符号。

i2d

操作	将int转换为double
格式	i2d
形式	i2d = 135 (0x87)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是int类型的。它从操作数栈中弹出，然后被转换为double结果。结果被推入操作数栈。
注释	i2d指令执行拓宽原生转换(JLS§5.1.2)。因为所有int类型的值都可以用double类型精确表示，所以转换是精确的。

i2f

操作	将int转换为float
格式	i2f
形式	i2f = 134 (0x86)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是int类型的。它从操作数栈中弹出，然后使用四舍五入策略(§2.8)被转换为float结果。结果被推入操作数栈。
注释	i2f指令执行拓宽原生转换(JLS§5.1.2)，但可能导致精度损失，因为float类型的值只有24位有效位。

i2l

操作	将int转换为long
格式	i2l
形式	i2l = 133 (0x85)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是int类型的。它从操作数栈中弹出并符号扩展为一个long结果。结果被推入操作数栈。
注释	i2l指令执行拓宽原生转换(JLS§5.1.2)。因为int类型的所有值都可以由long类型精确表示，所以转换是精确的。

i2s

操作	将int转换为short
格式	i2s
形式	i2s = 147 (0x93)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是int类型的。它从操作数堆栈中弹出，截断为short，然后符号扩展为int结果。结果被推入操作数栈。
注释	i2s指令执行收窄原生转换(JLS§5.1.3)。它可能会丢失关于value总体大小的信息。结果也可能没有与value相同的符号。

iadd

操作	int 加法
格式	iadd
形式	iadd = 96 (0x60)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。值从操作数栈弹出。int结果是value1 + value2。结果被推入操作数栈。 结果是真正数学结果的32个低阶位，采用足够宽的二进制补码格式，表示为int类型的值。如果发生溢出，则结果的符号可能与两个值的数学和的符号不相同。 尽管可能发生溢出，但执行iadd指令绝不会引发运行时异常。

iaload

操作	从数组加载 int
格式	iaload
形式	iaload = 46 (0x2e)
操作数栈	..., arrayref, index→ ..., value
描述	arrayref必须是类型reference，并且必须引用其组件类型为int的数组。index必须是int类型的。arrayref和index都是从操作数栈中弹出的。检索index处数组组件中的int值，并将其压入操作数栈。
运行时异常	如果arrayref为空，iaload抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的范围内，iaload指令将抛出ArrayIndexOutOfBoundsException异常。

iand

操作	int 按位与
格式	iand
形式	iand = 126 (0x7e)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。它们从操作数栈中弹出。通过对value1和value2进行按位与来计算int结果。结果被推入操作数栈。

iastore

操作	存储到 int 数组
格式	iastore
形式	iastore = 79 (0x4f)
操作数栈	..., arrayref, index, value→ ...
描述	arrayref必须是类型reference，并且必须引用其组件类型为int的数组。index和value都必须是int类型。arrayref、index和value从操作数栈中弹出。int值存储为按index索引的数组的组件。
运行时异常	如果arrayref为空，iastore抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的范围内，iastore指令将抛出ArrayIndexOutOfBoundsException异常。

iconst_<i>

操作	压入 int 常量
格式	iconst_<i>
形式	iconst_m1 = 2 (0x2) iconst_0 = 3 (0x3) iconst_1 = 4 (0x4) iconst_2 = 5 (0x5) iconst_3 = 6 (0x6) iconst_4 = 7 (0x7) iconst_5 = 8 (0x8)
操作数栈	...→ ..., <i>
描述	将int常量<i>（-1、0、1、2、3、4或5）推送到操作数栈上。
注释	除了操作数<i>是隐式的以外，这一系列指令中的每一个指令都相当于<i>的各自值的bipush<i>。

idiv

操作	int 除法
格式	idiv
形式	idiv = 108 (0x6c)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。值从操作数栈弹出。int结果是Java编程语言表达式value1 / value2 (JLS§15.17.2)的值。结果被推入操作数栈。 int除法舍入到0;也就是说，n/d中int值的商是一个int值q，其大小尽可能大，且满足|d·q|≤|n|。此外，当|n|≥|d|且n和d有相同的符号时，q为正，但当|n|≥|d|且n和d有相反的符号时，q为负。 有一种特殊情况不满足这个规则:如果被除数是int类型的最大可能大小的负整数，并且除数是-1，那么就会发生溢出，结果等于被除数。尽管存在溢出，但在这种情况下不会抛出异常。
运行时异常	如果int除法中的除数的值为0,idiv抛出ArithmeticException。

if_acmp<cond>

操作	如果引用比较成功，则跳转分支
格式	if_acmp<cond> branchbyte1 branchbyte2
形式	if_acmpeq = 165 (0xa5) if_acmpne = 166 (0xa6)
操作数栈	..., value1, value2→ ...
描述	value1和value2都必须是类型reference。它们都从操作数栈中弹出并进行比较。对比结果如下: • if_acmpeq 成功当且仅当value1 = value2 • if_acmpne 成功当且仅当value1≠value2 如果比较成功，则使用无符号的branchbyte1和branchbyte2构造有符号的16位偏移量，其中偏移量计算为(branchbyte1 << 8) | branchbyte2。然后，从if_acmp<cond>指令的操作码地址的偏移量处继续执行。目标地址必须是方法中包含此if_acmp<cond>指令的指令操作码的地址。 否则，如果比较失败，则在if_acmp<cond>指令后面的指令地址继续执行。

if_icmp<cond>

操作	如果 int 比较成功，则跳转分支
格式	if_icmp<cond> branchbyte1 branchbyte2
形式	if_icmpeq = 159 (0x9f) if_icmpne = 160 (0xa0) if_icmplt = 161 (0xa1) if_icmpge = 162 (0xa2) if_icmpgt = 163 (0xa3) if_icmple = 164 (0xa4)
操作数栈	..., value1, value2→ ...
描述	value1和value2都必须是int类型。它们都从操作数栈中弹出并进行比较。所有比较都是有符号的。比较结果如下: • if_icmpeq 成功当且仅当value1 = value2 • if_icmpne 成功当且仅当value1 ≠value2 • if_icmplt 成功当且仅当value1 < value2 • if_icmple 成功当且仅当value1 ≤value2 • if_icmpgt 成功当且仅当value1 > value2 • if_icmpge 成功当且仅当value1 ≥value2 如果比较成功，则使用无符号branchbyte1和branchbyte 2构造有符号16位偏移量，其中偏移量计算为（branchbytes1<<8）|branchbytec2。然后，执行在该if_icmp < cond > 指令的操作码地址的偏移处继续。目标地址必须是包含此if_icmp<cond>指令的方法中指令的操作码的地址。 否则，在if_icmp<cond>指令后面的指令地址继续执行。

if<cond>

操作	如果 int 与 0 比较成功，则跳转分支
格式	if_icmp<cond> branchbyte1 branchbyte2
形式	ifeq = 153 (0x99) ifne = 154 (0x9a) iflt = 155 (0x9b) ifge = 156 (0x9c) ifgt = 157 (0x9d) ifle = 158 (0x9e)
操作数栈	..., value→ ...
描述	value必须为int类型。它从操作数栈中弹出并与0进行比较。所有比较都是有符号的。比较结果如下: • ifeq 成功当且仅当value = 0 • ifne 成功当且仅当value≠0 • iflt 成功当且仅当value < 0 • ifle 成功当且仅当lue ≤ 0 • ifgt 成功当且仅当value > 0 • ifge 成功当且仅当value ≥0 如果比较成功，则使用无符号branchbyte1和branchbyte 2构造有符号16位偏移量，其中偏移量计算为（branchbytes1<<8）|branchbytec2。然后，执行在该if < cond > 指令的操作码地址的偏移处继续。目标地址必须是包含此if < cond > 指令的方法中指令的操作码的地址。 否则，执行将在该if<cond>指令之后的指令地址继续。

ifnonnull

操作	如果引用非空，则跳转分支
格式	ifnonnull branchbyte1 branchbyte2
形式	ifnonnull = 199 (0xc7)
操作数栈	..., value→ ...
描述	value必须为类型reference。它从操作数栈弹出。如果value不为null，则使用无符号的branchbyte1和branchbyte2构造有符号的16位偏移量，其中偏移量计算为(branchbyte1 << 8) | branchbyte2。然后，从这个ifnonnull指令的操作码地址的偏移量开始执行。目标地址必须是包含此ifnonnull指令的方法中指令的操作码的地址。 否则，在这个ifnonnull指令后面的指令地址继续执行。

ifnull

操作	如果引用为空，则跳转分支
格式	ifnull branchbyte1 branchbyte2
形式	ifnull = 198 (0xc6)
操作数栈	..., value→ ...
描述	value必须为类型reference。它从操作数栈弹出。如果value为null，则使用无符号的branchbyte1和branchbyte2构造有符号的16位偏移量，其中偏移量计算为(branchbyte1 << 8) | branchbyte2。然后，从这个ifnull指令的操作码地址的偏移量处继续执行。目标地址必须是包含该ifnull指令的方法中指令的操作码的地址。 否则，在ifnull指令后面的指令地址继续执行。

iinc

操作	局部变量加一个常数
格式	iinc index const
形式	iinc = 132 (0x84)
操作数栈	没有变化
描述	index是一个无符号字节，必须是当前帧的局部变量数组的索引(§2.6)。const是直接带符号字节。在index处的局部变量必须包含一个int值。值const首先被符号扩展为int型，值const首先符号扩展为int，然后index处的局部变量递增该值。
注释	iinc操作码可以与wide指令(§wide)一起使用，使用一个双字节无符号索引访问一个局部变量，并增加一个双字节直接有符号值。

iload

操作	从局部变量加载 int
格式	iload index
形式	iload = 21 (0x15)
操作数栈	...→ ..., value
描述	index是一个无符号字节，必须是当前帧的局部变量数组的索引(§2.6)。在index处的局部变量必须包含一个int值。index处的局部变量的值被推入操作数栈。
注释	iIload操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

iload_<n>

操作	从局部变量加载 int
格式	iload_<n>
形式	iload_0 = 26 (0x1a) iload_1 = 27 (0x1b) iload_2 = 28 (0x1c) iload_3 = 29 (0x1d)
操作数栈	...→ ..., value
描述	<n>必须是当前帧的局部变量数组的索引（§2.6）。<n>处的局部变量必须包含int。<n>处的局部变量的值被推送到操作数栈上。
注释	每个iload_ < n > 指令与带索引 < n > 的iload相同，只是操作数 < n > 是隐式的。

imul

操作	int 乘法
格式	imul
形式	imul = 104 (0x68)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。值从操作数栈弹出。int的结果是value1 * value2。结果被推入操作数栈。 结果是真正数学结果的32个低阶位，采用足够宽的二进制补码格式，表示为int类型的值。如果发生溢出，则结果的符号可能与两个值的数学乘法的符号不相同。 尽管可能发生溢出，但执行imul指令永远不会抛出运行时异常。

ineg

操作	int 取反
格式	ineg
形式	ineg = 116 (0x74)
操作数栈	..., value→ ..., result
描述	value必须为int类型。它从操作数栈弹出。int结果是value,-value的算术取反。结果被推入操作数栈。 对于int值，负数与从零减法相同。因为Java虚拟机对整数使用二进制补码表示，而且二进制补码值范围不是对称的，所以最大的负整数取反会得到相同的最大负数。尽管发生了溢出，但不会引发异常。 对所有int值x, -x 等于(~x)+1。

instanceof

操作	确定对象是否具有给定的类型
格式	instanceof indexbyte1 indexbyte2
形式	instanceof = 193 (0xc1)
操作数栈	..., objectref→ ..., result
描述	必须是类型reference的objectref从操作数堆栈中弹出。无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对类、数组或接口类型的符号引用。 如果objectref为空，则instanceof指令将int类型的结果0作为int类型压入操作数栈。 否则，已命名的类、数组或接口类型将被解析(§5.4.3.1)。如果objectref是解析类或数组类型的实例，或实现解析接口，则instanceof指令将int类型的结果1作为int类型压入操作数栈;否则，它推入一个int型结果0。 以下规则用于确定非null的objectref是否是解析类型的实例。如果S是objectref引用的对象的类型，T是解析的类、数组或接口类型，则instanceof确定objectref是否为T的实例，如下所示: • 如果S是一个类类型，那么: – 如果T是一个类类型，那么S必须是与T相同的类，或者S必须是T的子类; – 如果T是一个接口类型，那么S必须实现接口T。 • 如果S是一个数组类型SC[]，即SC类型组件的数组，则: – 如果T是一个类类型，那么T必须是Object。 – 如果T是一种接口类型，那么T必须是数组实现的接口之一(JLS§4.10.3)。 – 如果T是数组类型TC[]，即由TC类型的组件组成的数组，则以下条件之一必须为true: ➢ TC和SC是相同的原生类型。

	➢ TC和SC是引用类型，类型SC可以通过这些运行时规则转换为TC。
链接异常	在解析类、数组或接口类型的符号引用时，可以抛出§5.4.3.1中记录的任何异常。
注释	instanceof指令与checkcast指令非常相似（§checkcast）。它的不同之处在于对null的处理、测试失败时的行为（checkcast抛出异常、instanceof推送结果代码）以及对操作数栈的影响。

invokedynamic

操作	调用动态计算的调用站点
格式	invokedynamic indexbyte1 indexbyte2 0 0
形式	invokedynamic = 186 (0xba)
操作数栈	..., [arg1, [arg2 ...]]→ ...
描述	首先，无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是动态计算调用站点的符号引用(§5.1)。第三和第四个操作数字节的值必须始终为零。 这个特定的invokedynamic指令的符号引用被解析(§5.4.3.6)，以获得对java.lang.invoke.CallSite实例的引用。java.lang.invoke.CallSite的实例被认为“绑定”到这个特定的 invokedynamic指令。 java.lang.invoke.CallSite的实例指示目标方法句柄。nargs参数值从操作数栈中弹出，并调用目标方法句柄。调用就像通过执行一个invokevirtual指令一样发生，该指令指示符号引用R的运行时常量池索引，其中: • R是对类的方法的符号引用; • 对于要在其中找到方法的类的符号引用，R指定java.lang.invoke.MethodHandle; • 对于方法名, R指定invokeExact; • 对于方法描述符, R指定动态计算调用站点中的方法描述符。 在这里，就好像下列项按顺序被压入到操作数栈中: • 对目标方法句柄的引用; • nargs参数值，其中值的数量、类型和顺序必须与动态计算调用站点中的方法描述符一致。
链接异常	在解析对动态计算调用站点的符号引用期间，可以抛出与动态计

	算调用站点解析相关的任何异常。
注释	如果可以解析对动态计算的调用站点的符号引用，则意味着对java.lang.invoke.CallSite实例的非空引用被绑定到invokedynamic指令。因此，目标方法句柄，由java.lang.invoke.CallSite的实例指示，是非空的。 类似地，成功解析意味着符号引用中的方法描述符在语义上等于目标方法句柄的类型描述符。 这些不变量意味着绑定到java.lang.invoke.CallSite实例的invokedynamic指令永远不会抛出NullPointerException或java.lang.invoke.WrongMethodTypeException。

invokeinterface

操作	调用接口方法
格式	invokeinterface indexbyte1 indexbyte2 count 0
形式	invokeinterface = 185 (0xb9)
操作数栈	..., objectref, [arg1, [arg2 ...]]→ ...
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。 索引处的运行时常量池条目必须是对接口方法的符号引用(§5.1)，它给出了接口方法的名称和描述符(§4.3.3)，以及对接口方法所在接口的符号引用。命名接口方法被解析(§5.4.3.4)。 解析的接口方法不能是实例初始化方法，也不能是类或接口初始化方法(§2.9.1，§2.9.2)。 count操作数是一个无符号字节，不能为零。objectref必须具有类型reference，并且必须在操作数栈上紧跟nargs参数值，其中值的数量、类型和顺序必须与解析接口方法的描述符一致。第四个操作数字节的值必须始终为零。 设C是objectref的类。针对C和解析方法选择了一种方法(§5.4.6)。这是要调用的方法。 如果要调用的方法是同步的，那么就像在当前线程中执行一个monitorenter指令(§monitorenter)一样，进入或重新进入与objectref相关的监视器。 如果要调用的方法不是本地的，nargs参数值和objectref将从操作数栈中弹出。在Java虚拟机堆栈上为正在调用的方法创建一个新帧。objectref和参数值连续成为新帧的局部变量的值，objectref在局部变量0中，arg1在局部变量1中（或者，如果arg1是long或double类型，则在局部变量2和1中），以此类推。然后使新的帧

	成为当前帧，并将Java虚拟机pc设置为要调用的方法的第一条指令的操作码。继续执行该方法的第一条指令。 如果要调用的方法是本地的，并且实现它的依赖平台的代码尚未绑定到Java虚拟机中（§5.6），那么就完成了。nargs参数值和objectref从操作数栈中弹出，并作为参数传递给实现该方法的代码。传递参数并以依赖于实现的方式调用代码。当平台相关代码返回时： • 如果本地方法已同步，则与objectref关联的监控器将更新，并且可能会像在当前线程中执行monitorexit指令（§monitorexit）一样退出。 • 如果本地方法返回值，则平台相关代码的返回值将以依赖于实现的方式转换为本地方法的返回类型，并推送到操作数栈上。
链接异常	在解析接口方法的符号引用时，可以抛出与接口方法解析有关的任何异常（§5.4.3.4）。 否则，如果解析的方法是静态的，invokeinterface指令将抛出IncompatibleClassChangeError。 注意，invokeinterface可能引用接口中声明的私有方法，包括嵌套接口。
运行时异常	否则，如果objectref为空，invokeinterface指令将抛出NullPointerException。 否则，如果objectref的类没有实现已解析的接口，invokeinterface将抛出一个IncompatibleClassChangeError。 否则，如果所选方法既不是公共的也不是私有的，invokeinterface将抛出一个IllegalAccessError。 否则，如果所选方法是抽象的，invokeinterface将抛出AbstractMethodError。 否则，如果所选方法是本地的，并且实现该方法的代码无法绑定，则invokeinterface将抛出一个UnsatisfiedLinkError。 否则，如果未选择任何方法，并且C中有多个最大特定的超接口方法与解析方法的名称和描述符匹配，并且不是抽象的，invokeinterface将抛出一个IncompatibleClassChangeError。 否则，如果未选择任何方法，并且C中没有与解析方法的名称和描述符匹配且不是抽象的最大特定超接口方法，invokeinterface将抛出AbstractMethodError。
注释	invokeinterface指令的count操作数记录参数值数量的度量，其中long或double类型的参数值为count值贡献两个单位，而任何其他类型的参数贡献一个单位。该信息也可以从所选方法的描述符中得出。冗余是历史性的。 第四个操作数字节用于为Oracle的某些Java虚拟机实现中使用的额

	外操作数保留空间，这些实现在运行时用专用伪指令替换invokeinterface指令。为了向后兼容，必须保留它。 nargs参数值和objectref与第一个nargs+1局部变量不是一对一的。long和double类型的参数值必须存储在两个连续的局部变量中，因此可能需要多个nargs局部变量才能将nargs参数值传递给调用的方法。 选择逻辑允许选择在超接口中声明的非抽象方法。仅当类层次结构中没有匹配方法时，才考虑接口中的方法。如果在超接口层次结构中有两个非抽象方法，且两个方法中任何一个都不比另一个更具体，则会发生错误；没有试图消除歧义（例如，一个可能是引用的方法，另一个可能不相关，但我们不会偏好于引用的方法）。另一方面，如果有许多抽象方法，但只有一个非抽象方法，则选择非抽象方法（除非抽象方法更具体）。

invokespecial

操作	调用实例方法;直接调用实例初始化方法和当前类及其超类型的方法
格式	invokespecial indexbyte1 indexbyte2
形式	invokespecial = 183 (0xb7)
操作数栈	..., objectref, [arg1, [arg2 ...]]→ ...
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对方法或接口方法的符号引用(§5.1)，它给出了方法或接口方法的名称和描述符(§4.3.3)，以及对方法或接口方法所在的类或接口的符号引用。命名的方法被解析(§5.4.3.3，§5.4.3.4)。 如果以下所有条件都成立，则设C为当前类的直接超类: • 解析的方法不是一个实例初始化方法(§2.9.1)。 • 符号引用命名一个类(而不是接口)，该类是当前类的超类。 • ACC_SUPER标志是为class文件设置的(§4.1)。 否则，设C为由符号引用命名的类或接口。 要调用的实际方法由以下查找过程选择: 1. 如果C包含一个实例方法的声明，其名称和描述符与解析方法相同，那么该方法就是要调用的方法。 2. 否则，如果C是一个类并具有一个超类，则执行搜索与解析方法具有相同名称和描述符的实例方法的声明，从C的直接超类开始，继续搜索该类的直接超类，以此类推，直到找到匹配或不再存在其他超类为止。如果找到匹配，那么它就是要调用的方法。 3. 否则，如果C是一个接口，并且类Object包含一个公共实例方法的声明，该方法具有与解析方法相同的名称和描述符，那么它就是要调用的方法。 4. 否则，如果在C的超接口中刚好有一个最大特定的方法(§5.4.3.3)匹配解析方法的名称和描述符，并且不是抽象的，那么它就是要调用的方法。

	objectref必须是类型reference，并且必须在操作数堆栈上后跟nargs参数值，其中值的数量、类型和顺序必须与所选实例方法的描述符一致。 如果该方法是同步的，则与objectref关联的监视器将被进入或重新进入，就像在当前线程中执行monitorenter指令（§monitorenter）一样。 如果该方法不是本地方法，则从操作数栈中弹出nargs参数值和objectref。在Java虚拟机栈上为正在调用的方法创建一个新帧。objectref和参数值连续成为新帧的局部变量的值，objectref在局部变量0中，arg1在局部变量1中（或者，如果arg1是long或double类型，则在局部变量2和1中），以此类推。然后使新的帧成为当前帧，并将Java虚拟机pc设置为要调用的方法的第一条指令的操作码。继续执行该方法的第一条指令。 如果该方法是本地的，并且实现它的依赖于平台的代码尚未绑定（§5.6）到Java虚拟机中，那么就完成了。nargs参数值和objectref从操作数栈中弹出，并作为参数传递给实现该方法的代码。传递参数并以依赖于实现的方式调用代码。当平台相关代码返回时，会发生以下情况： • 如果本地方法已同步，则与objectref关联的监控器将更新，并且可能会像在当前线程中执行monitorexit指令（§monitorexit）一样退出。 • 如果本地方法返回值，则平台相关代码的返回值将以依赖于实现的方式转换为本地方法的返回类型，并推送到操作数栈上。
链接异常	在方法符号引用的解析过程中，可以抛出与方法解析相关的任何异常（§5.4.3.3）。 否则，如果解析的方法是实例初始化方法，并且声明该方法的类不是指令符号引用的类，则会抛出NoSuchMethodError。 否则，如果解析的方法是类（静态）方法，invokespecial指令将抛出IncompatibleClassChangeError。
运行时异常	否则，如果objectref为空，invokespecial指令将抛出NullPointerException。 否则，如果查找过程的步骤1、步骤2或步骤3选择了抽象方法，invokespecial将抛出AbstractMethodError。 否则，如果查找过程的步骤1、步骤2或步骤3选择了本地方法，并且实现该方法的代码无法绑定，则invokespecial将抛出UnsatifiedLinkError。 否则，如果查找过程的步骤4确定存在多个与解析方法的名称和描述符匹配且不是抽象的最大特定的C的超接口方法，invokespecial

	将抛出IncompatibleClassChangeError, 否则，如果查找过程的步骤4确定不存在与解析方法的名称和描述符匹配且不是抽象的最大特定的C的超接口方法，invokespecial将抛出AbstractMethodError。
注释	invokespecial指令和invokevirtual指令（§invokevirtual）之间的区别在于invokevirtual调用基于对象类的方法。invokespecial指令用于直接调用实例初始化方法（§2.9.1）以及当前类及其超类的方法。 invokespecial指令在JDK 1.0.2版之前被命名为invokenonvirtual。 nargs参数值和objectref与第一个nargs+1局部变量不是一对一的。long和double类型的参数值必须存储在两个连续的局部变量中，因此可能需要多个nargs局部变量才能将nargs参数值传递给调用的方法。 invokespecial指令处理通过直接超接口或超类引用的非抽象接口方法的调用。在这些情况下，选择规则基本上与invokeinterface的规则相同（除了搜索从不同的类开始）。

invokestatic

操作	调用类（静态）方法
格式	invokestatic indexbyte1 indexbyte2
形式	invokestatic = 184 (0xb8)
操作数栈	..., [arg1, [arg2 ...]]→ ...
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对方法或接口方法的符号引用(§5.1)，它给出了方法或接口方法的名称和描述符(§4.3.3)，以及对方法或接口方法所在的类或接口的符号引用。命名的方法被解析(§5.4.3.3，§5.4.3.4)。 解析的方法不能是实例初始化方法，也不能是类或接口初始化方法(§2.9.1，§2.9.2)。 解析方法必须是静态的，因此不能是抽象的。 在成功解析方法时，如果类或接口尚未初始化，则声明被解析方法的类或接口将被初始化(§5.5)。 操作数栈必须包含nargs参数值，其中值的数量、类型和顺序必须与解析方法的描述符一致。 如果方法是同步的，与解析的Class对象相关联的监视器将被进入或重新进入，就像在当前线程中执行一个monitorenter指令(§monitorenter)一样。 如果方法不是本地的，nargs参数值将从操作数栈中弹出。在Java虚拟机堆栈上为正在调用的方法创建一个新帧。nargs参数值连续地成为新帧的局部变量值，其中arg1在局部变量0中（或者，如果arg1是long或double类型，则在局部变量0和1中），以此类推。然后使新的帧成为当前帧，并将Java虚拟机pc设置为要调用的方法的第一条指令的操作码。继续执行该方法的第一条指令。 如果该方法是本地的，并且实现它的依赖于平台的代码尚未绑定（§5.6）到Java虚拟机中，那么就完成了。nargs参数值从操作数栈中弹出，并作为参数传递给实现该方法的代码。传递参数并以

	依赖于实现的方式调用代码。当平台相关代码返回时，会发生以下情况： · 如果本地方法已同步，则与解析的Class对象关联的监视器将被更新，并可能通过在当前线程中执行monitorexit指令（§monitorexit）退出。 · 如果本地方法返回值，则平台相关代码的返回值将以依赖于实现的方式转换为本地方法的返回类型，并推送到操作数栈上。
链接异常	在方法符号引用的解析过程中，可以抛出与方法解析相关的任何异常（§5.4.3.3）。 否则，如果解析的方法是实例方法，invokestatic指令将抛出IncompatibleClassChangeError。
运行时异常	否则，如果该invokestatic指令的执行导致被引用类或接口的初始化，invokestatic可能会抛出Error，详见§5.5。 否则，如果解析的方法是本地的，并且实现该方法的代码无法绑定，invokestatic将抛出UnsatifiedLinkError。
注释	nargs参数值与第一个nargs局部变量不是一对一的。long和double类型的参数值必须存储在两个连续的局部变量中，因此可能需要多个nargs局部变量才能将nargs参数值传递给调用的方法。

invokevirtual

操作	调用实例方法；基于类的调度
格式	invokevirtual indexbyte1 indexbyte2
形式	invokevirtual = 182 (0xb6)
操作数栈	..., objectref, [arg1, [arg2 ...]]→ ...
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是一个方法的符号引用(§5.1)，它给出了方法的名称和描述符(§4.3.3)，以及一个方法所在类的符号引用。命名的方法被解析(§5.4.3.3)。 如果解析的方法不是签名多态的(§2.9.3)，则invokevirtual指令按照以下步骤进行。 设C是objectref的类。针对C和解析方法选择了一种方法(§5.4.6)。这是要调用的方法。 在操作数堆栈上，objectref后面必须跟着nargs参数值，其中值的数量、类型和顺序必须与所选实例方法的描述符一致。 如果要调用的方法是同步的，那么就像在当前线程中执行一个monitorenter指令(§monitorenter)一样，进入或重新进入与objectref相关的监视器。 如果要调用的方法不是本地的，nargs参数值和objectref将从操作数栈中弹出。在Java虚拟机栈上为正在调用的方法创建一个新帧。objectref和参数值连续成为新帧的局部变量的值，objectref在局部变量0中，arg1在局部变量1中（或者，如果arg1是long或double类型，则在局部变量1和2中），以此类推。然后使新的帧成为当前帧，并将Java虚拟机pc设置为要调用的方法的第一条指令的操作码。继续执行该方法的第一条指令。 如果要调用的方法是本地的，并且实现它的依赖于平台的代码尚未绑定（§5.6）到Java虚拟机中，那么就完成了。nargs参数值和objectref从操作数栈中弹出，并作为参数传递给实现该方法的代码。传递参数并以依赖于实现的方式调用代码。当平台相关代码返回时，会发生以

	下情况: • 如果本地方法已同步，则与objectref关联的监控器将更新，并且可能会像在当前线程中执行monitorexit指令（§monitorexit）一样退出。 • 如果本地方法返回值，则平台相关代码的返回值将以依赖于实现的方式转换为本地方法的返回类型，并推送到操作数栈上。 如果解析的方法是签名多态的（§2.9.3），并在java.lang.invoke.MethodHandle类中声明，则invokevirtual指令按如下方式进行，其中D是指令符号引用的方法的描述符。 首先，对java.lang.invoke.MethodType实例的引用是通过解析一个方法类型(§5.4.3.5)的符号引用获得的，该方法类型具有与D相同的参数和返回类型。 • 如果命名的方法是invokeExact, java.lang.invoke.MethodType的实例必须在语义上等于接收方法句柄objectref的类型描述符。要调用的方法句柄是objectref。 • 如果命名的方法是invoke，并且java.lang.invoke.MethodType的实例在语义上等于接收方法句柄objectref的类型描述符，那么要调用的方法句柄是objectref。 • 如果命名的方法是invoke，并且java.lang.invoke.MethodType的实例在语义上不等于接收方法句柄objectref的类型描述符，那么Java虚拟机尝试调整接收方法句柄的类型描述符，就像调用java.lang.invoke.MethodHandle的asType方法一样，以获得一个完全可调用的方法句柄m。要调用的方法句柄是m。 在操作数栈上，objectref后面必须跟着nargs参数值，其中值的数量、类型和顺序必须与要调用的方法句柄的类型描述符一致。(该类型描述符将对应于被调用的方法句柄类型的方法描述符，如§5.4.3.5所述。) 然后，如果要调用的方法句柄具有字节码行为，Java虚拟机调用方法句柄，就像执行与方法句柄的类型相关联的字节码行为一样。如果类型是5 (REF_invokeVirtual)， 6 (REF_invokeStatic)， 7 (REF_invokeSpecial)， 8 (REF_newInvokeSpecial)，或9 (REF_invokeInterface)，那么将在执行字节码行为的过程中创建一个帧并使其成为当前帧;然而，这个帧是不可见的，当字节码行为调用的方法完成(正常或突然)时，它的调用者的帧被认为是包含这个invokevirtual指令的方法的帧。 否则，如果要调用的方法句柄没有字节码行为，Java虚拟机将以依赖于实现的方式调用它。 如果解析的方法是签名多态的，并在java.lang.invoke.VarHandle类中声明，那么invokevirtual指令将按照以下步骤执行，其中N和D是指令象征性引用的方法的名称和描述符。

	首先，对java.lang.invoke.VarHandle.AccessMode实例的引用是通过调用java.lang.invoke.VarHandle.AccessMode的valueFromMethodName方法获得的，该方法带有一个表示N的String参数。 其次，对java.lang.invoke.MethodType实例的引用是通过调用实例objectref上的java.lang.invoke.VarHandle的accessModeType方法获得的，并以java.lang.invoke.VarHandle.AccessMode的实例作为参数。 第三，对java.lang.invoke.MethodHandle实例的引用是通过调用java.lang.invoke.MethodHandles的varHandleExactInvoker方法获得的，其中java.lang.invoke.VarHandle.AccessMode的实例作为第一个参数，java.lang.invoke.MethodType的实例作为第二个参数。得到的实例称为调用程序方法句柄。得到的实例称为调用程序方法句柄。 最后，从操作数栈中弹出nargs参数值和objectref，并调用调用程序方法句柄。调用就像通过执行一个invokevirtual指令一样发生，该指令指示符号引用R的运行时常量池索引，其中: • R是对类的方法的符号引用; • 对于要在其中找到方法的类的符号引用，R指定java.lang.invoke.MethodHandle; • 对于方法名, R指定invoke; • 对于方法的描述符, R指定由D的返回描述符指示的返回类型，并指定java.lang.invoke.VarHandle的第一个参数类型，然后依次是由D的参数描述符指示的参数类型(如果有的话)。 并且其中，好像以下项按顺序被推到操作数栈上: • 对java.lang.invoke.MethodHandle 的实例的引用(调用程序方法句柄); • objectref; • nargs参数值，其中值的数量、类型和顺序必须与调用程序方法句柄的类型描述符一致。
链接异常	在方法的符号引用的解析过程中，可以抛出与方法解析相关的任何异常（§5.4.3.3）。 否则，如果解析的方法是类（静态）方法，invokevirtual指令将抛出IncompatibleClassChangeError。 否则，如果解析的方法是签名多态的，并在java.lang.invoke.MethodHandle类中声明，则在解析从方法符号引用中的描述符派生的方法类型期间，可以抛出与方法类型解析有关的任何异常（§5.4.3.5）。 否则，如果解析的方法是签名多态的，并在java.lang.invoke.VarHandle类中声明，则调用调用程序方法句柄时可能出现的任何链接异常都可以被抛出。调用valueFromMethodName、accessModeType和varHandleExactInvoker方法时不会引发链接异常。

运行时异常	否则，如果objectref为空，invokevirtual指令将抛出NullPointerException。 否则，如果解析的方法不是签名多态的: • 如果所选方法是抽象的，invokevirtual将抛出AbstractMethodError。 • 否则，如果所选方法是本地的，并且实现该方法的代码无法绑定，invokevirtual将抛出一个UnsatisfiedLinkError。 • 否则，如果未选择任何方法，并且存在多个与解析方法的名称和描述符匹配且不是抽象的最大特定的C的超接口方法，invokevirtual将抛出IncompatibleClassChangeError。 • 否则，如果未选择任何方法，并且C中没有与解析方法的名称和描述符匹配且不是抽象的最大特定超接口方法，invokevirtual将抛出AbstractMethodError。 否则，如果解析的方法是签名多态的，并在java.lang.invoke.MethodHandle类中声明，则: • 如果方法名是invokeExact，并且获得的java.lang.invoke.MethodType实例在语义上不等于接收方法句柄objectref的类型描述符，则invokvirtual指令抛出java.lang.invoke.WrongMethodTypeException。 • 如果方法名是invoke，并且获得的java.lang.invoke.MethodType的实例不是在接收方法句柄objectref上调用的java.lang.invoke.MethodHandle的asType方法的有效参数，则invokvirtual指令抛出一个java.lang.invoke.WrongMethodTypeException。 否则，如果解析的方法是签名多态的，并在java.lang.invoke.VarHandle类中声明，则可以抛出调用程序方法句柄调用可能引起的任何运行时异常。调用valueFromMethodName、accessModeType和varHandleExactInvoker方法时不会抛出运行时异常，除非objectref为空时抛出NullPointerException。
注释	nargs参数值和objectref与第一个nargs+1局部变量不是一一对应的。long和double类型的参数值必须存储在两个连续的局部变量中，因此可能需要更多的nargs局部变量来将nargs参数值传递给调用的方法。invokevirtual指令的符号引用可能解析为接口方法。在这种情况下，有可能在类层次结构中没有重写方法，但非抽象接口方法与解析方法的描述符匹配。选择逻辑匹配这样的方法，遵循与invokeinterface相同的规则。

ior

操作	int 按位或
格式	ior
形式	ior = 128 (0x80)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。它们从操作数栈中弹出。int型结果是通过取value1和value2的按位或来计算的。结果被推入操作数栈。

irem

操作	int 取余
格式	irem
形式	irem = 112 (0x70)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。值从操作数栈弹出。int结果是value1 - (value1 / value2) * value2。结果被推入操作数栈。 irem指令的结果是(a/b)*b + (a%b)等于a。即使在这种特殊情况下，被除数是其类型的最大可能大小的负整数，除数为-1(余数为0)，这个恒等式也成立。根据这条规则，只有当被除数为负时，余数运算的结果才可能为负，只有当被除数为正时，余数运算的结果才可能为正。而且，结果的大小总是小于除数的大小。
运行时异常	如果整型余数运算符的除数值为0,irem将抛出ArithmeticException。

ireturn

操作	从方法返回 int
格式	ireturn
形式	ireturn = 172 (0xac)
操作数栈	..., value→ [empty]
描述	当前方法必须有返回类型boolean、byte、char、short或int。该值必须为int类型。如果当前方法是一个同步方法，在调用该方法时进入或重新进入的监视器将被更新并可能退出，就像在当前线程中执行一个monitorexit指令(§monitorexit)一样。如果没有抛出异常，则从当前帧的操作数栈中弹出值(§2.6)，并推入调用者帧的操作数栈中。当前方法的操作数栈上的任何其他值都将被丢弃。 在将值压入调用程序帧的操作数栈之前，可能需要对其进行转换。如果被调用方法的返回类型是byte、char或short，则value将从int转换为返回类型，就像分别执行i2b、i2c或i2s一样。如果被调用方法的返回类型为布尔型，则通过对value和1进行按位与运算将从int收窄为布尔型。 解释器然后将控制返回给方法的调用者，恢复调用者的框架。
运行时异常	如果Java虚拟机实现没有强制执行§2.11.10中描述的结构化锁定规则，那么如果当前方法是一个同步方法，并且当前线程不是调用该方法时进入或重新进入的监视器的所有者，ireturn将抛出IllegalMonitorStateException异常。例如，如果同步方法在其同步的对象上包含monitorexit指令，但没有monitorenter指令，就会发生这种情况。 否则，如果Java虚拟机实现强制执行§2.11.10中描述的结构化锁定规则，并且在当前方法调用过程中违反了第一条规则，那么ireturn将抛出IllegalMonitorStateException异常。

ishl

操作	int 左移
格式	ishl
形式	ishl = 120 (0x78)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。值从操作数栈弹出。通过将value1向左移动s位计算int结果，其中s是value2的低5位的值。结果被推入操作数栈。
运行时异常	这相当于(即使发生溢出)乘以2的s次方。实际使用的移位距离总是在0到31的范围内(闭区间)，就好像value2与一个掩码值0x1f进行了按位逻辑与运算。

istore

操作	将 int 存储到局部变量
格式	istore index
形式	istore = 54 (0x36)
操作数栈	..., value→ ...
描述	index是一个无符号字节，必须是当前帧的局部变量数组的索引(§2.6)。操作数堆栈顶部的值必须是int类型的。它从操作数栈中弹出，并且在index处的局部变量的值被设置为value。
运行时异常	istore操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

istore_<n>

操作	存储 int 到局部变量
格式	istore_<n>
形式	istore_0 = 59 (0x3b) istore_1 = 60 (0x3c) istore_2 = 61 (0x3d) istore_3 = 62 (0x3e)
操作数栈	..., value→ ...
描述	<n>必须是当前帧的局部变量数组的索引（§2.6）。操作数栈顶部的值必须为int类型。它从操作数栈中弹出，并且 < n > 处的局部变量的值设置为value。
注释	每个istore_<n>指令与带索引<n>的istore相同，只是操作数<n>是隐式的。

isub

操作	int 减法
格式	isub
形式	isub = 100 (0x64)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。值从操作数栈弹出。int的结果是value1 - value2。结果被推入操作数栈。 对于int减法，a-b产生的结果与a+(-b)相同。对于int值，从零减等于取反。 结果是真正数学结果的32个低阶位，采用足够宽的二进制补码格式，表示为int类型的值。如果发生溢出，则结果的符号可能与两个值的数学差值的符号不相同。 尽管可能发生溢出，但执行isub指令绝不会抛出运行时异常。

iushr

操作	int 逻辑右移
格式	iushr = 124 (0x7c)
形式	isub = 100 (0x64)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。值从操作数栈弹出。通过将value1右移s位(零扩展)计算int结果，其中s是value2的低5位的值。结果被推入操作数栈。
注释	如果value1为正数，s为value2 & 0x1f，则结果与value1 >> s相同; 如果value1为负数，则结果等于表达式 (value1 >> s) + (2 << ~s)的值。(2 << ~s)项的相加消去了传播的符号位。实际使用的移动距离总是在0到31的范围内，包括在内。实际使用的移位距离总是在0到31的范围内（闭区间）。

ixor

操作	int 按位异或
格式	ixor
形式	ixor = 130 (0x82)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是int类型。它们从操作数栈中弹出。通过将value1和value2进行按位异或来计算int结果。结果被推送到操作数栈上。

jsr

操作	跳转子程序
格式	jsr branchbyte1 branchbyte2
形式	jsr = 168 (0xa8)
操作数栈	...→ ..., address
描述	紧跟此jsr指令之后的指令的操作码地址作为returnAddress类型的值推送到操作数栈上。无符号的branchbyte1和branchbyte2用于构造有符号的16位偏移量，其中偏移量为(branchbyte1 << 8) | branchbyte2。从这个jsr指令的地址的偏移量开始执行。目标地址必须是包含此jsr指令的方法中指令的操作码的地址。
注释	注意，jsr将地址推入操作数栈，而ret(§ret)将地址从局部变量中获取。这种不对称是故意的。 在Java SE 6之前的Oracle对Java编程语言编译器的实现中，jsr指令和ret指令一起在finally子句的实现中使用(§3.13，§4.10.2.5)。

jsr_w

操作	跳转子程序(宽索引)
格式	jsr_w branchbyte1 branchbyte2 branchbyte3 branchbyte4
形式	jsr_w = 201 (0xc9)
操作数栈	...→ ..., address
描述	紧接在这条jsr_w指令后面的指令的操作码地址作为returnAddress类型的值推入操作数栈。无符号的branchbyte1、branchbyte2、branchbyte3和branchbyte4用于构造有符号的32位偏移量，其中偏移量为(branchbyte1 << 24) | (branchbyte2 << 16) | (branchbyte3 << 8) | branchbyte4。从这个jsr_w指令的地址的偏移量开始执行。目标地址必须是包含该jsr_w指令的方法中指令的操作码的地址。
注释	注意，jsr_w将地址推送到操作数栈中，而ret(§ret)将地址从局部变量中取出来。这种不对称是故意的。 在Java SE 6之前的Oracle对Java编程语言编译器的实现中，jsr_w指令和ret指令一起在finally子句的实现中使用(§3.13，§4.10.2.5)。 尽管jsr_w指令的分支偏移量为4字节，但其他因素会将方法的大小限制在65535字节(§4.11)。这个限制可能会在Java虚拟机的未来版本中提高。

l2d

操作	转换 long 为 double
格式	l2d
形式	l2d = 138 (0x8a)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是long类型的。它从操作数栈中弹出，并使用四舍五入舍入策略转换为double结果(§2.8)。结果被推入操作数栈。
注释	l2d指令执行拓宽原生转换(JLS§5.1.2)，这可能会失去精度，因为类型double的值只有53个有效位。

l2f

操作	转换 long 为 float
格式	l2f
形式	l2f = 137 (0x89)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是long类型的。它从操作数栈中弹出，并使用四舍五入策略转换为float结果(§2.8)。结果被推入操作数栈。
注释	l2f指令执行拓宽原生转换(JLS§5.1.2)，可能会失去精度，因为float类型的值只有24位有效位。

l2i

操作	转换 long 为 int
格式	l2i
形式	l2i = 136 (0x88)
操作数栈	..., value→ ..., result
描述	操作数栈顶部的值必须是long类型的。它从操作数栈中弹出，并通过接受long值的低阶32位而丢弃高阶32位来转换为int结果。结果被推入操作数栈。
注释	l2i指令执行收窄原生转换(JLS§5.1.3)。它可能会丢失关于value总体大小的信息。结果也可能没有与value相同的符号。

ladd

操作	long 的加法
格式	ladd
形式	ladd = 97 (0x61)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。值从操作数栈弹出。long结果是value1 + value2。结果被推入操作数栈。 结果是以足够宽的二进制补码格式表示的真正数学结果的64个低阶位，表示为long类型的值。如果发生溢出，则结果的符号可能与两个值的数学和的符号不相同。 尽管可能发生溢出，但ladd指令的执行永远不会引发运行时异常。

laload

操作	从数组加载 long
格式	laload
形式	laload = 47 (0x2f)
操作数栈	..., arrayref, index→ ..., value
描述	arrayref必须是类型reference，并且必须引用其组件类型为long的数组。index必须是int类型的。arrayref和index都是从操作数栈中弹出的。检索index处数组组件中的long值，并将其压入操作数栈。
运行时异常	如果arrayref为空，laload抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的范围内，laload指令将抛出ArrayIndexOutOfBoundsException异常。

land

操作	long 的按位与
格式	land
形式	laload = 47 (0x2f)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。它们从操作数栈中弹出。一个long结果通过对value1和value2进行按位与运算得到。结果被推入操作数栈。

lastore

操作	存储到 long 数组
格式	lastore
形式	lastore = 80 (0x50)
操作数栈	..., arrayref, index, value→ ...
描述	arrayref必须是类型reference，并且必须引用其组件类型为long的数组。index的类型必须是int, value的类型必须是long。arrayref、index和value从操作数栈中弹出。long值存储为按index索引的数组的组件。
运行时异常	如果arrayref为空，lastore抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的范围内，lastore指令将抛出ArrayIndexOutOfBoundsException异常。

lcmp

操作	long 的比较
格式	lcmp
形式	lcmp = 148 (0x94)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。它们都从操作数栈中弹出，并执行有符号整数比较。如果value1大于value2，则将int值1压入操作数栈。如果value1等于value2，则将int值0压入操作数栈。如果value1小于value2，则将int值-1压入操作数栈。

lconst_<l>

操作	压入 long 常量
格式	lconst_<l>
形式	lconst_0 = 9 (0x9) lconst_1 = 10 (0xa)
操作数栈	...→ ..., <l>
描述	将long常量 <l> （0或1）推送到操作数栈上。

ldc

操作	从运行时常量池中压入项
格式	ldc index
形式	ldc = 18 (0x12)
操作数栈	...→ ..., value
描述	index是一个无符号字节，必须是当前类的运行时常量池的有效索引(§2.5.5)。index处的运行时常量池条目必须是可加载的(§5.1)，而不是以下任何一个: • long或double类型的数值常量。 • 对动态计算的常量的符号引用，其字段描述符为J(表示long)或D(表示double)。 如果运行时常量池条目是int或float类型的数值常量，则该数值常量的值将分别作为int或float推入操作数栈。 否则，如果运行时常量池条目是字符串常量，即对String类实例的引用，则value(对该实例的引用)将被推入操作数栈。 否则，如果运行时常量池条目是一个类或接口的符号引用，那么命名的类或接口将被解析(§5.4.3.1)，对Class对象的引用(即value，表示类或接口)将被推入操作数堆栈。 否则，运行时常量池条目是对方法类型、方法句柄或动态计算常量的符号引用。符号引用被解析(§5.4.3.5，§5.4.3.6)，解析结果value被推入操作数栈。
链接异常	在解析符号引用期间，可以抛出与该类符号引用解析相关的任何异常。

ldc_w

操作	从运行时常量池中推入项(宽索引)
格式	ldc_w indexbyte1 indexbyte2
形式	ldc_w = 19 (0x13)
操作数栈	...→ ..., value
描述	无符号indexbyte1和indexbyte2被组装为当前类(§2.5.5)的运行时常量池的无符号16位索引，其中索引的值计算为(indexbyte1 << 8) | indexbyte2。索引必须是当前类的运行时常量池的有效索引。索引处的运行时常量池条目必须是可加载的(§5.1)，而不是以下任何一个: • long或double类型的数值常量。 • 对动态计算的常量的符号引用，其字段描述符为J(表示long)或D(表示double)。 如果运行时常量池条目是int或float类型的数值常量，或字符串常量，则根据为ldc指令提供的规则确定value并将其压入操作数栈。否则，运行时常量池条目是对类、接口、方法类型、方法句柄或动态计算常量的符号引用。根据为ldc指令提供的规则，它被解析并确定value并推入操作数栈。
链接异常	在解析符号引用期间，可以抛出与该类符号引用解析相关的任何异常。
注释	ldc_w指令与ldc指令(§ldc)相同，除了它有更宽的运行时常量池索引。

ldc2_w

操作	从运行时常量池中推入 long 或 double(宽索引)
格式	ldc2_w indexbyte1 indexbyte2
形式	ldc2_w = 20 (0x14)
操作数栈	...→ ..., value
描述	无符号indexbyte1和indexbyte2被组装为当前类(§2.5.5)的运行时常量池的无符号16位索引，其中索引的值计算为(indexbyte1 << 8) | indexbyte2。索引必须是当前类的运行时常量池的有效索引。索引处的运行时常量池条目必须是可加载的(§5.1)，特别是以下其中之一: • long或double类型的数值常量。 • 对动态计算的常量的符号引用，其字段描述符为J(表示long)或D(表示double)。 如果运行时常量池条目是long或double类型的数值常量，则该数值常量的值将分别作为long或double类型压入操作数栈。 否则，运行时常数池条目是对动态计算的常量的符号引用。符号引用被解析(§5.4.3.6)，解析结果value被推入操作数栈。
链接异常	在解析动态计算常量的符号引用期间，可以抛出与动态计算常量解析相关的任何异常。
注释	只有ldc2_w指令的宽索引版本存在;没有ldc2指令推送具有单字节索引的long型或double型指令。

ldiv

操作	long 除法
格式	ldiv
形式	ldiv = 109 (0x6d)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。值从操作数栈弹出。long结果是Java编程语言表达式value1/value2的值。结果被推送到操作数栈上。 long除法舍入到0;即n / d中long值所产生的商是一个long值q，其大小尽可能大，且满足|d·q|≤|n|。此外，当|n|≥|d|和n、d有相同的符号时，q为正，但当|n|≥|d|和n、d有相反的符号时，q为负。有一种特殊情况不满足这一规则:如果被除数是long类型的最大可能大小的负整数，且除数为-1，则发生溢出，结果等于被除数;尽管存在溢出，但在这种情况下不会抛出异常。
运行时异常	如果long除法中的除数值为0,ldiv将抛出ArithmeticException。

lload

操作	从局部变量加载 long
格式	lload index
形式	lload = 22 (0x16)
操作数栈	...→ ..., value
描述	index是无符号字节。index和index+1都必须是当前帧的局部变量数组的索引(§2.6)。index处的局部变量必须包含long。index处的局部变量的值被推入操作数栈。
注释	lload操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

lload_<n>

操作	从局部变量加载 long
格式	lload_<n>
形式	lload_0 = 30 (0x1e) lload_1 = 31 (0x1f) lload_2 = 32 (0x20) lload_3 = 33 (0x21)
操作数栈	...→ ..., value
描述	<n>和<n>+1都必须是当前帧的局部变量数组的索引（§2.6）。<n>处的局部变量必须包含long。<n>处的局部变量的值被推送到操作数栈上。
注释	每个lload_ < n > 指令与带索引 < n > 的lload相同，只是操作数 < n > 是隐式的。

lmul

操作	long 乘法
格式	lmul
形式	lmul = 105 (0x69)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。值从操作数栈弹出。long结果是value1 * value2。结果被推入操作数栈。 结果是以足够宽的二进制补码格式表示的真正数学结果的64个低阶位，表示为long类型的值。如果发生溢出，则结果的符号可能与两个值的数学乘法的符号不相同。 尽管可能会发生溢出，但执行lmul指令永远不会抛出运行时异常。

lneg

操作	long 取反
格式	lneg
形式	lneg = 117 (0x75)
操作数栈	..., value→ ..., result
描述	该值必须为long类型。它从操作数栈弹出。long结果是value，-value的算术取反。结果被推送到操作数栈上。 对于long值，取反与从零减去相同。因为Java虚拟机对整数使用两进制补码表示，而且二进制补码值的范围不是对称的，所以对最大负long值的取反会得到相同的最大负数。尽管发生了溢出，但不会引发异常。 对于所有long值x，-x = (~x)+1。

lookupswitch

操作	通过键匹配和跳转访问跳转表
格式	lookupswitch <0-3 byte pad> defaultbyte1 defaultbyte2 defaultbyte3 defaultbyte4 npairs1 npairs2 npairs3 npairs4 match-offset pairs...
形式	lookupswitch = 171 (0xab)
操作数栈	..., key→ ...
描述	lookupswitch是一个可变长度的指令。紧跟在lookupswitch操作码之后，0到3个字节之间必须充当填充，例如defaultbyte1的起始地址是当前方法起始地址(其第一个指令的操作码)的4个字节的倍数。紧跟在填充之后的是一系列带符号的32位值:default, npairs, 然后是有符号32位值的npairs对。npairs必须大于或等于0。每个npairs对由一个int匹配和一个有符号的32位offset组成。每个带符号的32位值都是由四个无符号字节构成的(byte1 << 24) | (byte2 << 16) | (byte3 << 8) | byte4。 lookupswitch指令的表match-offset对必须根据match按递增的数字顺序排序。 该key必须为int类型，并从操作数栈中弹出。该key将与match值进行比较。如果它等于其中一个，则通过将相应的offset加到该lookupswitch指令的操作码的地址上来计算目标地址。如果key不

	匹配任何match值，则通过将default添加到此lookupswitch指令的操作码地址中来计算目标地址。然后在目标地址继续执行。 可以从每个match-offset对的offset计算出的目标地址，以及从default计算出的目标地址，必须是包含该lookupswitch指令的方法中指令的操作码的地址。
注释	当且仅当包含lookupswitch的方法位于4字节边界上时，lookupswitch指令的4字节操作数所需的对齐保证了这些操作数的4字节对齐。 对match-offset对进行排序，以支持比线性搜索更快的查找例程。

lor

操作	long 的按位或
格式	lor
形式	lor = 129 (0x81)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。它们从操作数栈中弹出。通过对value1和value2进行按位或来计算long结果。结果被推送到操作数栈上。

lrem

操作	long 取余
格式	lrem
形式	lrem = 113 (0x71)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。值从操作数栈中弹出。long结果是value1-（value1/value2）*value2。结果被推送到操作数栈上。 lrem指令的结果是（a/b）*b+（a%b）等于a。即使在这种特殊情况下，被除数是其类型的最大可能大小的负long，除数为-1(余数为0)，这个恒等式也成立。根据这一规则，只有当被除数为负时，余数运算的结果才可能为负;只有当被除数为正时，余数运算的结果才可能为正;而且，结果的大小总是小于除数的大小。
运行时异常	如果long的余数运算符的除数值为0,lrem抛出ArithmeticException。

lreturn

操作	从方法返回 long
格式	lreturn
形式	lreturn = 173 (0xad)
操作数栈	..., value→ [empty]
描述	当前方法的返回类型必须为long。该值必须为long类型。如果当前方法是一个同步方法，在调用该方法时进入或重新进入的监视器将被更新并可能退出，就像在当前线程中执行一个monitorexit指令(§monitorexit)一样。如果没有抛出异常，则从当前帧的操作数栈中弹出value(§2.6)，并推入调用者帧的操作数栈中。当前方法的操作数栈上的任何其他值都将被丢弃。 解释器然后将控制返回给方法的调用者，恢复调用者的框架。
运行时异常	如果Java虚拟机实现没有强制执行§2.11.10中描述的结构化锁定规则，那么如果当前方法是一个同步方法，并且当前线程不是调用该方法时进入或重新进入的监视器的所有者，lreturn将抛出IllegalMonitorStateException异常。例如，如果同步方法在其同步的对象上包含monitorexit指令，但没有monitorenter指令，就会发生这种情况。 否则，如果Java虚拟机实现强制执行§2.11.10中描述的结构化锁定规则，并且在当前方法调用过程中违反了第一条规则，那么lreturn将抛出IllegalMonitorStateException异常。

lshl

操作	long 左移位
格式	lshl
形式	lshl = 121 (0x79)
操作数栈	..., value1, value2→ ..., result
描述	value1必须是long类型，value2必须是int类型。值从操作数栈弹出。long结果是通过将value1向左移动s位来计算的，其中s是value2的低6位。结果被推入操作数栈。
注释	这相当于(即使发生溢出)乘以2的s次方。因此，实际使用的移位距离总是在0到63的范围内（闭区间），就好像value2与掩码0x3f进行了按位逻辑与运算。

lshr

操作	long 算术右移位
格式	lshr
形式	lshr = 123 (0x7b)
操作数栈	..., value1, value2→ ..., result
描述	value1必须是long类型，value2必须是int类型。值从操作数栈弹出。long结果是通过将value1右移s位(带符号扩展)来计算，其中s是value2的低6位的值。结果被推入操作数栈。
注释	结果值是floor(value1 / 2^s)，其中s是value2 & 0x3f。对于非负值1，这相当于用2的s次方截断long的除法。因此，实际使用的移位距离总是在0到63的范围内（闭区间），就好像value2与掩码0x3f进行了按位逻辑与运算。

lstore

操作	存储 long 到局部变量
格式	lstore index
形式	lstore = 55 (0x37)
操作数栈	..., value→ ...
描述	index是无符号字节。index和index+1都必须是当前帧的局部变量数组的索引(§2.6)。操作数栈顶部的值必须是long类型的。它从操作数栈中弹出，并且在index和index+1处的局部变量被设置为value。
注释	lstore操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

lstore_<n>

操作	存储 long 到局部变量
格式	lstore_<n>
形式	lstore_0 = 63 (0x3f) lstore_1 = 64 (0x40) lstore_2 = 65 (0x41) lstore_3 = 66 (0x42)
操作数栈	..., value→ ...
描述	<n>和<n>+1都必须是当前帧的局部变量数组的索引（§2.6）。操作数栈顶部的值必须为long类型。它从操作数栈中弹出，并且<n>和<n>+1处的局部变量设置为value。
注释	每个lstore_<n>指令都与索引为<n>的lstore相同，只是操作数<n>是隐式的。

lsub

操作	long 的减法
格式	lsub
形式	lsub = 101 (0x65)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。值从操作数栈中弹出。long结果是value1-value2。结果被推送到操作数栈上。 对于long减法，a-b产生与a+(-b)相同的结果。对于long值，从零减去与取反相同。 结果是真正数学结果的64个低阶位，采用足够宽的2进制补码格式，表示为long类型的值。如果发生溢出，则结果的符号可能与两个值的数学差的符号不同。 尽管可能发生溢出，但执行lsub指令决不会引发运行时异常。

lushr

操作	long 的逻辑右移
格式	lushr
形式	lushr = 125 (0x7d)
操作数栈	..., value1, value2→ ..., result
描述	value1的类型必须为long，value2的类型必须是int。值从操作数栈中弹出。long结果的计算方法是将value1逻辑右移s位（零扩展），其中s是value2的低6位的值。结果被推入操作数栈。
注释	如果value1为正数，s为value2 & 0x3f，则结果与value1 >> s相同; 如果value1为负数，则结果等于表达式 (value1 >> s) + (2L << ~s)的值。(2L << ~s)项的相加消去了传播的符号位。实际使用的移动距离总是在0到63的范围内（闭区间）。

lxor

操作	long 的按位异或
格式	lxor
形式	lxor = 131 (0x83)
操作数栈	..., value1, value2→ ..., result
描述	value1和value2都必须是long类型。它们从操作数栈中弹出。通过对value1和value2进行按位异或来计算long结果。结果被推入操作数栈。

monitorenter

操作	进入对象的监视器
格式	monitorenter
形式	monitorenter = 194 (0xc2)
操作数栈	..., objectref→ ...
描述	objectref必须是类型reference。 每个对象都与一个监视器相关联。当且仅当监视器有所有者时，它才被锁定。执行monitorenter的线程尝试获得与objectref相关的监视器的所有权，如下所示: • 如果与objectref关联的监视器的条目计数为零，则线程进入监视器并将其条目计数设置为1。然后，线程就是监视器的所有者。 • 如果线程已经拥有与objectref关联的监视器，那么它将重新进入监视器，增加其条目计数。 • 如果另一个线程已经拥有与objectref关联的监视器，那么该线程将阻塞，直到监视器的条目计数为零，然后再次尝试获得所有权。
运行时异常	如果objectref为空，monitorenter抛出NullPointerException异常。
注释	一个monitorenter指令可以与一个或多个monitorexit指令(§monitorexit)一起使用，以在Java编程语言(§3.14)中实现一个同步语句。monitorenter和monitorexit指令并不用于同步方法的实现，尽管它们可以用于提供等价的锁定语义。调用同步方法时的监视入口，以及返回时的监视出口，都由Java虚拟机的方法调用和返回指令隐式处理，就像使用了monitorenter和monitorexit一样。 监视器与对象的关联可以通过各种超出本规范范围的方式进行管理。例如，监视器可以与对象同时分配和释放。或者，它可以在线程试图获得对对象的独占访问时被动态分配，并在稍后某个时间当没有线程保留在对象的监视器中时被释放。 Java编程语言的同步结构除了入口和出口之外，还需要对监视器上的操作的支持。这些包括等待监视器（Object.wait）和通知其他线程等待监视器（Object.notifyAll和Object.notify）。Java虚拟机提供的标准包java.lang支持这些操作。Java虚拟机的指令集中没

	有对这些操作的显式支持。

monitorexit

操作	退出对象的监视器
格式	monitorexit
形式	monitorexit = 195 (0xc3)
操作数栈	..., objectref→ ...
描述	objectref必须是类型reference。 执行monitorexit的线程必须是与objectref引用的实例相关联的监视器的所有者。 线程减少与objectref关联的监视器的条目计数。如果结果是条目计数的值为零，则线程退出监视器，不再是它的所有者。其他正在阻塞以进入监视器的线程可以尝试这样做。
运行时异常	如果objectref为空，monitorexit抛出NullPointerException异常。 否则，如果执行monitorexit的线程不是与objectref引用的实例相关联的监视器的所有者，则monitorexit会抛出IllegalMonitorStateException异常。 否则，如果Java虚拟机实现强制执行了§2.11.10中描述的结构化锁定规则，并且如果这个monitorexit指令的执行违反了第二条规则，那么monitorexit将抛出一个IllegalMonitorStateException异常。
注释	一个或多个monitorexit指令可以与一个monitorenter指令(§monitorenter)一起使用，以在Java编程语言(§3.14)中实现一个同步语句。monitorenter和monitorexit指令并不用于同步方法的实现，尽管它们可以用于提供等价的锁定语义。 Java虚拟机对同步方法和同步语句中抛出异常的支持是不同的: • 同步方法正常完成时的监视器退出由Java虚拟机的返回指令处理。同步方法突然完成时的监视器退出由Java虚拟机的athrow指令隐式处理。 • 当同步语句中抛出异常时，使用Java虚拟机的异常处理机制(§3.14)从执行同步语句之前进入的监视器中退出。

multianewarray

操作	创建新的多维数组
格式	multianewarray indexbyte1 indexbyte2 dimensions
形式	multianewarray = 197 (0xc5)
操作数栈	..., count1, [count2, ...]→ ..., arrayref
描述	维度操作数是一个无符号字节，必须大于或等于1。它表示要创建的数组的维数。操作数栈必须包含维度值。每个这样的值表示要创建的数组的某个维度中组件的数量，必须为int类型，且必须非负。count1是第一维度中的所需长度，count2是第二维度中的长度，等等。 所有count值都从操作数栈中弹出。无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对类、数组或接口类型的符号引用。命名的类、数组或接口类型被解析(§5.4.3.1)。生成的条目必须是维度大于或等于维数的数组类类型。 从垃圾收集堆中分配一个新的数组类型的多维数组。如果任何count值为零，则不会分配后续维度。第一维中的数组组件被初始化为第二维类型的子数组，依此类推。数组最后分配维度的组件初始化为数组类型的元素类型的默认初始值（§2.3，§2.4）。新数组的引用arrayref被推送到操作数栈上。
链接异常	在解析类、数组或接口类型的符号引用时，可以抛出§5.4.3.1中记录的任何异常。 否则，如果当前类没有访问已解析数组类的元素类型的权限，multianewarray将抛出IllegaAccessError。
运行时异常	否则，如果操作数栈上的任何维度值小于零，则multianewarray指令抛出NegativeArraySizeException异常。
注释	在创建单个维度的数组时，使用newarray或anewarray(§newarray，§anewarray)可能更有效。

	通过运行时常量池引用的数组类可能比multianewarray指令的维度操作数具有更多的维度。在这种情况下，只创建数组维度的第一个维度。

new

操作	创建新对象
格式	new indexbyte1 indexbyte2
形式	new = 187 (0xbb)
操作数栈	...→ ..., objectref
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对类或接口类型的符号引用。命名的类或接口类型被解析(§5.4.3.1)，结果应该是类类型。从垃圾收集堆中为该类的新实例分配内存，并且新对象的实例变量初始化为默认初始值(§2.3，§2.4)。对实例的引用objectref被推入操作数栈。 在成功解析类时，如果它还没有初始化，则初始化它(§5.5)。
链接异常	在解析类或接口类型的符号引用时，可以抛出§5.4.3.1中记录的任何异常。 否则，如果对类或接口类型的符号引用解析为接口或抽象类，new将抛出一个InstantiationError。
运行时异常	否则，如果这个new指令的执行导致了被引用类的初始化，new可能会抛出一个Error，详情见JLS§15.9.4。
注释	new指令不完全创建一个新实例;直到在未初始化的实例上调用了实例初始化方法(§2.9.1)，实例的创建才会完成。

newarray

<table>
<tr><td>操作</td><td>创建新数组</td></tr>
<tr><td>格式</td><td><table><tr><td>newarray</td></tr><tr><td>atype</td></tr></table></td></tr>
<tr><td>形式</td><td>newarray = 188 (0xbc)</td></tr>
<tr><td>操作数栈</td><td>..., count→
..., arrayref</td></tr>
<tr><td>描述</td><td>count必须为int类型。它从操作数栈中弹出。count表示要创建的数组中元素的数量。
atype是指示要创建的数组类型的代码。必须为以下值之一:
表 6.5.newarray-A. 数组类型代码<table><tr><th>数组类型</th><th>atype</th></tr><tr><td>T_BOOLEAN</td><td>4</td></tr><tr><td>T_CHAR</td><td>5</td></tr><tr><td>T_FLOAT</td><td>6</td></tr><tr><td>T_DOUBLE</td><td>7</td></tr><tr><td>T_BYTE</td><td>8</td></tr><tr><td>T_SHORT</td><td>9</td></tr><tr><td>T_INT</td><td>10</td></tr><tr><td>T_LONG</td><td>11</td></tr></table>从垃圾收集堆中分配一个新数组，其组件类型为atype且长度为count。这个新数组对象的引用arrayref被推入操作数栈。新数组的每个元素都初始化为数组类型的元素类型的默认初始值(§2.3，§2.4)。</td></tr>
<tr><td>运行时异常</td><td>如果count小于0,newarray抛出NegativeArraySizeException异常。</td></tr>
<tr><td>注释</td><td>在Oracle的Java虚拟机实现中，类型为boolean的数组(atype为T_BOOLEAN)被存储为8位值的数组，并使用baload和bastore指令(§baload，§bastore)进行操作，这些指令也访问类型为byte的数组。其他实现可能实现打包布尔数组;仍然必须使用baload和bastore指令来访问这些数组。</td></tr>
</table>

nop

操作	什么都不做
格式	nop
形式	nop = 0 (0x0)
操作数栈	没有变化
描述	什么都不做

pop

操作	弹出顶部操作数栈值
格式	pop
形式	pop = 87 (0x57)
操作数栈	..., value→ ...
描述	从操作数栈中弹出顶部值。 除非value是种类1计算类型的值（§2.11.1），否则不得使用pop指令。

pop2

操作	弹出顶部的一个或两个操作数栈值
格式	pop2
形式	pop2 = 88 (0x58)
操作数栈	形式 1: ..., value2, value1→ ... 其中，value1 和 value2 均为种类 1 计算类型的值（§2.11.1）。 形式 2: ..., value→ ... 其中，value 是种类 2 计算类型的值（§2.11.1）。
描述	从操作数栈中弹出顶部的一个或两个值。

putfield

操作	在对象中设置字段
格式	putfield indexbyte1 indexbyte2
形式	putfield = 181 (0xb5)
操作数栈	..., objectref, value→ ...
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是字段的符号引用(§5.1)，它给出了字段的名称和描述符，以及字段所在类的符号引用。引用字段被解析(§5.4.3.2)。 putfield指令存储的值的类型必须与被引用字段的描述符兼容(§4.3.2)。如果字段描述符类型为boolean、byte、char、short或int，则值必须为int。如果字段描述符类型为float、long或double，则值必须分别为float、long或double。如果字段描述符类型是引用类型，那么值的类型必须与字段描述符类型赋值兼容(JLS§5.2)。如果字段是final，它必须在当前类中声明，并且指令必须出现在当前类的实例初始化方法中(§2.9.1)。 value和objectref从操作数栈中弹出。 objectref必须是类型reference，而不是数组类型。 如果值是int类型且字段描述符类型是布尔型，则通过对value和1进行按位与运算来收窄int值，结果是value'。objectref中的引用字段被设置为value'。 否则，将objectref中的引用字段设置为value。
链接异常	在解析对字段的符号引用的过程中，可以抛出任何与字段解析相关的异常(§5.4.3.2)。 否则，如果解析的字段是静态字段，putfield抛出IncompatibleClassChangeError。 否则，如果解析的字段是final，则必须在当前类中声明它，并且指令必须出现在当前类的实例初始化方法中。否则，抛出IllegalAccessError。

运行时异常	否则，如果objectref为空，putfield指令抛出NullPointerException。

putstatic

操作	在类中设置静态字段
格式	putstatic indexbyte1 indexbyte2
形式	putstatic = 179 (0xb3)
操作数栈	..., value→ ...
描述	无符号indexbyte1和indexbyte2被用来在当前类(§2.6)的运行时常量池中构造一个索引，其中索引的值为(indexbyte1 << 8) | indexbyte2。索引处的运行时常量池条目必须是对字段的符号引用(§5.1)，它给出了字段的名称和描述符，以及对要查找字段的类或接口的符号引用。引用字段被解析(§5.4.3.2)。 成功解析该字段后，如果类或接口尚未初始化，则声明该解析字段的类或接口将被初始化(§5.5)。 putstatic指令存储的值的类型必须与引用字段的描述符兼容(§4.3.2)。如果字段描述符类型为boolean、byte、char、short或int，则值必须为int。如果字段描述符类型为float、long或double，则值必须分别为float、long或double。如果字段描述符类型是引用类型，那么值的类型必须与字段描述符类型赋值兼容(JLS§5.2)。如果字段为final，它必须在当前类或接口中声明，并且指令必须出现在当前类或接口的类或接口初始化方法中(§2.9.2)。 从操作数栈中弹出值。 如果值是int类型且字段描述符类型是布尔型，则通过对value和1进行按位与运算来收窄int值，结果是value'。类或接口中的引用字段被设置为value'。 否则，将类或接口中的引用字段设置为value。
链接异常	在解析类或接口字段的符号引用时，可以抛出任何与字段解析相关的异常(§5.4.3.2)。 否则，如果解析的字段不是静态(类)字段或接口字段，putstatic抛出IncompatibleClassChangeError。 否则，如果解析字段为final，则必须在当前类或接口中声明它，并且指令必须出现在当前类或接口的类或接口初始化方法中。否

	则，抛出IllegalAccessError。
运行时异常	否则，如果执行这个putstatic指令导致被引用的类或接口初始化，putstatic可能抛出§5.5中详细描述的Error。
注释	putstatic指令只能用于在接口字段初始化时设置该字段的值。接口字段只能赋值一次，在接口初始化时执行接口变量初始化表达式时(§5.5,JLS§9.3.1)。

ret

操作	从子程序返回
格式	ret index
形式	ret = 169 (0xa9)
操作数栈	没有变化
描述	索引是0到255之间的无符号字节。当前帧(§2.6)中index处的局部变量必须包含一个类型为returnAddress的值。局部变量的内容被写入Java虚拟机的pc寄存器，并在那里继续执行。
注释	注意，jsr(§jsr)将地址推入操作数栈，而ret将它从局部变量中取出。这种不对称是故意的。 在Java SE 6之前的Oracle对Java编程语言编译器的实现中，ret指令在finally子句(§3.13，§4.10.2.5)的实现中与jsr和jsr_w指令(§jsr，§jsr_w)一起使用。 ret指令不应该与return指令(§return)混淆。return指令将方法的控制权返回给它的调用程序，而不将任何值传回调用程序。 ret操作码可以与wide指令(§wide)一起使用，使用双字节无符号索引访问局部变量。

return

操作	从方法返回 void
格式	return
形式	return = 177 (0xb1)
操作数栈	...→ [empty]
描述	当前方法必须有返回类型void。如果当前方法是一个同步方法，在调用该方法时进入或重新进入的监视器将被更新并可能退出，就像在当前线程中执行一个monitorexit指令(§monitorexit)一样。 如果没有抛出异常，则丢弃当前帧(§2.6)的操作数栈上的任何值。 解释器然后将控制返回给方法的调用者，恢复调用者的帧。
运行时异常	如果Java虚拟机实现没有强制执行§2.11.10中描述的结构化锁定规则，那么如果当前方法是一个同步方法，并且当前线程不是调用该方法时进入或重新进入的监视器的所有者，return将抛出IllegalMonitorStateException异常。例如，如果同步方法在其同步的对象上包含monitorexit指令，但没有monitorenter指令，就会发生这种情况。 否则，如果Java虚拟机实现强制执行§2.11.10中描述的结构化锁定规则，并且在当前方法调用过程中违反了第一条规则，则return抛出一个IllegalMonitorStateException异常。

saload

操作	从数组加载 short
格式	saload
形式	saload = 53 (0x35)
操作数栈	..., arrayref, index→ ..., value
描述	arrayref必须是类型reference，并且必须引用其组件类型为short的数组。index必须是int类型的。arrayref和index都是从操作数栈中弹出的。检索index处数组的组件，并将其符号扩展为int值。该值被压入操作数栈。
运行时异常	如果arrayref为空，saload抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的边界内，saload指令将抛出ArrayIndexOutOfBoundsException异常。

sastore

操作	存储到 short 数组
格式	sastore
形式	sastore = 86 (0x56)
操作数栈	..., arrayref, index, value→ ...
描述	arrayref必须是类型reference，并且必须引用其组件类型为short的数组。index和value都必须是int类型。arrayref、index和value从操作数栈中弹出。int值被截断为short，并存储为按index索引的数组的组件。
运行时异常	如果arrayref为空，sastore抛出NullPointerException异常。 否则，如果index不在arrayref引用的数组的边界内，sastore指令将抛出ArrayIndexOutOfBoundsException异常。

sipush

操作	压入 short
格式	sipush byte1 byte2
形式	sipush = 17 (0x11)
操作数栈	...→ ..., value
描述	直接的无符号byte1和byte2值被组装成一个中间short，其中short的值为(byte1 << 8) | byte2。然后将中间值符号扩展为int值。该值被压入操作数栈。

swap

操作	交换最上面的两个操作数栈值
格式	swap
形式	swap = 95 (0x5f)
操作数栈	..., value2, value1→ ..., value1, value2
描述	交换最上面的两个操作数栈值。 除非value1和value2都是种类1计算类型的值(§2.11.1)，否则swap指令不能被使用。
注释	Java虚拟机不提供在种类2计算类型的操作数上实现交换的指令。

tableswitch

操作	通过索引和跳转访问跳转表
格式	tableswitch <0-3 byte pad> defaultbyte1 defaultbyte2 defaultbyte3 defaultbyte4 lowbyte1 lowbyte2 lowbyte3 lowbyte4 highbyte1 highbyte2 highbyte3 highbyte4 jump offsets...
形式	tableswitch = 170 (0xaa)
操作数栈	..., index→ ...
描述	tableswitch是一个可变长度的指令。紧跟在tableswitch操作码之后，0到3个字节之间必须充当填充符，这样defaultbyte1的起始地址从当前方法开始(它的第一个指令的操作码)的4个字节的倍数开始。紧跟在填充之后的是构成三个有符号32位值的字节：default, low, 和high。紧随其后的是构成一系列high - low + 1有符号32位偏移量的字节。low的值必须小于或等于high的值。high - low + 1有符号32位偏移被视为基于0的跳转表。每一个有符号的32位值

	都构造为(byte1 << 24) | (byte2 << 16) | (byte3 << 8) | byte4。 index必须为int类型，并且从操作数栈中弹出。如果index小于low或index大于high，则通过将default添加到该tableswitch指令的操作码地址中来计算目标地址。否则，将提取跳转表的位置index - low处的偏移量。目标地址的计算方法是将该偏移量加到tableswitch指令的操作码地址上。然后在目标地址继续执行。 可以从每个跳转表偏移量计算的目标地址，以及可以从默认值计算的目标地址，必须是包含该tableswitch指令的方法中指令的操作码的地址。
注释	当且仅当包含tableswitch的方法从4字节边界开始时，tableswitch指令的4字节操作数所需的对齐保证了这些操作数的4字节对齐。

wide

操作	将局部变量索引扩展额外的字节
格式 1	wide <opcode> indexbyte1 indexbyte2 其中<opcode>是iload, fload, aload, lload, dload, istore, fstore, astore, lstore, dstore, 或ret之一
格式 2	wide iinc indexbyte1 indexbyte2 constbyte1 constbyte2
形式	wide = 196 (0xc4)
操作数栈	与修改后的指令相同
描述	wide指令修改另一个指令的行为。它采用两种格式之一，取决于被修改的指令。第一种形式的wide指令修改以下指令之一：iload, fload, aload, lload, dload, istore, fstore, astore, lstore, dstore, 或ret (§iload, §fload, §aload, §lload, §dload, §istore, §fstore, §astore, §lstore, §dstore, §ret). 第二种形式的wide指令只适用于iinc 指令 (§iinc)。 在这两种情况下，在编译代码中，wide操作码本身后面跟着wide修改的指令的操作码。在这两种形式中，两个无符号字节indexbyte1和indexbyte2遵循修改后的操作码，并被组装成当前帧(§2.6)中局部变量的16位无符号索引，其中索引值为(indexbyte1 << 8) | indexbyte2。计算的索引必须是当前帧的局部变量数组的索引。当wide指令修改lload、dload、lstore或dstore指令时，计算索引(index + 1)后面的索引也必须是局部变量数组的索引。在第二种形式中，两个直接无符号字节constbyte1和constbyte2紧跟

	在代码流中的indexbyte1和indexbyte2之后。这些字节也被组装成一个带符号的16位常量，其中常量是(constbyte1 << 8) | constbyte2。 除了使用更宽的索引和更大的增量范围(在第二种形式的情况下)之外，加宽的字节码操作正常。
注释	尽管我们说wide“修改另一条指令的行为”，但wide指令有效地将组成被修改指令的字节视为操作数，从而改变了该过程中的嵌入式指令。在修改过的iinc指令中，iinc的一个逻辑操作数甚至不在操作码的正常偏移量上。嵌入式指令绝对不能直接执行;它的操作码决不能是任何控制转移指令的目标。