KVM的执行引擎——栈和帧、指令集 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/167/2021_2022_KVM_E7_9A_84_E6_89_A7_E8_c104_167342.htm 接下来的两篇将介绍在KVM中字节是如何执行的，这是KVM中比较核心的内容，分为两部分来讲，本篇先介绍虚拟机中的栈和帧是如何实现的。 首先来看一些全局指针，在头文件kvm/vmcommon/h/interpret.h中定义有以下结构：struct GlobalStateStruct { BYTE* gs_ip. /* Instruction pointer (program counter) */ cell* gs_sp. /* Execution stack pointer */ cell* gs_lp. /* Local variable pointer */ FRAME gs_fp. /* Current frame pointer */ CONSTANTPOOL gs_cp. /* Constant pool pointer */}. 这五个变量就像CPU中的寄存器一样，在KVM的运行过程中起到非常基础性的作用。它们分别是程序计数器、执行栈指针、局部变量指针、当前帧指针和当前常量池指针。 Java虚拟机为每一个线程开设一个栈，栈中存储的数据以“ 帧 ”为单位，虚拟机在调用一个新的方法时，会向栈中压入一个新帧，帧内数据是这个方法的运行状态，Java字节码的执行总是在当前帧内进行，方法运行结束时这个帧会被弹出。所以这个栈可以称为“ 方法栈 ”，帧可以称为“ 方法帧 ”。 按照Java虚拟机的规范，一个帧应由三个部分组成：局部变量区，操作数栈和帧数据区。每个帧的局部变量区和操作数栈的大小都可能不一样，要依方法本身的庞大程度而定，但在调用一个方法时，可以根据这个方法的字节码计算出所需要的局部变量区和操作数栈的大小。规范对帧数据区的大小没有规定，帧数据区的大小和内容可由虚拟机实现来决定。 局部变量区： 局

部变量区一般会位于帧中最前面（即地址最小）的位置，它包含了对应方法的参数和局部变量，一般情况下，它的大小是向4字节对齐的，每4字节是一个“字”，变量以“字”为单位来存入。在它的最前面顺序存放的是对应方法的参数，类型为int、float、reference和returnAddress的参数占一个“字”，类型为byte、short和char类型的参数对被转化为int型，所以也占一个字；long和double类型的值要占用两个字。当然，“字长”选为多少是由虚拟机实现自己来决定的，不是一定要选4字节为一个字，如果选8字节构成一个字的话，所有值都只占一个字，更加整齐，但是浪费了很多空间。 如果方法不是静态的，那么虚拟机会自动将方法所在对象的句柄存在局部变量区中索引为0的位置，真正的参数从位置1开始存；而如果方法是静态的，它就与具体的对象没有关系，所以不必存放对象句柄，参数从位置0开始存放。 在局部变量区接下来的空间中，虚拟机可以按照任意的方式来存贮方法内的局部变量。 操作数栈：操作数栈的作用相当于CPU中的通用寄存器，由于Java虚拟机是一台虚拟的机器，它没有真正的寄存器，而Java虚拟机也没有选择与CPU相似的方式来模拟通用寄存器，而是选择了另一种方法 使用栈，Java指令所使用的操作数都从操作数栈中得到。 某方法在被调用的时候，同样可计算出它需要多大的操作数栈，所以在一个帧中，操作数栈的大小也是固定，而它的位置可以由实现来决定，不过在接下来KVM的实例中我们会发现，把操作数栈放在帧的最后面（地址最大）的地方是一个好办法。 帧数据区：帧数据是由虚拟机实现任意设计的，通常它都被用来实现常量池解析和异常处理等等。 下面来看一看，在KVM中如何实现栈和帧

。 数据结构： 在头文件kvm/vmcommon/h/frame.h中定义了栈和帧的结构：/* STACK */struct stackStruct { STACK next. short size. short xxunusedxx. /* must be multiple of 4 on all platforms */ cell cells[STACKCHUNKSIZE].}.typedef struct stackStruct* STACK. 