**>>内存管理**

1. Java虚拟机的分区：方法区，虚拟机栈，本地方法栈，堆，程序计数器，
   1. 程序计数器：当前线程所执行的字节码的行号指示器，每个线程有一个独立的程序计数器。属于线程私有内存。如果执行的native方法区，则程序计数器的值为null
   2. 虚拟机栈：线程私有，存储局部变量，
   3. 本地方法栈：主要储存native方法中的局部变量
   4. Java堆：线程共享，存放对象实例、数组
   5. 方法区：线程共享的内存区域，存储加载的类信息，常量，静态变量；
      1. 运行时常量池：是方法区的一部分。用于存放编译器生成的各种字面量（字符串，final修饰的变量）和符号引用（完全限定名，字段名称和描述符，方法名称和描述符）。
2. 垃圾回收：
   1. 判断一个对象是否存活：引用计数法，可达性分析
      1. 引用计数法：当一个对象被引用的时候，引用计数器+1，当引用失效的时候，引用计数器-1，当引用计数器为0时，对象不可在使用。缺点是：无法解决相互引用的问题。
      2. 可达性分析：通过一系列的GCRoot对象作为根节点，向下搜索，搜索走过的路径计时引用链，当没有一条引用链可以到达一个对象的时候，就说明此对象不可达，判断为垃圾对象。可以作为GCRoot的 对象有虚拟机栈中引用的对象，方法区中静态类属性引用的对象，方法区中常量应用的对象，native方法引用的对象。
3. **垃圾收集算法：**
   1. 标记清除法：将所有垃圾对象进行标记，然后一并清除所有的垃圾对象；缺点：效率不高，标记和清除两个过程的效率都不高；空间问题标记清除后会产生大量的内存碎片，造成内存不完整。
   2. 复制算法：将内存分成对等 的两块区域，只使用其中一块，当一块内存满的时候，触发垃圾回收，将有用的对象复制到另一快内存上，然后回收这块内存上面的垃圾对象。缺点·：内存缩小为原来的一半
   3. 标记整理算法：先将垃圾对象进行标记，然后将存活对象移到内存空间的一边，将边界范围以为的对象进行垃圾回收。
4. **分代收集算法：**
   1. 新生代：将内存分为一块较大的eden区和两块较小的Survivor区，通常比例是8：1：1，每次只使用一块eden区和一块survivor区，当回收时，将eden区和Survivor区中存活的对象移到另一块Survivor区中，然后清理垃圾对象。当survivor区内存不够的时候，对象将被移到老年代。
   2. 老年代：使用标记整理算法

**>>类文件结构**

1.write once, run anywhere!

2.Java虚拟机并不仅仅只是运行Java语言，还可以支持其他语言，其他语言使用对应的编译器将源代码编译成字节码文件（即.class文件）

3.class文件解析

1）class前四个字节称为Magic Number，作用是确定这个class文件是否能被虚拟机接受，因为后缀名是可以修改的。

2）接下来的四个字节是Class文件的版本号，第5，6个字节是Minor Version，第七八个字节是Major Version.

>>虚拟机加载机制

1. 类的生命周期：
   1. 加载->验证->准备->解析->初始化->使用->卸载
2. 对类进行初始化有且仅有五中情况
   1. 遇到new, getstatic, setstatic, invokestatic四条字节码指令，如果类没有过初始化，则先触发初始化
   2. 使用java.lang.reflect包的方法进行反射调用的时候，如果类没有进行过初始化先触发其初始化
   3. 当初始化类的时候，如果父类没有初始化，则先初始化父类；一个接口在初始化的时候，并不要求其父类接口都完成初始化，只是真正使用到父接口的时候才会初始化
   4. 虚拟机启动时，包含main方法的类会初始化
   5. 当使用JDK1.7的动态语言支持时，如果一个java.lang.invoke.MethodHandle实例最后的解析结果REF\_getStatic..的方法句柄，并且对应类没有初始化，则需要先触发器初始化。
   6. 通过子类引用父类中的静态字段不会导致子类初始化
   7. 调用类中的常量也不会触发类的初始化
3. 类加载过程：加载，验证，准备，解析，初始化
   1. 加载过程需要完成三件事：
      1. 通过类的全限定名获取定义此类的二进制文件
      2. 将这个字节流所代表的静态存储结构转化为方法区运行时的数据结构
      3. 在内存中生成java.lang.Class对象，作为方法区各个数据访问入口
   2. 加载和连接是交叉进行的，但是加载和连接的交叉顺序任然是加载在前
   3. 验证是连接的第一步，确保Class文件中的信息符合虚拟机的要求，不会危害虚拟机自身的安全。
   4. 验证大概分为以下四部分：
      1. 验证文件格式：根据文件中的各个位置的十六进制数判断是否符合规范，如魔数0XCAFEBABE开头，主次版本号等等
      2. 元数据验证：对类的元数据信息进行语意校验
      3. 字节码验证：主要是对方法体中的一些验证，包括局部变量的类型，是否会运行到方法体之外的代码等等。
      4. 符号引用验证：确保解析动作能够正常执行
   5. 准备过程：为类变量（不包括实例变量,实例变量在java堆中）分配内存和设置类变量初始值（为默认值0，而不是”=”之后的值）的阶段，内存在方法去中分配。
   6. 解析过程
4. 类加载器：
   1. 两个类是否相等，只有这两个类是由同一个类加载器加载才有意义，即使这两个类来源于同一个class文件，被同一个虚拟机加载，只要加载他们的类加载器不同，那这两个类就必定不相同。
   2. 类加载器的层次关系：类加载器都是以组合方式来复用父类加载器
      1. 启动类加载器<-扩展类加载器<-应用程序加载器<-自定义类加载器
   3. 双亲委派模型：如果一个类加载器收到了类加载的请求，它首先不会自己尝试去加载这个类，而是由父类去加载，每一个层次均是如此，因此所有的类最终都会有启动类加载器加载，只有当父类无法加载的时候，子加载器才会尝试自己去加载。