1. 字节码执行引擎
2. 概述

物理机和虚拟机都有代码执行能力，物理机的执行引擎建立在处理器，缓存机，指令集和操作系统之上，而虚拟机的执行引擎则是由软件实现的，不会受到物理条件制约地定制指令集与执行引擎的结构体系，能够执行哪些不被硬件支持的指令集格式。

1. 虚拟机实现

《规范》中制定了Java虚拟机字节码执行引擎的概念模型，但是对于不同的虚拟机实现中，执行引擎在执行字节码的时候，通常有解释执行（通过解释器执行）和编译执行（即时编译器产生本地代码执行）两种方式。执行引擎的输入都是字节码二进制流，处理过程是字节码解析执行的等效过程，输出的是执行结果。

1. 运行时栈帧结构
2. 概述

Java虚拟机以方法作为最基本的执行单元，栈帧是进行方法调用和方法执行的数据结构，也是虚拟机运行时数据区中虚拟机栈的栈元素。每一个方法从调用开始到执行结束，都对应着一个栈帧在虚拟机栈里入栈到出栈。

1. 栈帧结构

栈帧中存储着局部变量表，操作数栈，动态连接和方法返回地址等，在编译Java源码时，栈帧中需要多大的局部变量表，多深的操作数栈都已经计算出来了，并且写到方法表的Code属性中，即一个栈帧所需分配的内存不会受到运行时变量数据的影响，仅取决于程序源码和虚拟机栈内存布局。

对于执行引擎来说，在活动线程中，只有位于栈顶的方法才是运行的，只有位于栈顶的栈帧才是生效的，称为当前栈帧，与这个栈帧关联的方法称为当前方法，执行引擎所运行的所有字节码指令都只针对当前栈帧。

1. 局部变量表
2. 存储内容

局部变量表示一组变量值的存储空间，用于存储方法参数和方法内部定义的局部变量，在程序源码被编译为Class文件时，方法的Code属性的max\_locals数据项确定了该方法所需分配的局部变量表的最大容量。

局部变量表的容量以变量槽为单位，对于32位以内的数据类型占用一个变量槽，对于64位的数据类型占用两个连续的变量槽。对于两个变量槽的读取操作的原子性问题，由于局部变量表是建立在线程堆栈上的，属于线程私有的数据，因此不会引起线程问题。

1. 索引

Java虚拟机通过索引定位的方式使用局部变量表，索引值的范围从0开始到局部变量表的最大变量槽数量，如果访问的是32位数据类型，则索引N代表第N个槽；如果访问64位数据类型，则索引N表示第N和第N+1两个变量槽，虚拟机不允许采用任何方式单独地访问其中的一个。

当方法被调用时，Java虚拟机会使用局部变量表完成参数值到参数变量列表的传递过程，即实参到形参的传递。如果调用的实例方法，则局部变量表中的第0位索引的变量槽默认是用于传递方法所属对象实例的引用，在方法中可以通过“this”访问到这个隐藏的参数。分配完参数后，在根据方法体内部定义的变量和作用域分配其他的变量槽。

1. 变量槽可重用

局部变量表中的变量槽是可以重用的，方法体中定义的变量的作用域不一定会覆盖整个方法体，如果当前字节码PC计数器的值已经超过了某个变量的作用域，则该变量对应的变量槽可以交给其他变量来重用。不过，这样做可能对垃圾回收有影响，即使代码已经离开了某个对象的作用域，但是在此之后没有发生过对该变量所对应的局部变量表的变量槽的读写操作，该变量槽还没有被其他变量复用，因此作为GC Roots一部分的局部变量表仍然保持着对该对象的关联，因此无法回收这个对象。

如何解决这个问题呢？ 推荐的编码准则是将该变量用完后设置为null，这样该变量所对应的变量变量槽就会被清空，就可以被回收了。但是，通过即时编译器施加了各种编译优化后，就不需要这种设置为Null的操作了。

1. 局部变量的初始化

对于类变量，是有两个阶段的赋初始值的操作的，一次是在准备阶段，赋予系统零值，另一次是在初始化阶段，赋予代码中的初始值。

对于局部变量，只有代码中的初始值赋值操作，没有系统零值赋值，因此局部变量定义后一定要赋值，不然是无法使用的。

1. 操作数栈
2. 存储内容

同局部变量表一样，操作数栈的最大深度也在编译时写入到Code属性的max\_stacks属性项中，操作数栈的每一个元素都可以是任意Java数据类型。32位的数据类型占一个栈容量，64位的数据类型占两个栈容量。Javac编译器的数据流分析保障了栈不会溢出。

1. 操作栈的作用

当方法开始执行时，方法的操作数栈是空的，执行过程中，会有各种字节码指令往操作数栈中写入和提取内容。操作数栈的元素的数据类型必须与字节码指令的序列严格匹配，编译器一定要保证这一点，同时在类校验阶段的数据流分析也会校验。

1. 动态连接

每个栈帧都包含了一个指向运行时常量池中该栈帧所属方法的引用，为了方便支持调用过程中的动态连接。Class文件的常量池中有大量的符号引用，字节码的方法调用指令以符号引用为参数，一部分会在类加载阶段或者第一次使用时转为直接引用，这种称为静态解析；另一部分会在运行时转为直接引用，称为动态连接。

1. 方法返回地址

方法有两种退出方法的方式，一种是正常退出，另一种是异常退出。

1. 正常退出

执行遇到任意一个方法返回的字节码指令，将返回值传递给上层的方法调用者。

1. 异常退出

在方法执行的过程中遇到了异常，并且这个异常没有在方法体内妥善处理，只要在本方法的异常表中没有所有到匹配的异常处理器，就会导致方法退出，不会有任何返回值。

1. 恢复状态

无论是哪种方法退出方式，退出之后都需要返回到最初方法被调用的位置，要实现这种方式，就需要在方法返回时在栈帧中保存一些信息，来恢复它的上层主调方法的执行状态。对于正常退出去，可以将主调方法的PC计数器的值作为返回地址，栈帧可以保存这个计数器的值；对于异常退出，返回地址是要通过异常处理器确定的，栈帧中一般不会保存这些信息。

1. 方法退出操作

恢复上层方法的局部变量表和操作数栈，将返回值（有的话）压入调用者栈帧的操作数栈中，调整PC计数器的值指向方法调用指令后的一条指令。