1. 字节码指令
2. 指令的组成

Java虚拟机的指令由一个字节长度的，代表着某种特定操作含义的数字（操作码），以及跟随其后的零至多个代表此操作所需的参数（操作数）构成。由于Java虚拟机采用面向操作数堆栈的架构，大多数指令都不包含操作数，只有一个操作码，操作数都存放在操作数栈中。

1. 指令的特点

限制了操作码的长度为一个字节，则指令集的操作码总数不能超过256条，同时由于Class文件格式放弃了编译后代码的操作数长度对齐，虚拟机在处理那些超过一个字节的数据时，需要从字节中重建出具体数据的结构，即如果处理一个16位长度的整数，就要使用两个字节存储。这样会损失一些性能，但是可以省略掉大量的填充和间隔符号，能够获得短小精干的编译代码，减小数据量，提高传输效率。

1. 数据类型

大多数指令都包含其操作所对应的数据类型信息，如iload指令用于从局部变量表中加载int型的数据到操作数栈中，而fload指令加载的则是float类型的数据，这两条指令的操作在虚拟机内部可能是由同一段代码实现的，但是在Class文件中它们必须拥有各自独立的操作码。

编译器会在编译期或者运行期将byte和short类型的数据带符号扩展为相应的int类型数据，将boolean和char类型数据零位扩展为相应的int类型数据。

1. 加载和存储指令

加载和存储指令用于将数据在栈帧中的局部变量表和操作数栈之间来回传输。

1. load指令

将一个局部变量加载到操作栈： iload , iload\_<n>等等，这种以尖括号结尾的代表着一组指令，代表着（iload\_0, iload1 …），这些都是某个带有一个操作数的通用指令的特殊形式 ，省略掉了显式的操作数，不需要进行取操作数的操作，因为操作数隐含在指令中。

1. stire指令

将一个数值从操作数栈存储到局部变量表中。

1. 运算指令

算术指令用于对两个操作数栈上的值进行某种特定运算，并把结果重新存入到操作栈顶。大体上运算指令分为两种：对整型数据的运算指令和对浮点型数据的运算指令。

1. 整型运算指令

在处理整型数据时，只有当除法指令以及求余指令中出现除数为零的情况时才会抛出ArithmeticException异常。

1. 浮点型运算指令

虚拟机处理浮点数时，需严格遵循IEEE754规范，所有运算结构都必须舍入到适当的精度，非精确的结果必须舍入为可被表示的最接近的精确值，优先选择最低有效为为0的。

1. 类型转换指令

将两种不同的数值类型相互转换，这些转换操作一般用于实现用户代码中的显式类型转换操作 。对于小范围 类型到大范围类型的宽化类型转换，Java虚拟机是直接支持的，无需显式的转换指令。处理窄化类型转换，必须显式地使用转换指令完成，可能会导致零度丢失，溢出等情况。但是数值类型的窄化转换指令永远不会导致虚拟机抛出运行时异常。

1. 对象创建与访问指令

Java虚拟机对于类实例与数组的创建与操作使用了不同的字节码指令，对象创建后，就可以通过对象访问指令获取对象实例 或者数组实例中的字段。

1. 操作数栈管理指令

Java虚拟机提供了直接操作操作数栈的指令，包括出栈，入栈操作 。

1. 控制转移指令

控制转移指令可以让Java虚拟机有条件或者无条件地从指定位置指令的下一条指令继续执行程序，即修改PC寄存器的值。与算术运算规则一致，对于boolean类型，byte类型，char类型和short类型的条件分支比较操作，会使用int类型的比较指令完成。对于Long，float类型和double类型的条件分支比较操作，先执行相应类型的比较运算指令，再返回一个整型值到操作数栈中，随后执行int类型的条件分支比较操作完成整个分支跳转，因此int类型的条件分支指令是最丰富的。

1. 方法调用和返回指令
2. invokevirtual

调用对象的实例方法，根据对象的实际类型进行分派（虚方法分派）。

1. invokeinterface

调用接口方法，会在运行时搜索一个实现了这个接口的方法对象，找出合适的方法进行调用。

1. invokespecial

调用一些需要特殊处理的实例方法，如实例初始化方法，私有方法，父类方法。

1. invokestatic

调用类静态方法。

1. invokedynamic

在运行时动态解析出调用点限定符所引用的方法，并执行该方法。

（十一）异常处理指令

Java程序中显式地抛出异常的操作由athrow指令实现，Java虚拟机中，处理异常（catch语句）不是由字节码指令实现的，而是采用异常表实现的。

（十二） 同步指令

Java虚拟机支持方法级的同步和方法内部一段指令序列的同步，都是使用Monitor锁来实现的

1. 方法级的同步

方法级的同步是隐式的，无需通过字节码指令控制，虚拟机通过从方法常量池中的方法表结构ACC\_SYNCHRONIZED访问标志中得知该方法是否是同步方法，如果是，则执行的线程就要求先成功持有锁，才能执行方法。当方法完成时（正常或者异常退出 ）释放锁。如果一个同步方法执行期间抛出了异常，并且在方法内部无法处理此异常，那么该方法所持有的锁将在异常抛到同步方法边界之外时自动释放。

1. 指令集序列的同步

由synchronized语句块实现的，Java虚拟机的指令集中有monitorenter和monitorexit两条指令来支持synchronized关键字的语义。

编译器必须确保无论方法通过什么方式完成，方法中调用过的每条monitorenter指令都必须有其对应的monitorexit指令，无论该方法正常结束或者异常退出。