详细介绍什么是Java的虚拟机 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/475/2021_2022__E8_AF_A6_E7_BB_86_E4_BB_8B_E7_c67_475827.htm 一、什么是Java虚拟机 当你谈到Java虚拟机时，你可能是指： 1、抽象的Java虚拟机规范 2、一个具体的Java虚拟机实现 3、一个运行的Java虚拟机实例 二、Java虚拟机的生命周期 一个运行中的Java虚拟机有着一个清晰的任务：执行Java程序。程序开始执行时他才运行，程序结束时他就停止。你在同一台机器上运行三个程序，就会有三个运行中的Java虚拟机。 Java虚拟机总是开始于一个main（）方法，这个方法必须是公有、返回void、直接受一个字符串数组。在程序执行时，你必须给Java虚拟机指明这个包换main（）方法的类名。 Main（）方法是程序的起点，他被执行的线程初始化为程序的初始线程。程序中其他的线程都由他来启动。Java中的线程分为两种：守护线程（daemon）和普通线程（non-daemon）。守护线程是Java虚拟机自己使用的线程，比如负责垃圾收集的线程就是一个守护线程。当然，你也可 以把自己的程序设置为守护线程。包含Main（）方法的初始线程不是守护线程。 只要Java虚拟机中还有普通的线程在执行，Java虚拟机就不会停止。如果有足够的权限，你可以调用exit（）方法终止程序。 三、Java虚拟机的体系结构 在Java虚拟机的规范中定义了一系列的子系统、内存区域、数据类型和使用指南。这些组件构成了Java虚拟机的内部结构，他们不仅仅为Java虚拟机的实现提供了清晰的内部结构，更是严格规定了Java虚拟机实现的外部行为。 每一个Java虚拟机都由一个类加载器子系统（class loader

subsystem），负责加载程序中的类型（类和接口），并赋予唯一的名字。每一个Java虚拟机都有一个执行引擎（execution engine）负责执行被加载类中包含的指令。 程序的执行需要一定的内存空间，如字节码、被加载类的其他额外信息、程序中的对象、方法的参数、返回值、本地变量、处理的中间变量等等。Java虚拟机将 这些信息统统保存在数据区（data areas）中。虽然每个Java虚拟机的实现中都包含数据区，但是Java虚拟机规范对数据区的规定却非常的抽象。许多结构上的细节部分都留给了 Java虚拟机实现者自己发挥。不同Java虚拟机实现上的内存结构千差万别。一部分实现可能占用很多内存，而其他以下可能只占用很少的内存；一些实现可 能会使用虚拟内存，而其他的则不使用。这种比较精炼的Java虚拟机内存规约，可以使得Java虚拟机可以在广泛的平台上被实现。 数据区中的一部分是整个程序共有，其他部分被单独的线程控制。每一个Java虚拟机都包含方法区（method area）和堆（heap），他们都被整个程序共享。Java虚拟机加载并解析一个类以后，将从类文件中解析出来的信息保存与方法区中。程序执行时创建的 对象都保存在堆中。 当一个线程被创建时，会被分配只属于他自己的PC寄存器“ pc register ”（程序计数器）和Java堆栈（Java stack）。当线程不掉用本地方法时，PC寄存器中保存线程执行的下一条指令。Java堆栈保存了一个线程调用方法时的状态，包括本地变量、调用方法的 参数、返回值、处理的中间变量。调用本地方法时的状态保存在本地方法堆栈中（native method stacks），可能再寄存器或者其他非平台独立的内存中。 Java堆栈有堆栈块（stack frames（or frames））组成。堆栈块包含Java方法调用的状态。当一

个线程调用一个方法时，Java虚拟机会将一个新的块压到Java堆栈中，当这个方法运行结束时，Java虚拟机会将对应的块弹出并抛弃。 Java虚拟机不使用寄存器保存计算的中间结果，而是用Java堆栈在存放中间结果。这是的Java虚拟机的指令更紧凑，也更容易在一个没有寄存器的设备上实现Java虚拟机。图中的Java堆栈中向下增长的，PC寄存器中线程三为灰色，是因为它正在执行本地方法，他的下一条执行指令不保存在PC寄存器中。 四、数据类型（Data Types） 所有Java虚拟机中使用的数据都有确定的数据类型，数据类型和操作都在Java虚拟机规范中严格定义。Java中的数据类型分为原始数据类型（primitive types）和引用数据类型（reference type）。引用类型依赖于实际的对象，但不是对象本身。原始数据类型不依赖于任何东西，他们就是本身表示的数据。 所有Java程序语言中的原始 数据类型，都是Java虚拟机的原始数据类型，除了布尔型（boolean）。当编译器将Java源代码编译为自己码时，使用整型（int）或者字节型 （byte）去表示布尔型。在Java虚拟机中使用整数0表示布尔型的false，使用非零整数表示布尔型的true，布尔数组被表示为字节数组，虽然他 们可能会以字节数组或者字节块（bit fields）保存在堆中。 除了布尔型，其他Java语言中的原始类型都是Java虚拟机中的数据类型。在Java中数据类型被分为：整形的byte，short，int，long；char和浮点型的float，double.Java语言中的数据类型在任何主机上都有同样的范围。 在Java虚拟机中还存在一个Java语言中不能使用的原始数据类型返回值类型（returnValue）。这种类型被用来实现Java程序中的“ finally clauses ”，具体的参见18章的“ Finally Clauses ”。 引用类型可

能被创建为：类类型（class type），接口类型（interface type），数组类型（array type）。他们都引用被动态创建的对象。当引用类型引用null时，说明没有引用任何对象。 Java虚拟机规范只定义了每一种数据类型表示的范围，没有定义在存储时每种类型占用的空间。他们如何存储由Java虚拟机的实现者自己决定。关于浮点型更多信息参见14章“ Floating Point Arithmetic ”。TypeRange byte8-bit signed twos complement integer (-27 to 27 - 1, inclusive) short16-bit signed twos complement integer (-215 to 215 - 1, inclusive) int32-bit signed twos complement integer (-231 to 231 - 1, inclusive) long64-bit signed twos complement integer (-263 to 263 - 1, inclusive) char16-bit unsigned Unicode character (0 to 216 - 1, inclusive) float32-bit IEEE 754 single-precision float double64-bit IEEE 754 double-precision float returnValueaddress of an opcode within the same method referencereference to an object on the heap, or null 