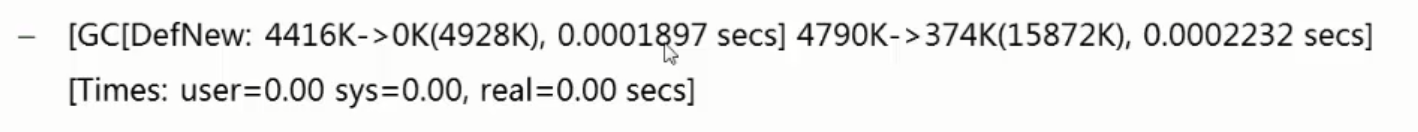
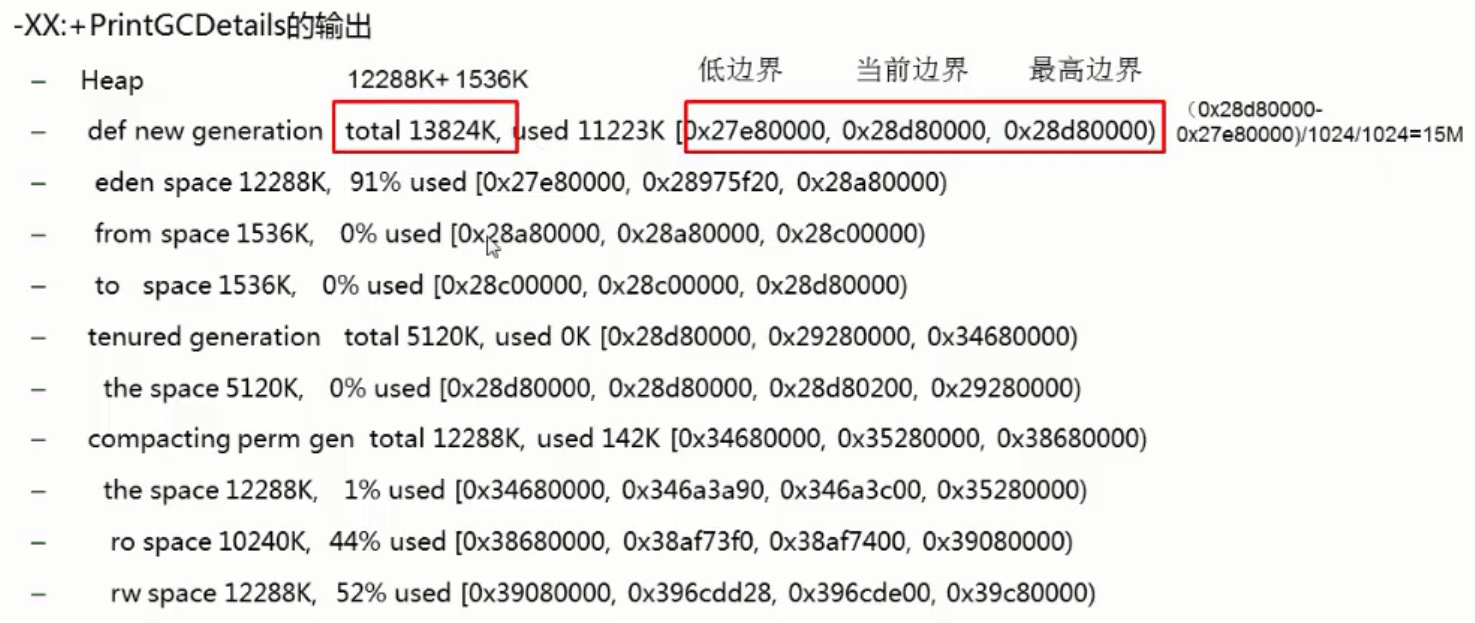
# 堆参数

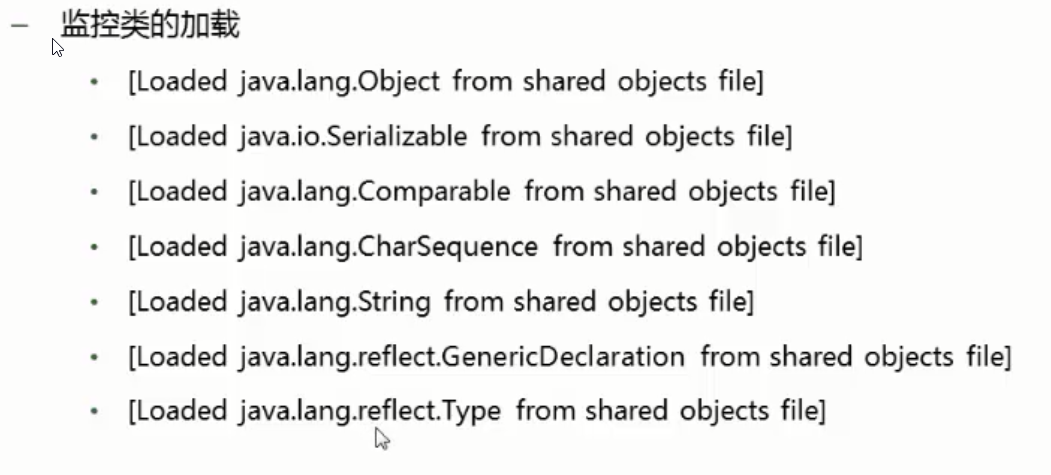
1. 参数 -XX:+PrintGCDetails 可打印详细的堆信息，在程序运行结束后打印堆的整个运行状况：

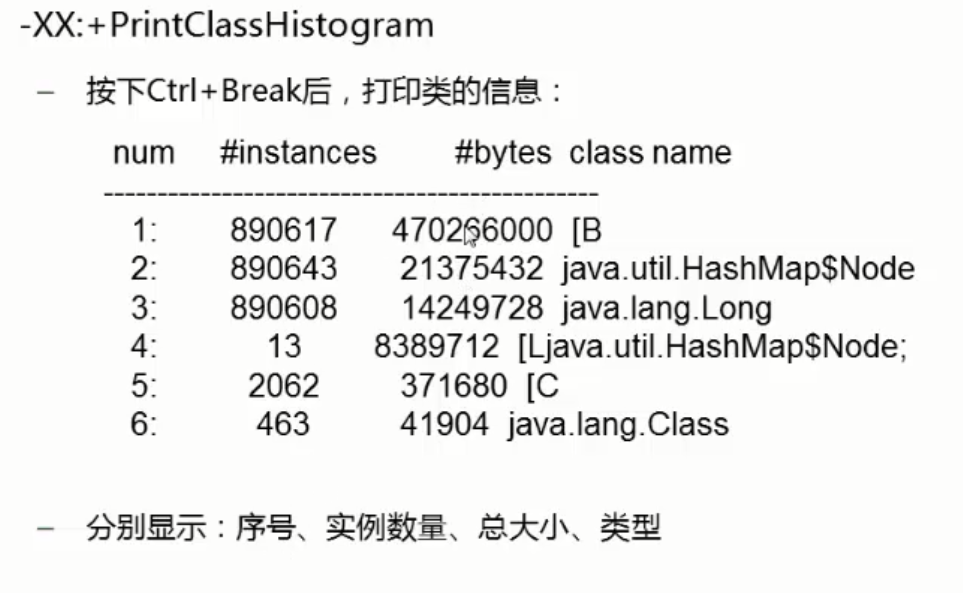


表示新生代空间在gc前占用4416K，在gc后占用0k，总共4928K



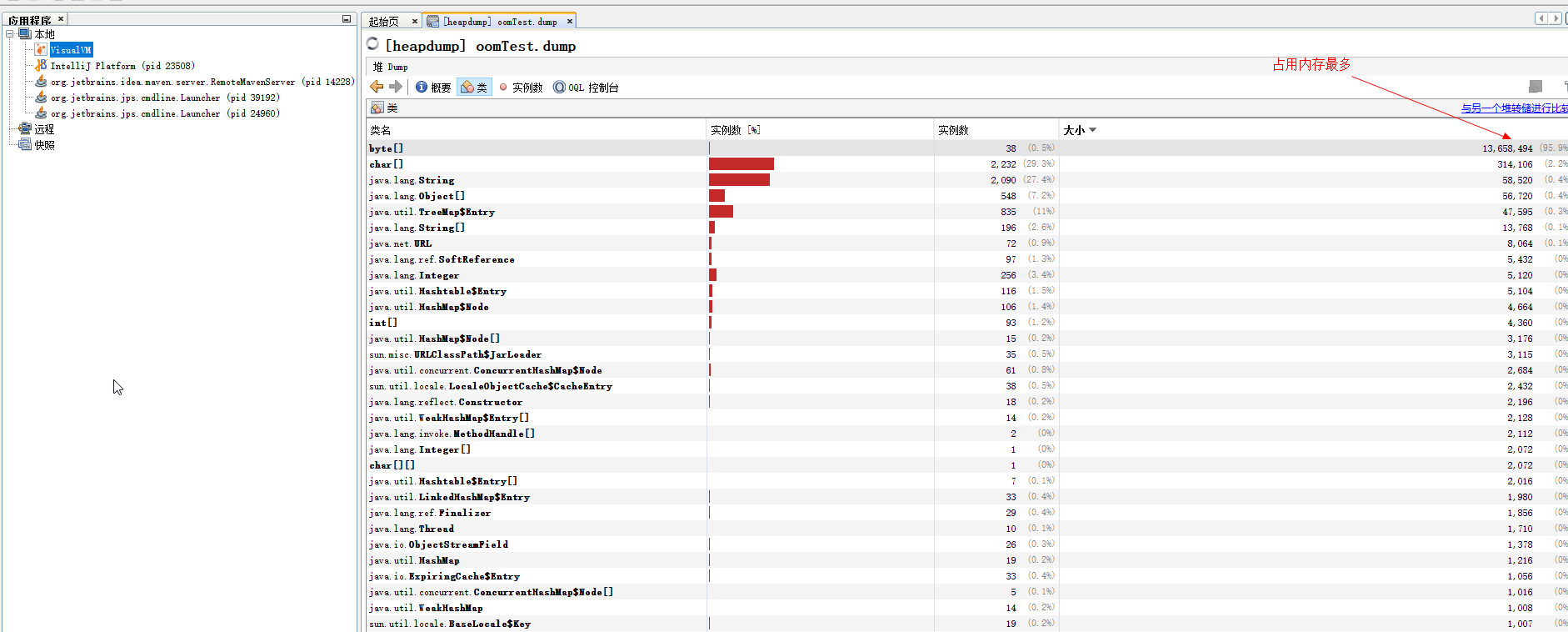
1. -Xloggc:log/gc.log 重定向gc的log日志位置，以文件输出
2. -XX:+PrintHeapAtGC 每次gc都打印gc前后的堆信息
3. -XX:+TraceClassLoading 监控类的加载



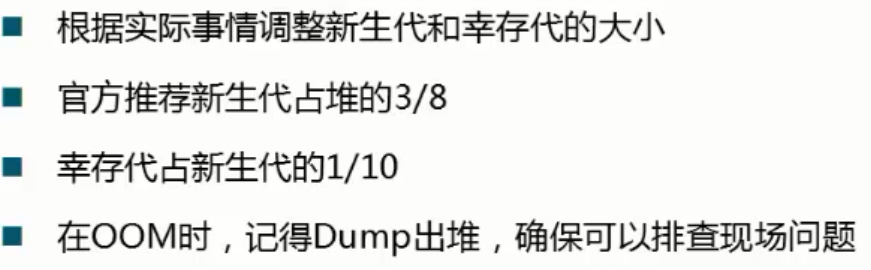
1. 
2. 
3. -Xmx20m -Xms5m -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=./jvm/oomTest.dump

这个参数是指发生内存溢出时，把堆内存转储成文件并放入具体路径的文件中。

使用java自带的jvisualvm工具，可以分析出内存情况



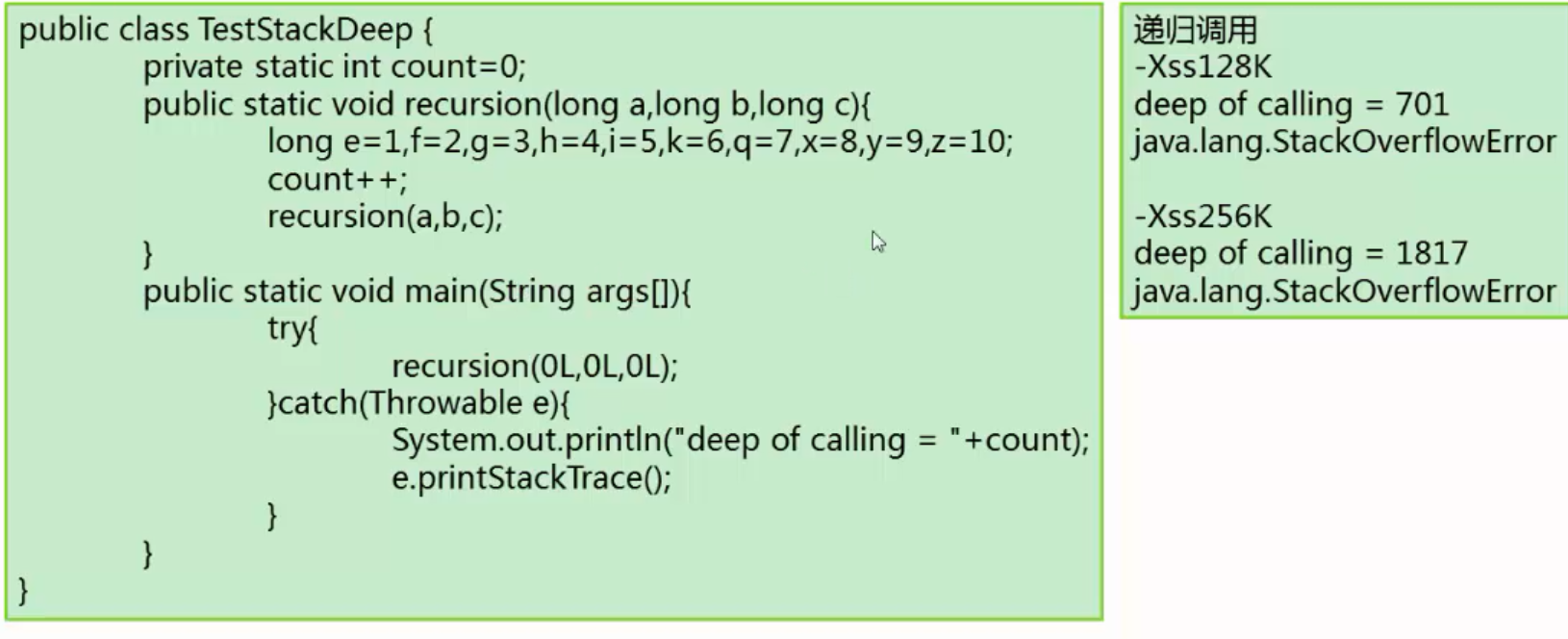
1. 堆内存分配建议



1. 栈空间内存也可以分配 如果栈空间太小 递归调用中可能会出现栈溢出

-Xss128K 设置栈内存 栈内存是线程私有的

deep of calling=701 栈深度 701 表示被调用了701次



减少调用次数和方法内的变量，可以提高调用深度。

1. 持久代（元空间），其实就是JVM规范中方法区的实现。
2. 认为最可能的启动JVM参数：

-Xms25m -Xmx40m -Xmn7m -XX:+PrintGCDetails -XX:PermSize=16m

首先 def new generation total 6464K, used 115K [0x34e80000, 0x35580000, 0x35580000)

eden space 5760K, 2% used [0x34e80000, 0x34e9cd38, 0x35420000)

from space 704K, 0% used [0x354d0000, 0x354d0000, 0x35580000)

to space 704K, 0% used [0x35420000, 0x35420000, 0x354d0000)

通过这一行可以知道年轻代大小是7m.

通过 tenured generation total 18124K, used 8277K [0x35580000, 0x36733000, 0x37680000)

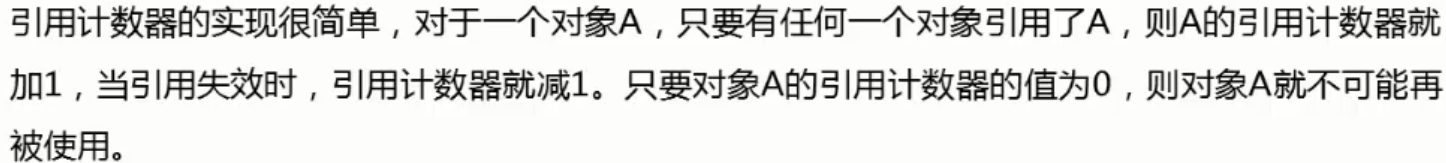
（0x37680000-0x35580000）/1024/1024得到的结果是33m

通过以上可以得到最大堆是40m。但通过eden大小和 tenured generation total 18124K计算出最小堆应该是25m

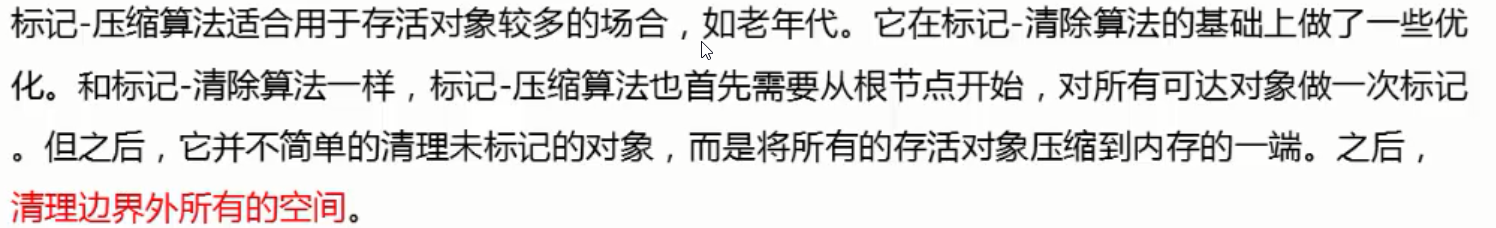
通过compacting perm gen total 16384K, 可以计算出持久堆-XX:PermSize=16m

# GC算法

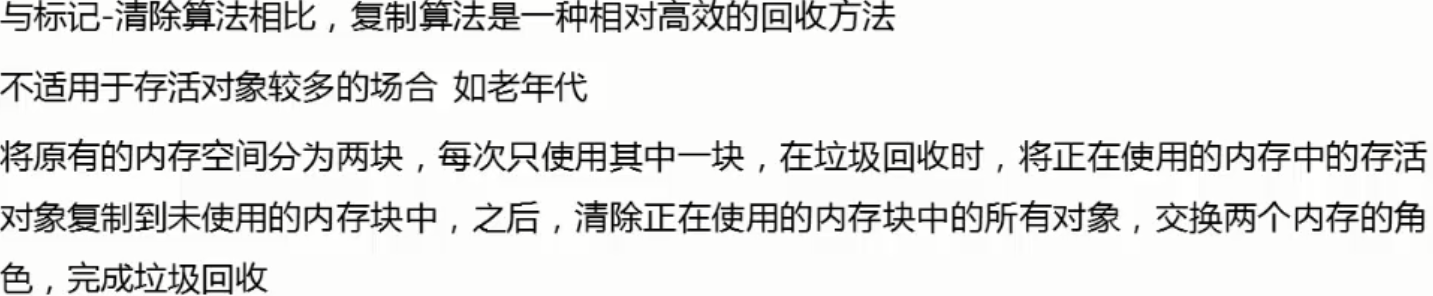
### 引用计数法



### 标记-压缩算法

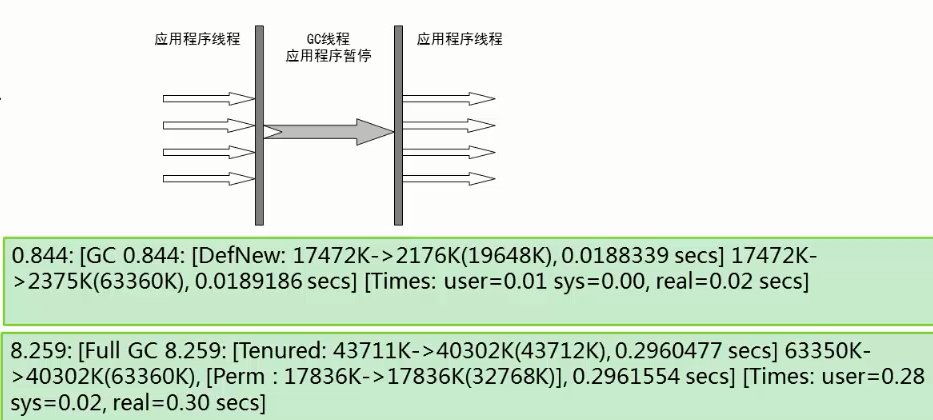


### 复制算法

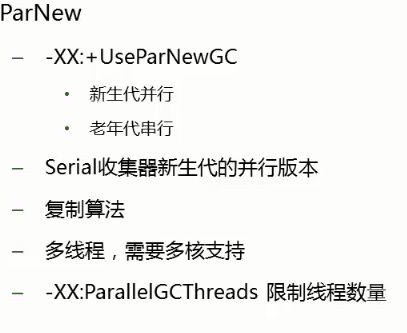


# 垃圾收集器

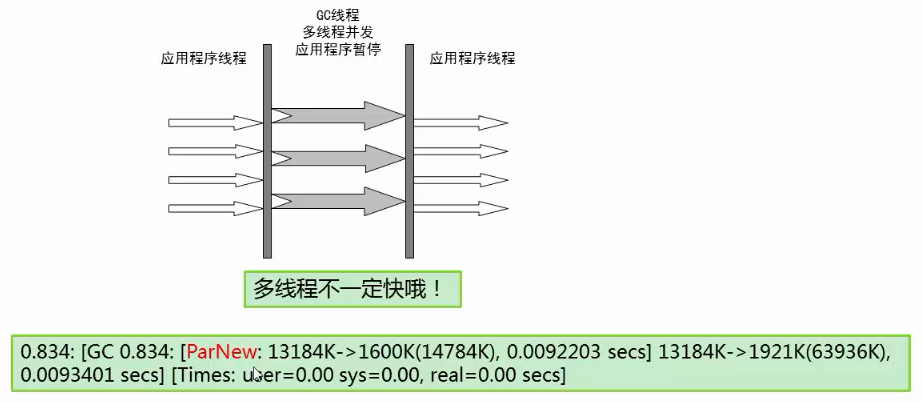
### 串行收集器



### 并行收集器



使用这个参数后，新生代使用并行回收，老年代依然都是用串行回收



# 类装载验证流程

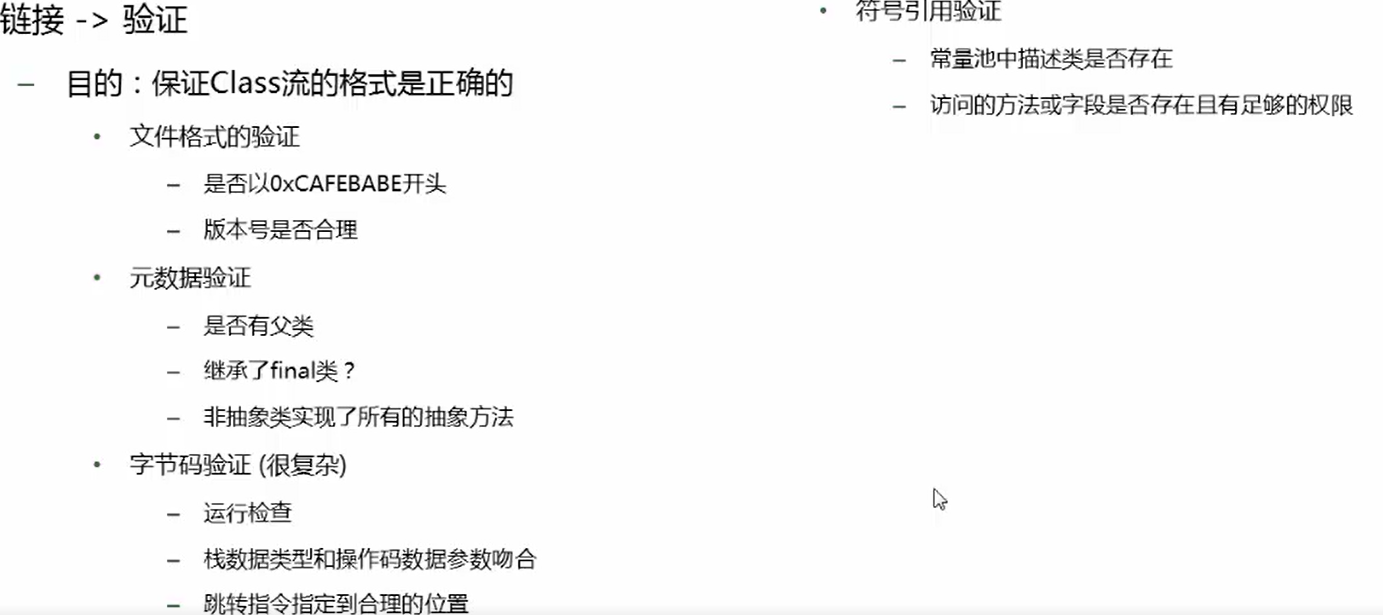
## 加载

加载类文件、网络类，把类转换成二进制流

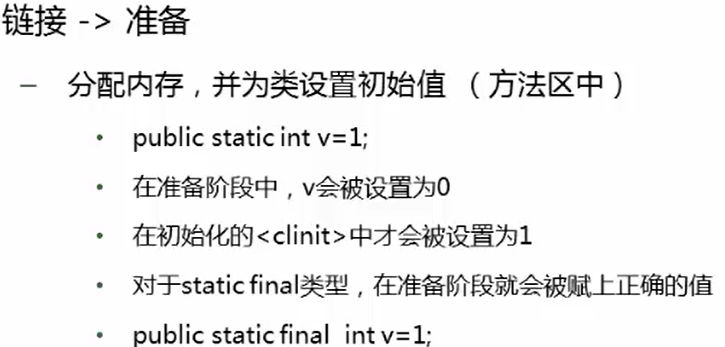
## 链接

### 验证

保证Class二进制流的格式正确，检查文件格式、原数据



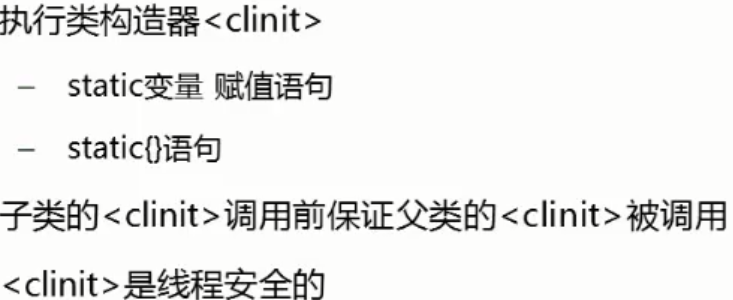
### 准备



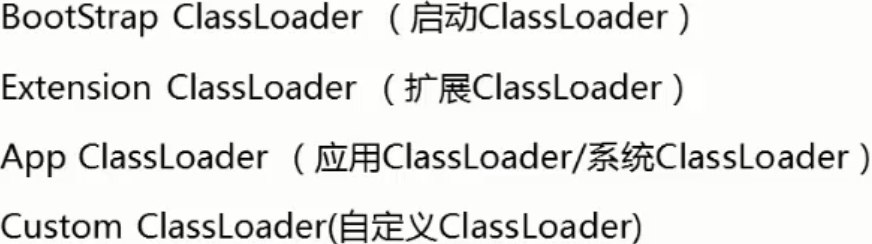
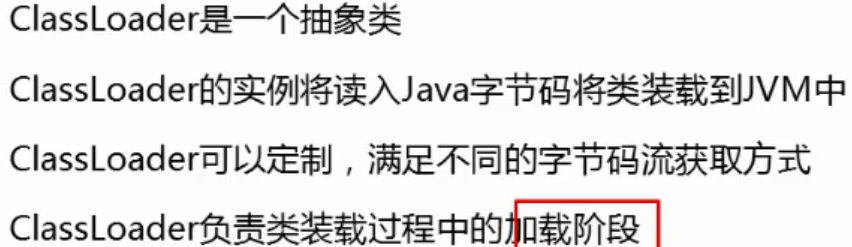
### 解析

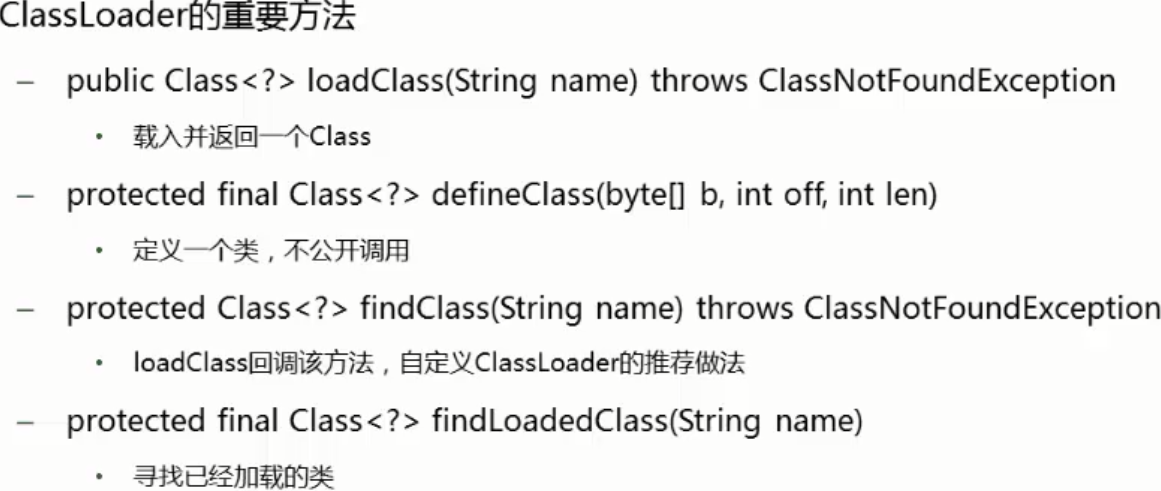
符号引用替换为直接引用

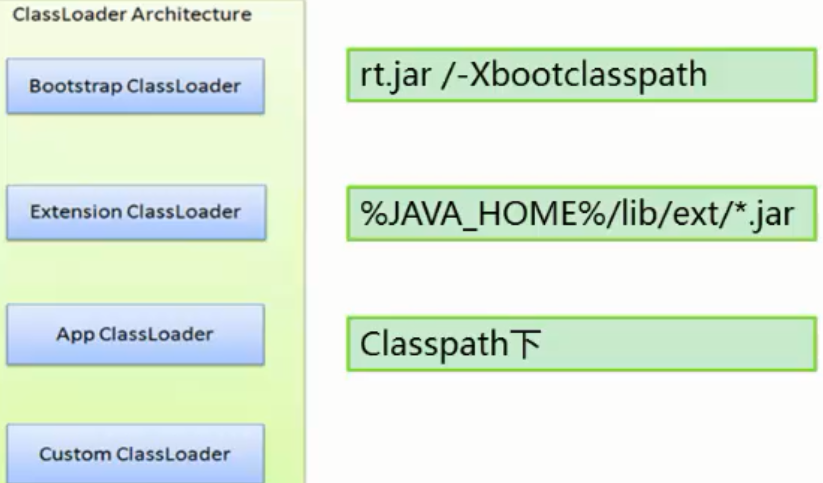
### 初始化



# 类装载器







1. Bootstrap ClassLoader 加载rt.jar包下的类，也就是jdk里面的源码包。
2. Extension ClassLoader 加载java拓展包的类。
3. App ClassLoader加载Classpath下的类，即是用户编写的应用程序。