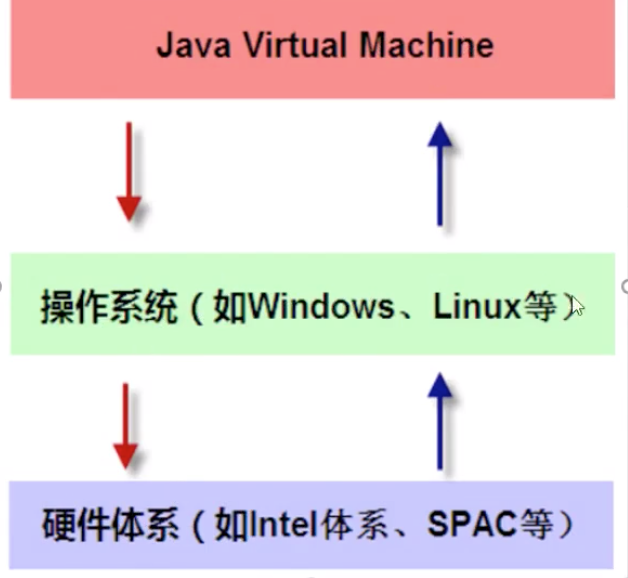
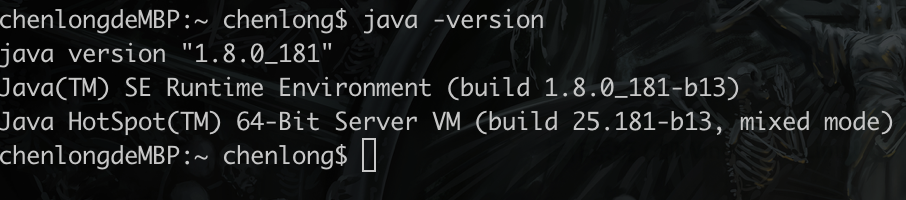
JVM



java虚拟机：指以软件的方式模拟具有完整到硬件系统功能，运行在一个完美的隔离环境中的完整计算机系统，是物理机的软件实现的

常用虚拟机java HotSpot



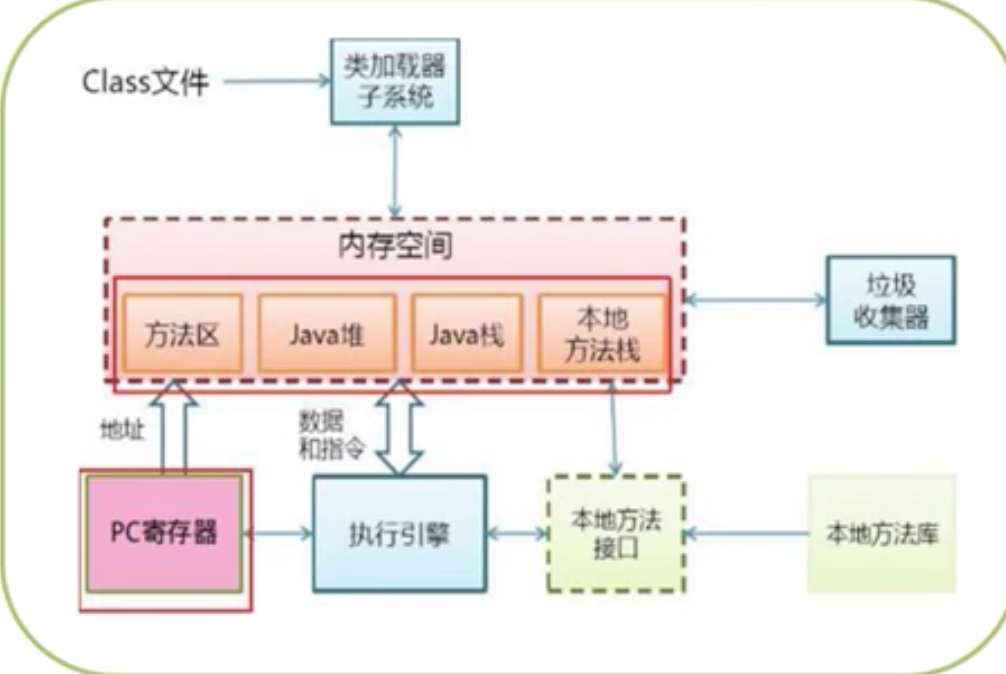
其他虚拟机IBM公司的J9

jvm由三个主要的子系统构成

类加载器子系统

运行时数据区

执行引擎



运行时数据区（内存）：堆，java栈，本地方法栈，方法区，程序计数器

明javap -c 将字节码文件编译成可读

1. 程序计数器

程序计数器是一块比较小的内存空间，他可以看作是当前线程所执行的字节码的行号指示器。

在虚拟机的概念模型里，字节码解释器工作时就是通过改变计数器来选取下一条需要执行的字节码指令，分之，循环，跳转，异常处理，线程恢复等基础功能都需要依赖这个计数器来完成。

1. java虚拟机栈

是线程私有的，每个方法在执行时都会创建一个栈帧，用于存储局部变量，动态连接，方法出口等信息，每一个方法从调用执行的过程，就对应者一个栈帧在虚拟机只能够入栈出栈的过程。（栈后进先出）

3本地方法栈

与虚拟机栈所发挥的作用是类似的，他们之间的区别不过时虚拟机栈为虚拟机执行java方法（字节码）服务，而本地方法栈则为虚拟机使用到Native方法服务。

HotSop将本地方法栈与java虚拟栈合二为一。

4 java堆

堆是java虚拟机所管理堆内存中最大堆一块，java堆是所有线程共享的一块内存区域，内存区域是唯一存放对象的实例，几乎所有的对象实例都在这里分配内存。

java虚拟机规定，java堆可以处于物理上不连续的内存空间中，只要逻辑上是连续的即可。

5方法区

是线程共享的内存区域，它用于虚拟机加载的类信息，常量，静态变量，

**垃圾收集器**

**GC垃圾收集**

1. 引用计数算法

给引用对象引用一个计数器，每一个方法引用他，技术器就加1，当引用失效时就减1，任何计数器为0的对象就不可能在被使用的，

优点：实现简单，判断效率也很高，在大部分情况十分优秀。

缺点：无法解决对象之间相互循环引用的问题。

1. 可达性分析算法

基本思想：以对象作为起始点，从这些节点开始向下搜索，搜索路径成为引用链，当一个对象没有任何引用链表时，证明对象不可用

**引用**

强引用：Object obj=new Object（）；，只要强引用还在，垃圾收集器就不会回收掉被引用的对象。

弱引用：是非必须对象

虚引用

弱引用

**垃圾回收算法**

1. 标记清除法，首先标记所有需要回收的对象，在标记完成后统一回收。

缺点：效率问题：标记和清除效率都不高，空间问题，标记清除之后会产生大量不连续的内存碎片。在以后分配较大的对象时，无法找到足够连续的内存空间

1. 复制算法

将内存按容量分为两块，每次只使用其中的一块，当这一块内存用完了，还存活的对象复制到另一面上面，然后将已经使用过的内存空间全清除。

缺点：缩小了一般的空间，代价太高

1. 标记整理算法

标记所有需要回收的对象，将存活的都向另一段移动，然后直接清理端边界另一段的内存。

1. 分代收集算法（Generational Collection）

根据对象的存活周期的不同将内存划分为几块，一般就分为新生代和老年代，根据各个年代的特点采用不同的收集算法。新生代（少量存活）用复制算法，老年代（对象存活率高）“标记-清理”算法

　　　　补充：分代划分内存介绍

　　　　整个JVM内存总共划分为三代：新生代（Young Generation）、年老代（Old Generation）、持久代（Permanent Generation）

　　　　1、年轻代：所有新生成的对象首先都放在年轻代内存中。年轻代的目标就是尽可能快速的手机掉那些生命周期短的对象。年轻代内存分为一块较大的Eden空间和两块较小的Survior空间，每次使用Eden和其中的一块Survior.当回收时，将Eden和Survior中还存活的对象一次性拷贝到另外一块Survior空间上，最后清理Eden和刚才用过的Survior空间。

　　　　2、老年老代：在年轻代经历了N次GC后，仍然存活的对象，就会被放在老年代中。因此可以认为老年代存放的都是一些生命周期较长的对象。

　　　　3、持久代：基本固定不变，用于存放静态文件，例如Java类和方法。持久代对GC没有显著的影响。持久代可以通过-XX:MaxPermSize=<N>进行设置。

-----------------------------------------------------------------------------------------------------

类加载的过程

就是加载，验证，准备，解析和初始化这5个阶段所执行的具体动作

1. 加载

加载是类加载过程的一个阶段

1. 通过一个类的权限定名来获取定义此类的二进制字节流。
2. 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构。
3. 在内存中生成一个代表这个类的java.lang.Class对象，作为方法区这个类的各种数据的访问入口。
4. 验证

javac编译出来的类都是正确的，但是如果是通过其他途径生成的字节码呢？是不是正确的呢？就比如你自己建一个文本文件，然后重命名该文件为Test.class，然后让JVM来运行这个类，显然是错误的。

a.类文件的结构检查：确保类文件遵从java类文件的固定格式

b.语义检查：确保类本身符合Java语言的语法规定，比如验证final类型的类没有子类，被final修饰的方法不能被覆盖。

c.字节码验证：检验是否有合法的操作数。

1. 准备:为类的静态变量分配内存，并将其初始化为默认值
2. 解析：把类中的符号引用转换为直接引用
3. 初始化：为类的静态变量赋予正确的初始值，上面是赋予默认值，这里是赋予正确的初始值，什么是正确的初始值，就是用户给赋予的值。

Java中编译和运行的区别。

|  |
| --- |
| 编译：编译时是调用检查你的源程序是否有语法错误，如果没有就将其翻译成字节码文件。即.class文件。  Java编译一个类时，如果这个类所依赖的类还没有被编译，编译器就会先编译这个被依赖的类，然后引用，否则直接引用。如果java编译器在指定目录下找不到该类所其依赖的类的.class文件或者.java源文件的话，编译器话报“cant find symbol”的错误。  编译后的字节码文件格式主要分为两部分：常量池和方法字节码。常量池记录的是一些类名，成员变量名等等 以及 符号引用（方法引用，成员变量引用等等）；方法字节码放的是类中各个方法的字节码。  运行：运行时是java虚拟机解释执行字节码文件。java类运行的过程大概可分为两个过程：1、类的加载 2、类的执行。  需要说明的是：JVM主要在程序第一次主动使用类的时候，才会去加载该类。也就是说，JVM并不是在一开始就把一个程序就所有的类都加载到内存中，而是到不得不用的时候才把它加载进来，而且只加载一次。 |

堆和栈

|  |
| --- |
| 栈是一种具有后进先出性质的数据结构，也就是说后存放的先取，先存放的后取。  堆是一种经过排序的树形数据结构，每个结点都有一个值。通常我们所说的堆的数据结构，是指二叉堆。堆的特点是根结点的值最小（或最大），且根结点的两个子树也是一个堆。由于堆的这个特性，常用来实现优先队列，堆的存取是随意的。  为什么要划分堆和栈  1、从软件设计的角度看，栈代表了处理逻辑，而堆代表了数据。这样分开，使得处理逻辑更为清晰。  2、堆与栈的分离，使得堆中的内容可以被多个栈共享。一方面这种共享提供了一种有效的数据交互方式(如：共享内存)，另一方面，堆中的共享常量和缓存可以被所有栈访问，节省了空间。  3、栈因为运行时的需要，比如保存系统运行的上下文，需要进行地址段的划分。由于栈只能向上增长，因此就会限制住栈存储内容的能力。而堆不同，堆中的对象是可以根据需要动态增长的，因此栈和堆的拆分，使得动态增长成为可能，相应栈中只需记录堆中的一个地址即可。  4、体现了Java面向对象这一核心特点（也可以继续说一些自己的理解） |