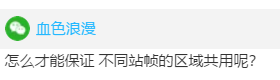
# 课程答疑汇总：

## 第一节课

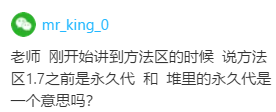


常量池分为运行时常量池和静态常量池：

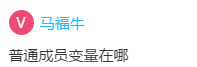
运行时的常量池（JDK1.7开始已经进入堆）。静态常量池（class）---方法区



主要体现在方法中有参数传递的情况。JVM一般会优化栈帧中数据的共享。



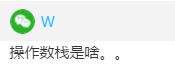
方法区是逻辑概念，任何虚拟机版本或者任意厂商的虚拟机有这个方法区。永久代和元空间是具体实现，HotSpot中jdk1.7及以前是方法区的实现是永久代，JDK1.8及以后是元空间。



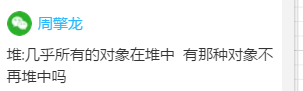
成员变量是跟随Class的实例，即对象，所以内存会跟随堆，在堆中。



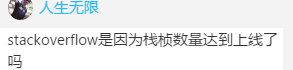
程序计数器是一小块内存，就是记录当前线程正在执行的字节码的地址或者行号，很小，基本上不用拓展，所以不会内存溢出。



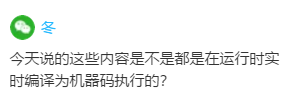
操作数栈是虚拟机栈中的栈帧中包含的一个栈的结构。虚拟机栈是用来压入栈帧的，操作数栈是用来处理操作数的。



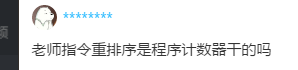
符合逃逸分析逃不出来方法的对象可以使用栈上分配，这个是虚拟机的优化技术。



JVM没有对栈帧数据做限制，但是限制了虚拟机栈的大小，一般情况下是1M，如果我们方法递归调用，不断的在执行过程中压入栈帧（且不出栈），每一个栈帧总会占据一定的空间，这样就会把虚拟机栈的空间给栈满（而HotSpot虚拟机一般虚拟机栈空间不会拓展）所以就会抛出StackOverFlow这样的错误，栈溢出。



大部分都是解释执行。并不是把所有的字节码都翻译成本地代码，而是在虚拟机执行过程中翻译一行执行一行。解释执行的优点在于启动速度快。



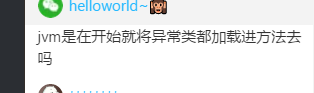
指令重排不是程序计数器干的。

在执行程序时，为了提高性能，编译器和处理器常常会对指令做重排序。重排序分3种类型。

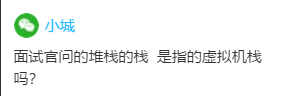
1）编译器优化的重排序。编译器在不改变单线程程序语义的前提下，可以重新安排语句的执行顺序。

2）指令级并行的重排序。现代处理器采用了指令级并行技术（Instruction-LevelParallelism，ILP）来将多条指令重叠执行。如果不存在数据依赖性，处理器可以改变语句对应机器指令的执行顺序。

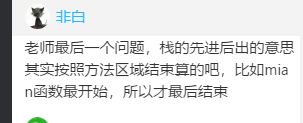
3）内存系统的重排序。由于处理器使用缓存和读/写缓冲区，这使得加载和存储操作看上去可能是在乱序执行。



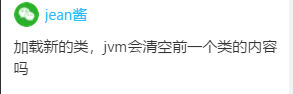
是的，默认情况下都会加载进去，并且不单单只是类，还有实例也会创建，这个属于JVM内存的操作。



这个要根据场景，你这个场景是虚拟机栈，如果是执行字节码往往是操作数栈。

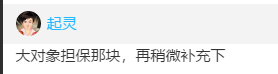


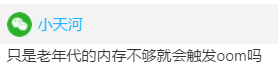
是的，这个是栈的特点，先进后出。



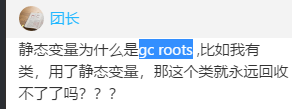
你这个问题应该是类加载器的重复加载一个类的问题，一个类加载器一般情况下不会重复加载。

## 第二节课



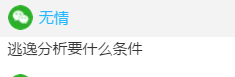


OOM一般触发是在堆，但是也有可能是方法区，也有可能是直接内存，老年代只是堆中一个区域，只是因为又担保策略，新生代就放老年代，所以一般往往会导致老年代放不下去导致OOM.



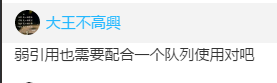
静态变量它本身就是class加载进来就可以直接使用，不像其他的实例变量需要你new出一个类的对象，所以它是作为GCRoots。

用了静态变量，这个类永久不会回收了吗？---类的回收有严格条件，如果这个类回收掉了，这个类上静态变量一般也会跟随类的回收条件。



符合方法逃逸和线程逃逸。

逃逸分析的原理：分析对象动态作用域，当一个对象在方法中定义后，它可能被外部方法所引用，比如：调用参数传递到其他方法中，这种称之为方法逃逸，甚至还有可能被外部线程访问到，例如：赋值给其他线程中访问的变量，这个称之为线程逃逸。

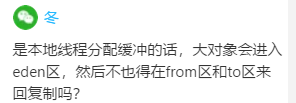


有两种方式创建弱引用：

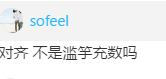
WeakReference类有两个构造函数

WeakReference(T referent) //创建一个指向给定对象的弱引用

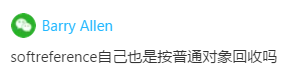
WeakReference(T referent, ReferenceQueue<? super T> q) //创建一个指向给定对象并且登记到给定引用队列的弱引用



一般大对象不会进行本地线程缓冲，大对象很多情况下直接在老年代分配。还有像G1这种垃圾回收器中直接有一个大对象区，专门存放大对象。



需要字节对齐的根本原因在于CPU访问数据的效率问题，如果不对齐，CPU获取数据是一个固定的字节长度一个固定的字节长度获取的，那么有可能发生错误，还有可能效率很低。



软引用只是判断对象的能不能被回收，回收都是垃圾回收器统一处理掉。

## 第三节课

较少没有整理