

2022/3/12 - JVM2

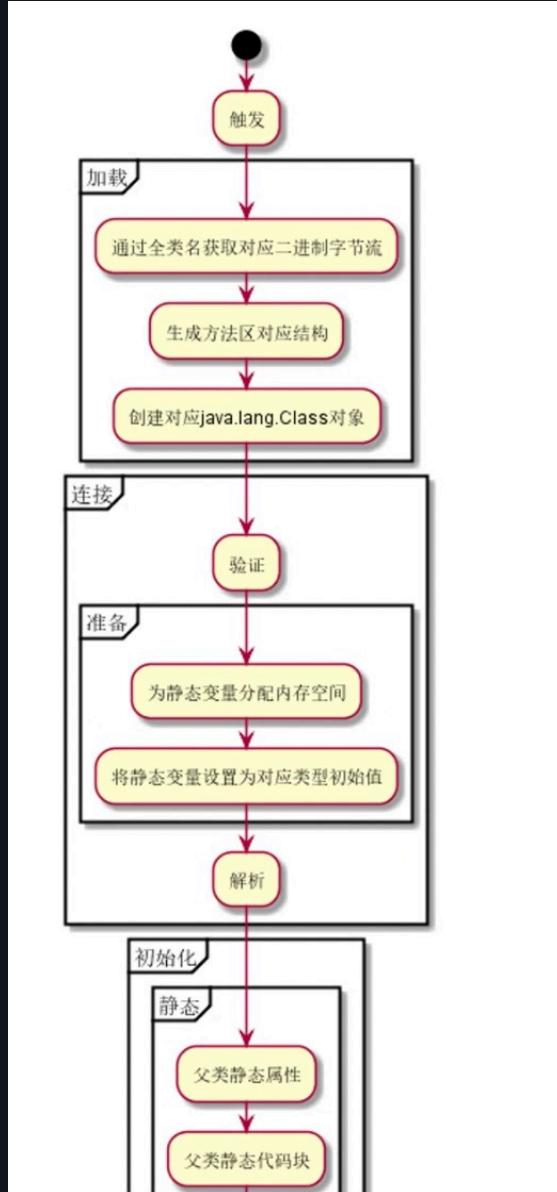
点评

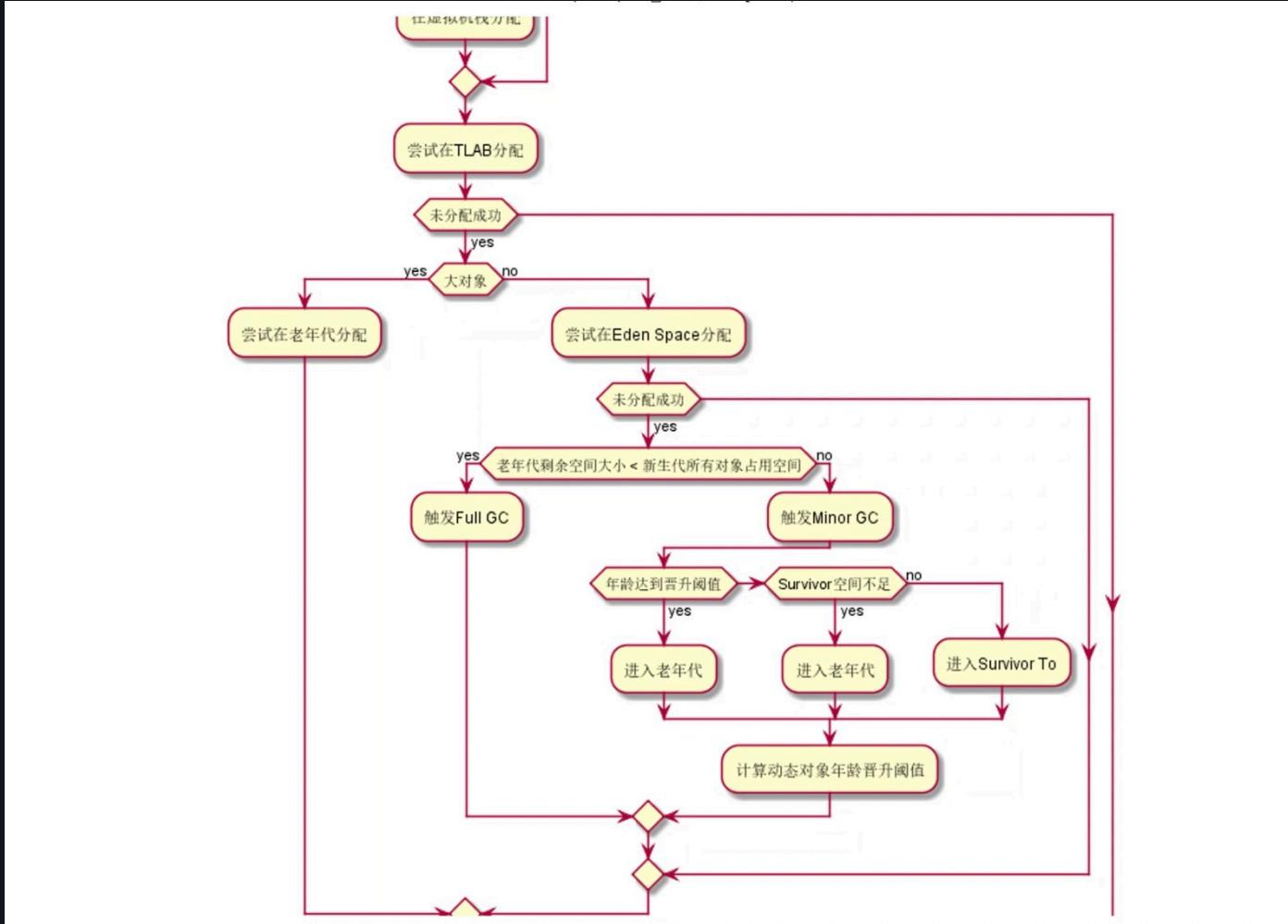
- 1、问题驱动
- 2、方法区为什么变成元空间
- 跟linux的进程有关
- 2、堆怎么进入串池
- 3、跨代引用怎么扫描gcroots
- 4、那些可作为GC Roots的对象
- 5、full GC

<https://developer.aliyun.com/article/849656#slide-23>

- 6、CMS为什么产生浮动垃圾

二组分析





<https://www.teqng.com/2021/04/27/%E8%BF%99-10-%E4%B8%AA%E7%9A%84-jvm-%E9%9D%A2%E8%AF%95%E9%A2%98%EF%BC%8C%E7%89%9B%E9%80%BC%EF%BC%81/>

1、Java 中都有哪些引用类型，他们的区别是什么

2、怎么晋升到老年代

当 Eden 区的空间满了，Java 虚拟机会触发一次 Minor GC，以收集新生代的垃圾，存活下来的对象，则会转移到 Survivor 区。大对象（需要大量连续内存空间的 Java 对象，如那种很长的字符串）直接进入老年态；如果对象在 Eden 出生，并经过第一次 Minor GC 后仍然存活，并且被 Survivor 容纳的话，年龄设为 1，每熬过一次 Minor GC，年龄 +1，若年龄超过一定限制（15），则被晋升到老年态。即长期存活的对象进入老年态。老年代满了而无法容纳更多的对象，Minor GC 之后通常就会进行 Full GC，Full GC 清理整个内存堆 – 包括年轻代和年老代。Major GC 发生在老年代的 GC，清理老年区，经常会伴随至少一次 Minor GC，比 Minor GC 慢 10 倍以上。

Full GC 的触发条件？

调用 System.gc()

老年代空间不足

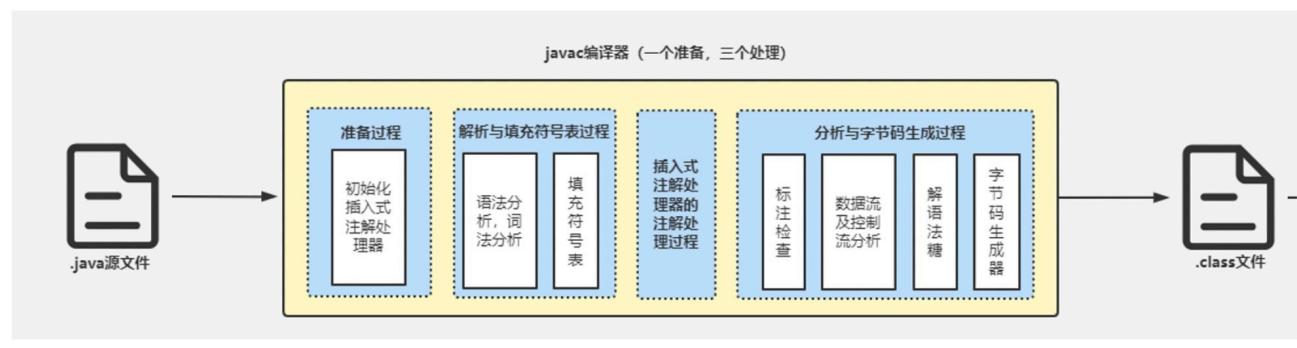
空间分配担保失败

JDK 1.7 及以前的永久代空间不足

对象一定分配在堆中吗？有没有了解逃逸分析技术？

元空间有没有发生溢出

Javac编译



类文件结构

类型	名称	数量
u4	magic	1
u2	minor_version	1
u2	major_version	1
u2	constant_pool_count	1
cp_info	constant_pool	constant_pool_count-1
u2	access_flags	1
u2	this_class	1
u2	super_class	1
u2	interfaces_count	1
u2	interfaces	interfaces_count
u2	fields_count	1
field_info	fields	fields_count
u2	methods_count	1
method_info	methods	methods_count
u2	attributes_count	1
attribute_info	attributes	attributes_count

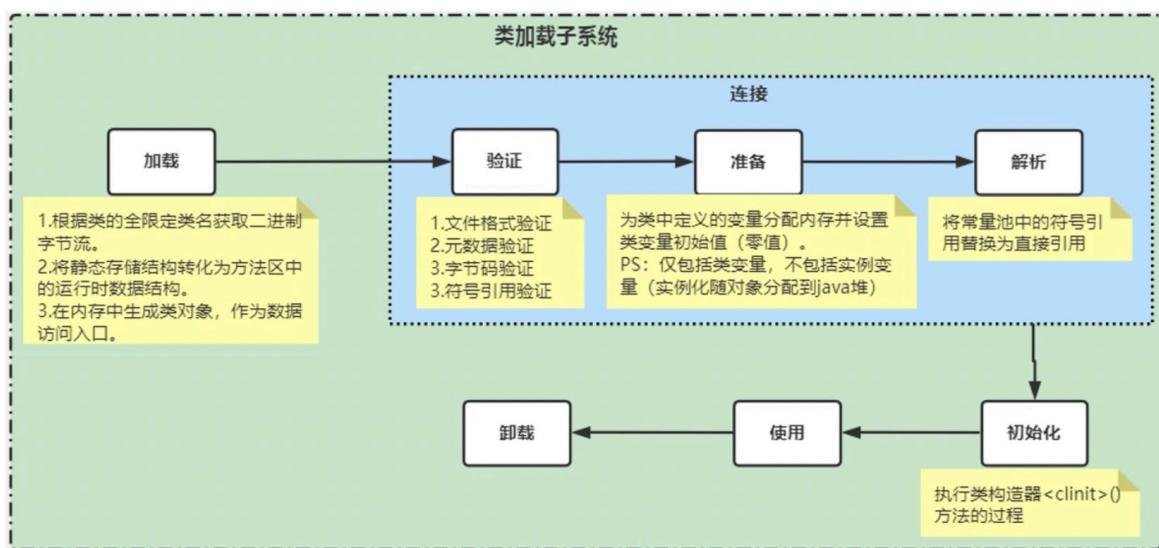
常量池的项目类型

类 型	标 志	描 述
CONSTANT_Utf8_info	1	UTF-8 编码的字符串
CONSTANT_Integer_info	3	整型字面量
CONSTANT_Float_info	4	浮点型字面量
CONSTANT_Long_info	5	长整型字面量
CONSTANT_Double_info	6	双精度浮点型字面量
CONSTANT_Class_info	7	类或接口的符号引用
CONSTANT_String_info	8	字符串类型字面量
CONSTANT_Fieldref_info	9	字段的符号引用
CONSTANT_Methodref_info	10	类中方法的符号引用
CONSTANT_InterfaceMethodref_info	11	接口中方法的符号引用
CONSTANT_NameAndType_info	12	字段或方法的部分符号引用
CONSTANT_MethodHandle_info	15	表示方法句柄
CONSTANT_MethodType_info	16	表示方法类型
CONSTANT_Dynamic_info	17	表示一个动态计算常量

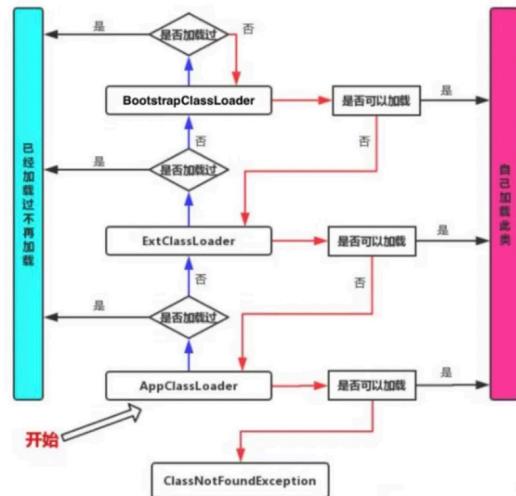
(续)

类 型	标 志	描 述
CONSTANT_InvokeDynamic_info	18	表示一个动态方法调用点
CONSTANT_Module_info	19	表示一个模块
CONSTANT_Package_info	20	表示一个模块中开放或者导出的包

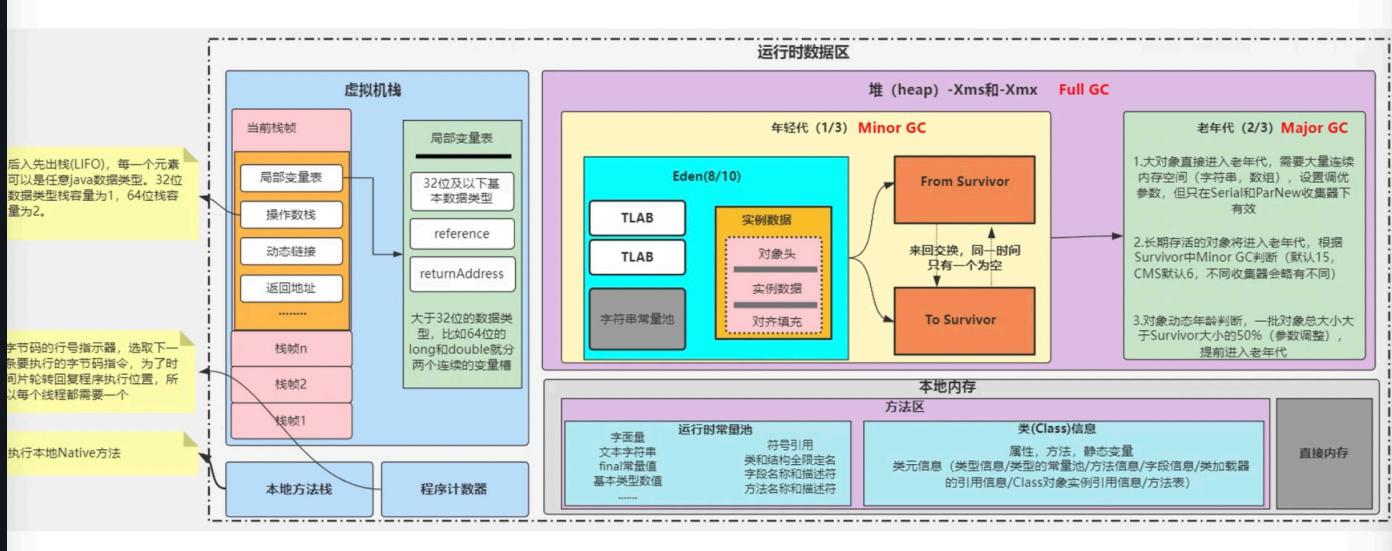
类加载机制



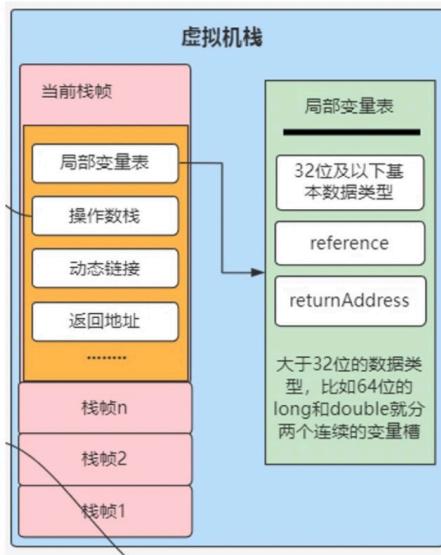
类加载器



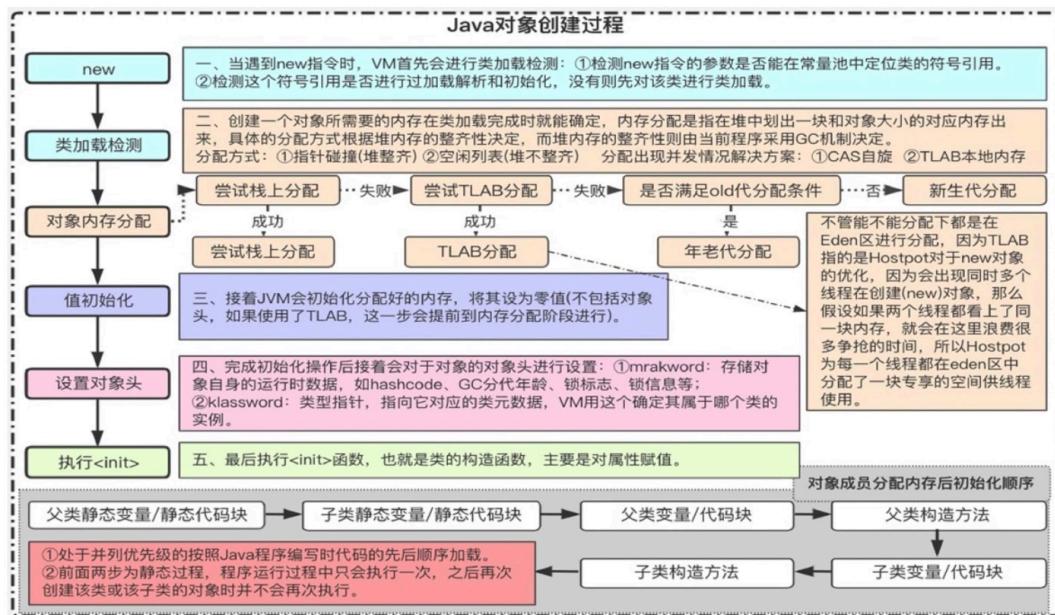
内存区域



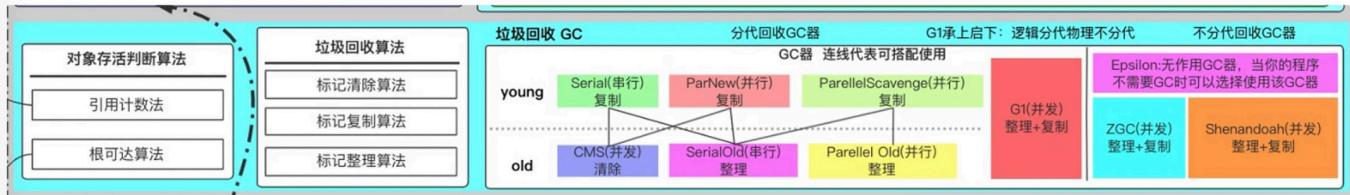
运行时栈帧结构



对象的创建



垃圾收集



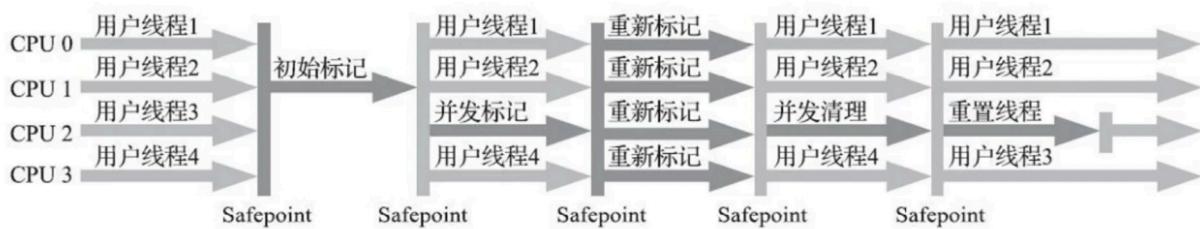
分代收集理论

- 弱分代假说 (Weak Generational Hypothesis) : 绝大多数对象都是朝生夕灭的。
- 强分代假说 (Strong Generational Hypothesis) : 熬过越多次垃圾收集过程的对象就越难以消亡。
- 跨代引用假说 (Intergenerational Reference Hypothesis) : 跨代引用相对于同代引用来说仅占极少数。

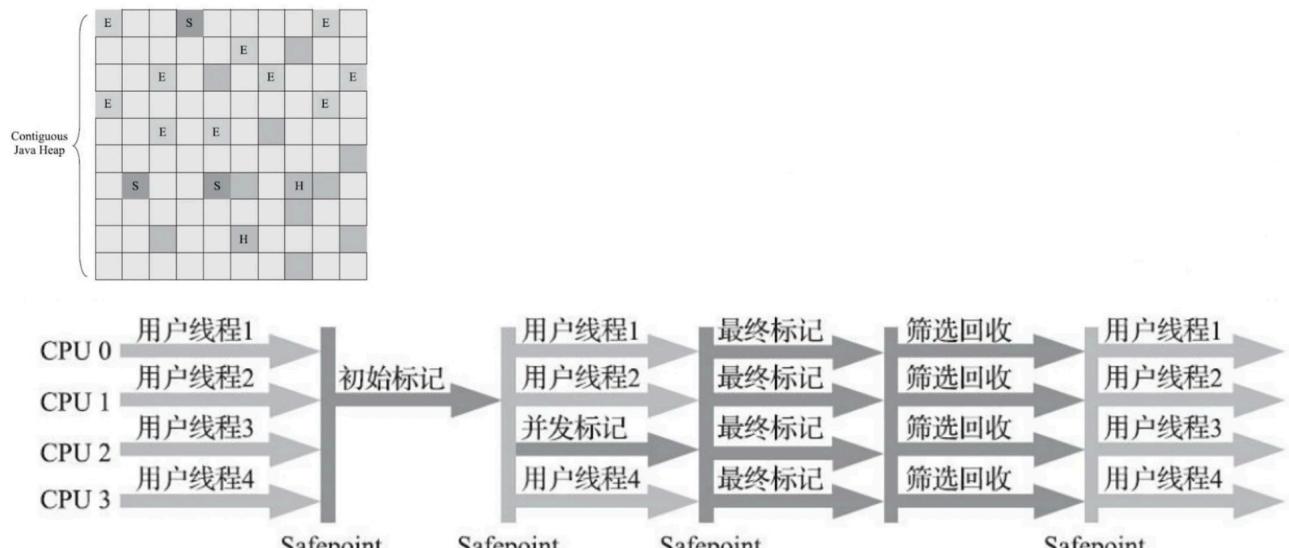
HotSpot的算法细节实现

- 根节点枚举
- 安全点
- 安全区域
- 记忆集和卡表
- 写屏障
- 并发可达性分析

CMS收集器



Garbage First收集器



回收方法区

方法区的垃圾收集主要回收两部分内容：废弃的常量和不再使用的类型。回收废弃常量与回收Java堆中的对象非常类似。

判定一个常量是否“废弃”还是相对简单，而要判定一个类型是否属于“不再被使用的类”的条件就比较苛刻了。

需要同时满足下面三个条件：

1. 该类所有的实例都已经被回收，也就是Java堆中不存在该类及其任何派生子类的实例。
2. 加载该类的类加载器已经被回收，这个条件除非是经过精心设计的可替换类加载器的场景，如OSGi、JSP的重加载等，否则通常是很困难达成的。
3. 该类对应的java.lang.Class对象没有在任何地方被引用，无法在任何地方通过反射访问该类的方法。

tlab

对象是不是一定分配在堆里面，什么是逃逸分析？

垃圾标记的时机

jvm运行时，虚拟机栈，方法区，堆三块区域之间的数据联系可以讲一下吗？

