浅谈Java 虚拟机

1. 什么是java虚拟机？Java 虚拟机都干了些什么事？

相信很多人对java虚拟机的理解停留在程序在软件（idea、Eclipse）编译成.字节码（.class）文件，字节码文件又经过组件（class Loader的子类）装载到java 虚拟机中运行.我们就看到了我们现在的看到的效果，至于中间的细节（jvm怎么帮助我们管理内存，jvm内存泄漏、垃圾回收怎么处理）很少有人去关注。如果我们透过现象看程序的本质那么结果是这样的：我们的程序都是由一个个Java文件组成的，随便看看一个java文件。

**public class** Test {  
  
 **private int a**; *//1、变量* **private int b**;  
 **private int c**;   
  
 **public int** add(**int** x,**int** y)  
 {  
 **c**=**a**+**b**;*//2、指令* **return c**;*//3.控制流* }  
  
}

我们可以看到其实任何Java 文件的本质有三个部分组成，1.变量（数据），2指令。3.控制流（if-else +-X/）等。

哪么我们的这个类是如何在jvm运行的？Cup中的线程（进程）又是怎么样执行的？在了解这些之前，我们聊聊jvm 的主要组成部分（jvm 运行时数据区）：

1. 方法区 保存类信息、常量、静态变量
2. 堆

对于大多数应用来说，Java 堆（Java Heap）是Java 虚拟机所管理的内存中最大的一块。Java 堆是被所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。这一点在Java 虚拟机规范中的描述是：所有的对象实例以及数组都要在堆上分配①，但是随着JIT 编译器的发展与逃逸分析技术的逐渐成熟，栈上分配、标量替换②优化技术将会导致一些微妙的变化发生，所有的对象都分配在堆上也渐渐变得不是那么“绝对”了。Java 堆是垃圾收集器管理的主要区域，因此很多时候也被称做“GC 堆”（Garbage Collected Heap，幸好国内没翻译成“垃圾堆”）。如果从内存回收的角度看，由于现在

收集器基本都是采用的分代收集算法，所以Java 堆中还可以细分为：新生代和老年代；再细致一点的有Eden 空间、From Survivor 空间、To Survivor 空间等。如果从内存分配的角度看，线程共享的Java 堆中可能划分出多个线程私有的分配缓冲区（Thread Local Allocation Buffer，TLAB）。不过，无论如何划分，都与存放内容无关，无论哪个区域，

存储的都仍然是对象实例，进一步划分的目的是为了更好地回收内存，或者更快地分配内存。在本章中，我们仅仅针对内存区域的作用进行讨论，Java 堆中的上述各个区域的分配和回收等细节将会是下一章的主题。

根据Java 虚拟机规范的规定，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，就像我们的磁盘空间一样。在实现时，既可以实现成固定大小的，也可以是可扩展的，不过当前主流的虚拟机都是按照可扩展来实现的（通过-Xmx和-Xms 控制）。如果在堆中没有内存完成实例分配，并且堆也无法再扩展时，将会抛出OutOfMemoryError 异常。

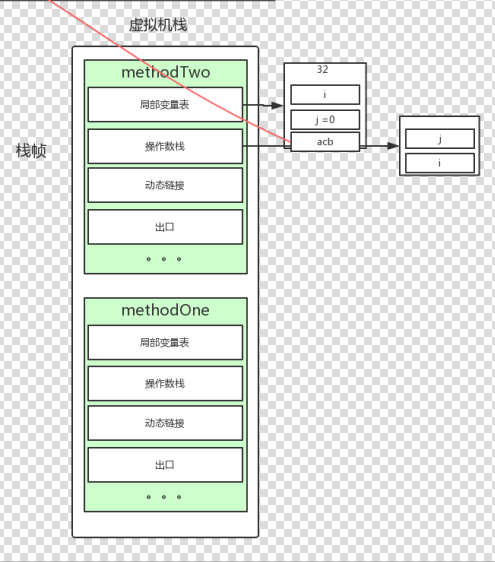
1. 程序计数器：指向当前程序正在执行字节码的地址，行号，每个线程都会拥有一个程序计数器，当多线程中会涉及到线程上下文的切换（cpu通过时间片分配算法执行程序，这个过程很快，很多时候我们感觉是多线程），当前线程可能执行到一半就切换到其他线程了，等一定时间cup可能会再次分配时间片给当前线程，计数器会记住上次执行的任务的状态。
2. 虚拟机栈 ：里面有一个栈帧,运行时方法会存储在栈帧里面，根据下图可以看到栈帧里面又主要有四个部分组成

4.1、局部变量表

4.2、操作数栈

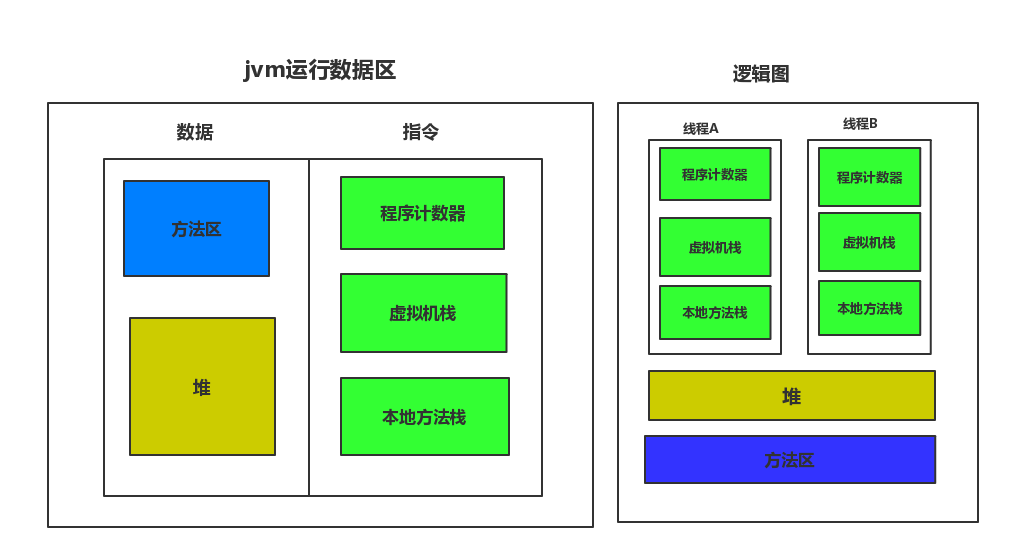
4.3、动态链接

4.4、出口



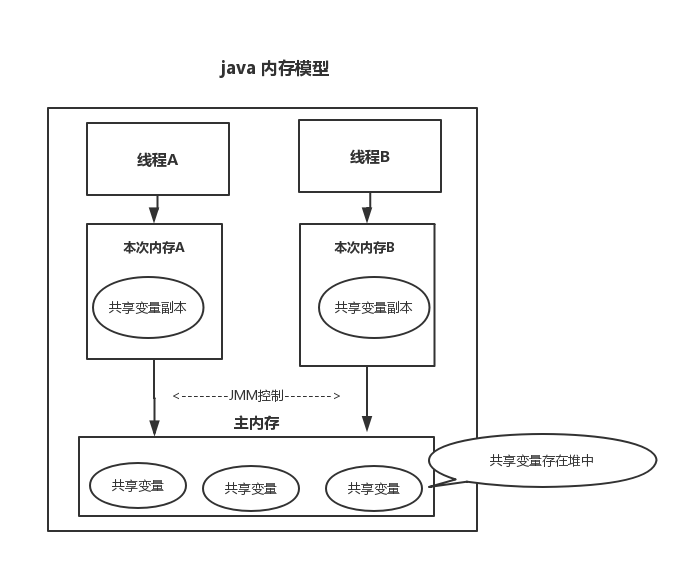
1. 本地方法栈

本地方法栈（Native Method Stacks）与虚拟机栈所发挥的作用是非常相似的，其区别不过是虚拟机栈为虚拟机执行Java 方法（也就是字节码）服务，而本地方法栈则是为虚拟机使用到的Native 方法服务。虚拟机规范中对本地方法栈中的方法使用的语言、使用方式与数据结构并没有强制规定，因此具体的虚拟机可以自由实现它。甚至有的虚拟机（譬如Sun HotSpot 虚拟机）直接就把本地方法栈和虚拟机栈合二为一。与虚拟机栈一样，本地方法栈区域也会抛出StackOverflowError 和OutOfMemoryError异常。



通过图形我们可以清晰的了解到jvm运行时数据区的主要组成部分，当然这也是我们需要特别关注地方。接着我们看到逻辑图：每个线程拥有自己的程序计数器，虚拟机栈，本地方法栈，和其他线程共享堆和方法区。

回头看看我们刚才的Test的执行过程，我们的常量、静态变量等即时编译的代码存储在我们的方法区和堆里面，执行方法的时候会把方法压如虚拟机栈创建的栈帧中。最终编译为我们机器（cup）认识的汇编语言（我们常说的指令）。



1、共享变量主要包括：实例域、静态域、数组元素；

1. 线程共享堆内存；
2. 线程之间的通讯通过jmm控制；
3. 线程A和线程B通讯可以分为两步：

第一步：线程A把本次内存的共享变量刷新到主内存

第二步：线程B从主内存中更新该共享变量到本地内存

1. 可见性：当一个线程在修改共享内存的时候，其他的线程是不能操作这个共享变量的（不能读取到这个变量的值），只有等该线程操作结束把共享变量重新刷新回主内存。其他的的线程才可以操作。

什么是线程安全？

保证线程安全带机制有哪些？