Java Virtual Machine
——自动内存管理

JVM体系


JVM体系结构
自动内存管理机制
Java内存区域与内存溢出异常
垃圾收集器与内存分配策略
虚拟机性能监控与故障处理工具
虚拟机执行子系统
类文件结构
虚拟机类加载机制
虚拟机字节码执行引擎
程序编译与代码优化
编译期优化
运行期优化
高效并发
Java内存模型与线程
线程安全与锁优化

JVM的内心独白

我该怎样划分内存区域，才能更
好地为不同对象提供服务？

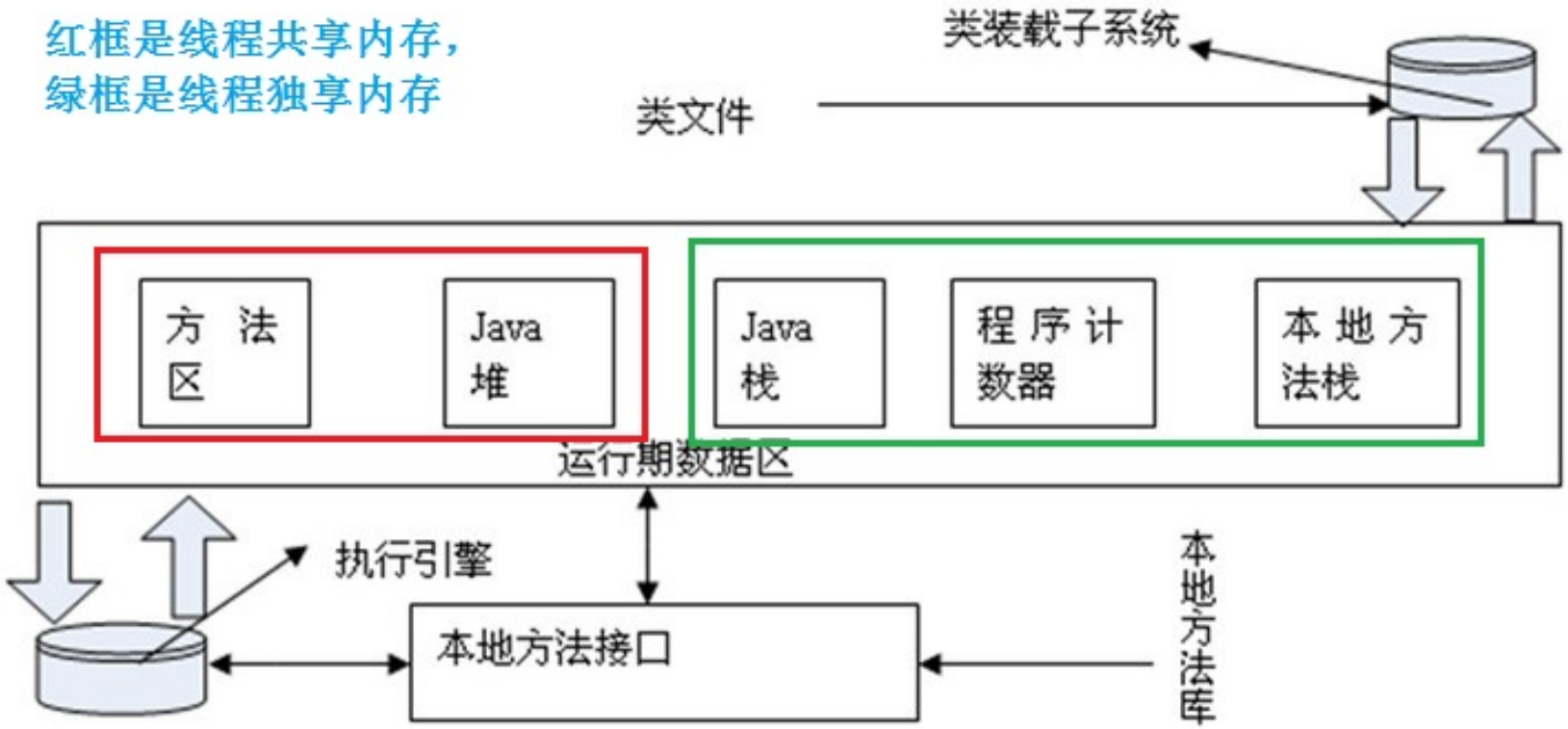
给对象分配内存的时候
1、如何解决内存碎片？
2、如何解决多线程并发争抢内存的情况？
3、对象在内存中应该保存什么数据？

内存回收时：
1、什么区域可以回收内存？
2、内存回收的时机是什么？
3、怎样执行内存回收？

目录

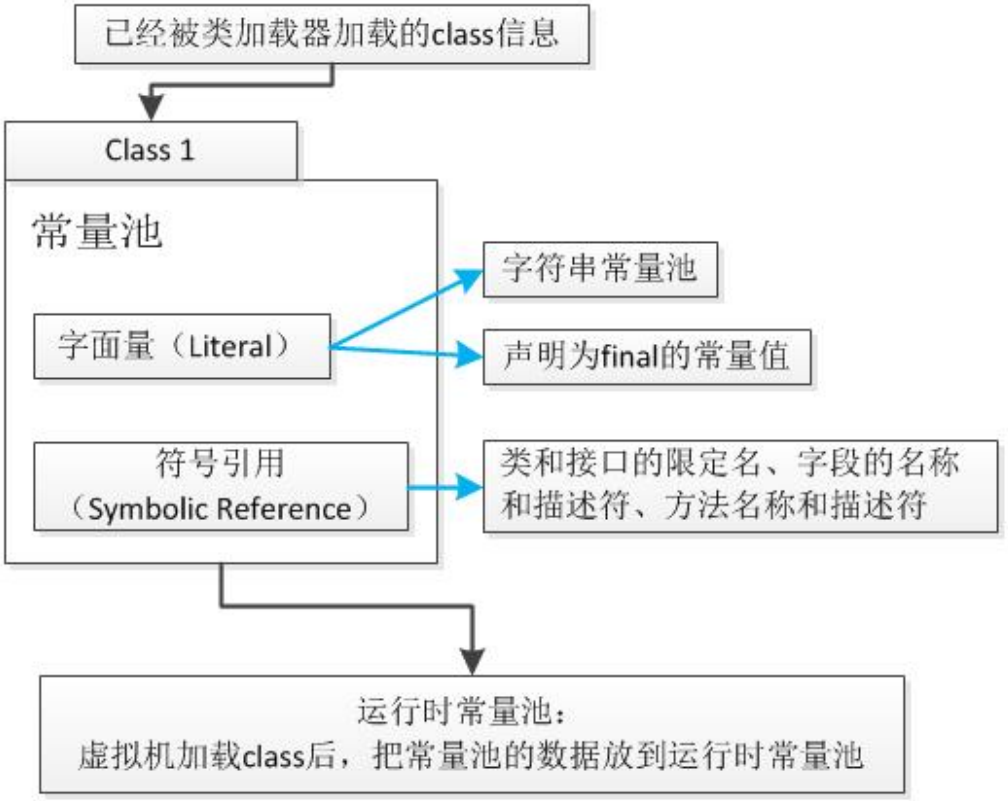
- JVM内存布局
- 对象内存分配
- 不同内存区域的异常
- 内存回收策略
- JVM性能监控与故障处理工具
- 实战

JVM内存布局


红框是线程共享内存，
绿框是线程独享内存
类装载子系统
类文件
方法区
Java堆
Java栈
程序计数器
本地方法栈
运行期数据区
执行引擎
本地方法接口
本地方法库

方法区内存布局

JVM方法区：常量池+运行时常量池



目录

- JVM内存布局
- 对象内存分配
- 不同内存区域的异常
- 内存回收策略
- JVM性能监控与故障处理工具
- 实战

对象内存分配

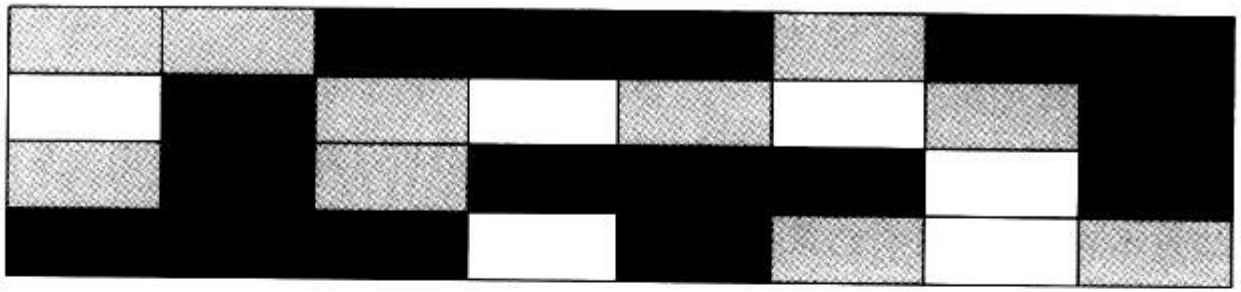
在常量池没有Class实例，或者Class没有被加载、解析、初始化

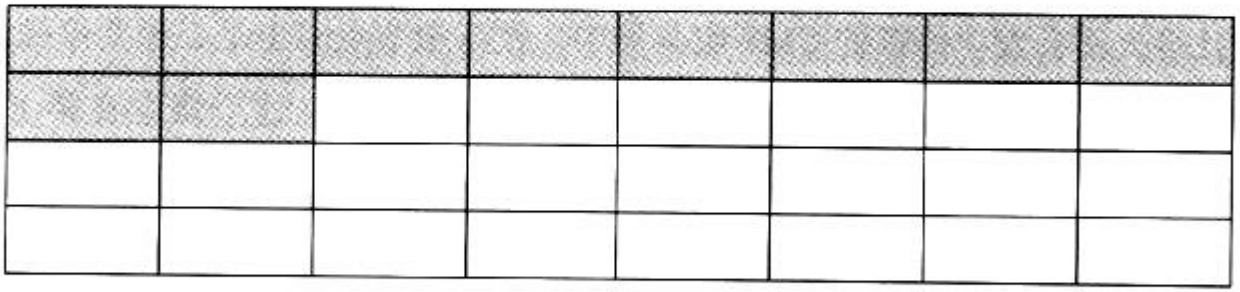
准备创建Class对象

否

执行Class的JVM加载过程

不同的GC收集器，垃圾回收后，内存的规整情况不同





寻找空闲内存

解决内存分配过程中，多

1、同步加锁，会导致阻塞
2、虚拟机采用的是CAS + 失

式

和空闲
处理，
子性

字

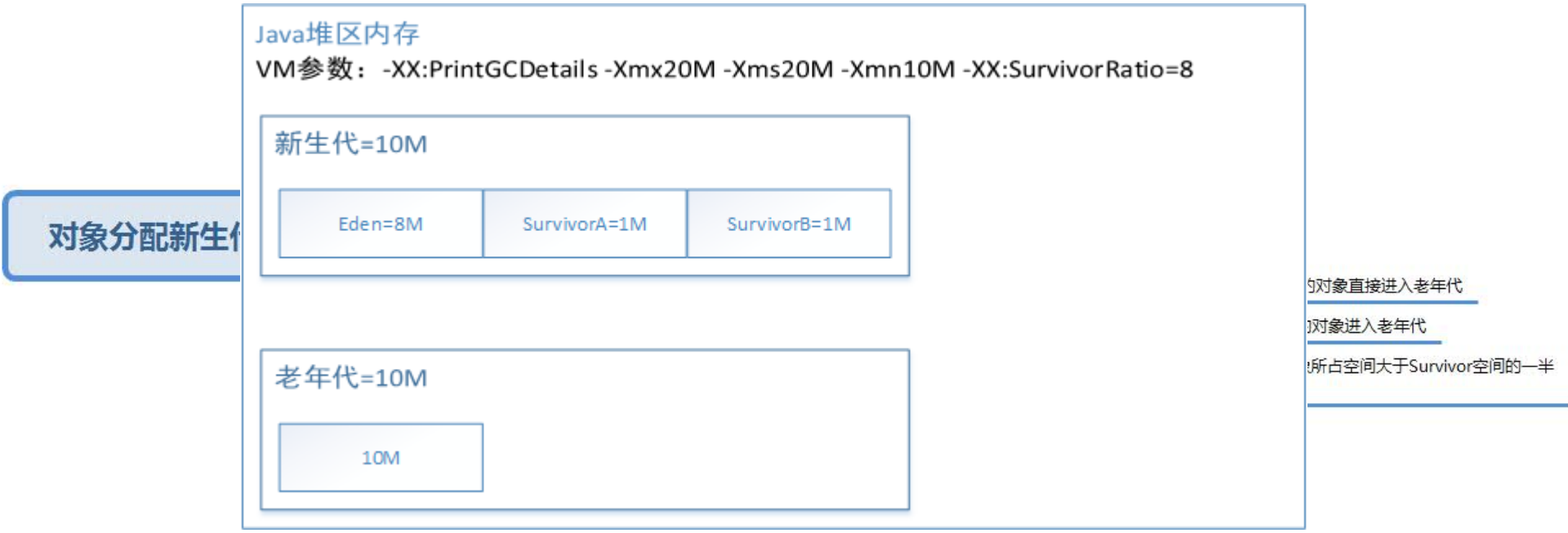
3、对象内存对齐填充为8byte的整数倍，方便HotSpot寻址

里程碑：JVM认为对象创建完成

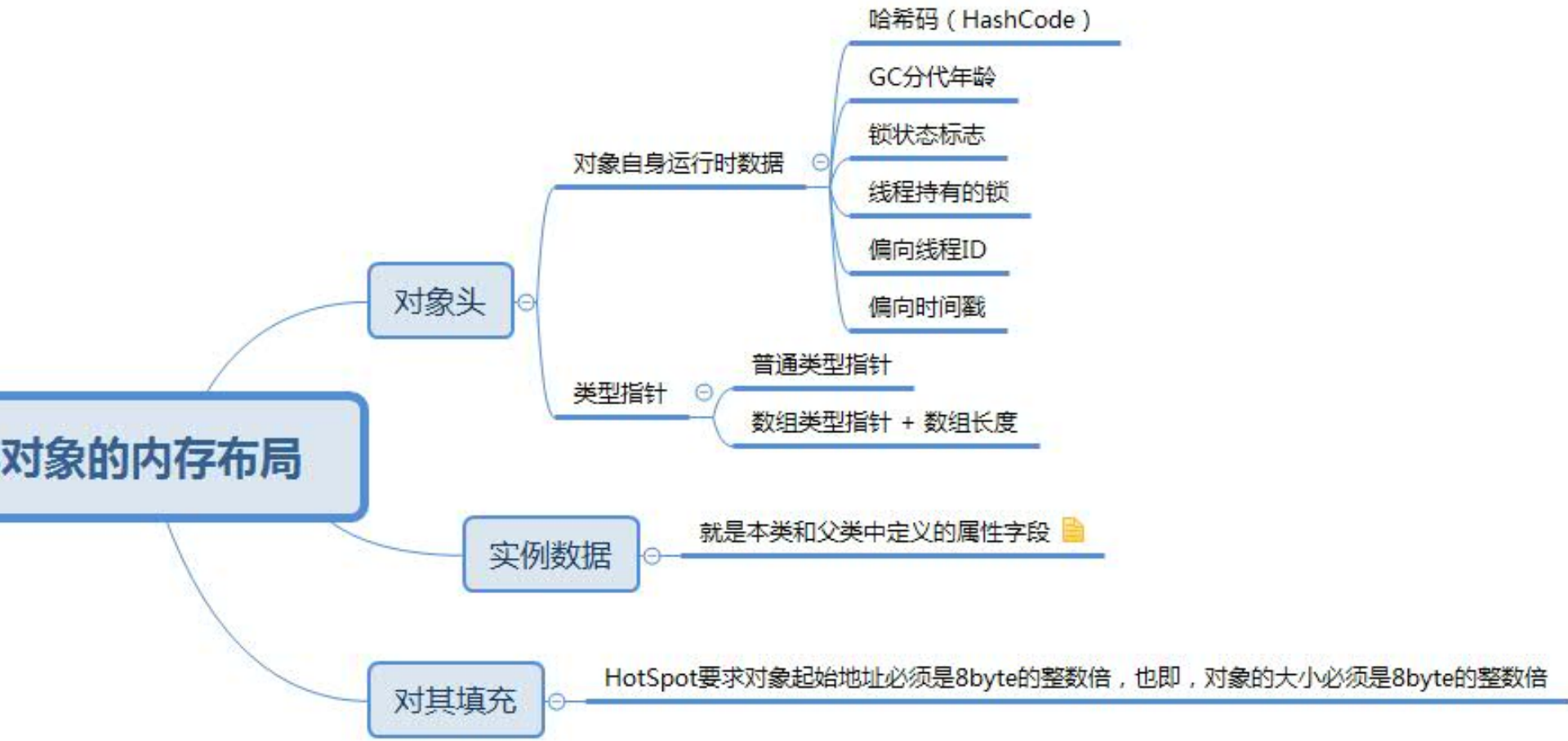
根据构造函数，对属性再次赋值

里程碑：程序员认为对象创建完成

空间分配担保机制


对象分配新生
Java堆区内存
VM参数：-XX:PrintGCDetails -Xmx20M -Xms20M -Xmn10M -XX:SurvivorRatio=8
新生代=10M
Eden=8M
SurvivorA=1M
SurvivorB=1M
老年代=10M
10M
对象直接进入老年代
对象进入老年代
所占空间大于Survivor空间的一半

对象内存布局


对象的内存布局
对象头
对象自身运行时数据
哈希码（HashCode）
GC分代年龄
锁状态标志
线程持有的锁
偏向线程ID
偏向时间戳
类型指针
普通类型指针
数组类型指针 + 数组长度
实例数据
就是本类和父类中定义的属性字段
对其填充
HotSpot要求对象起始地址必须是8byte的整数倍，也即，对象的大小必须是8byte的整数倍

对象访问定位

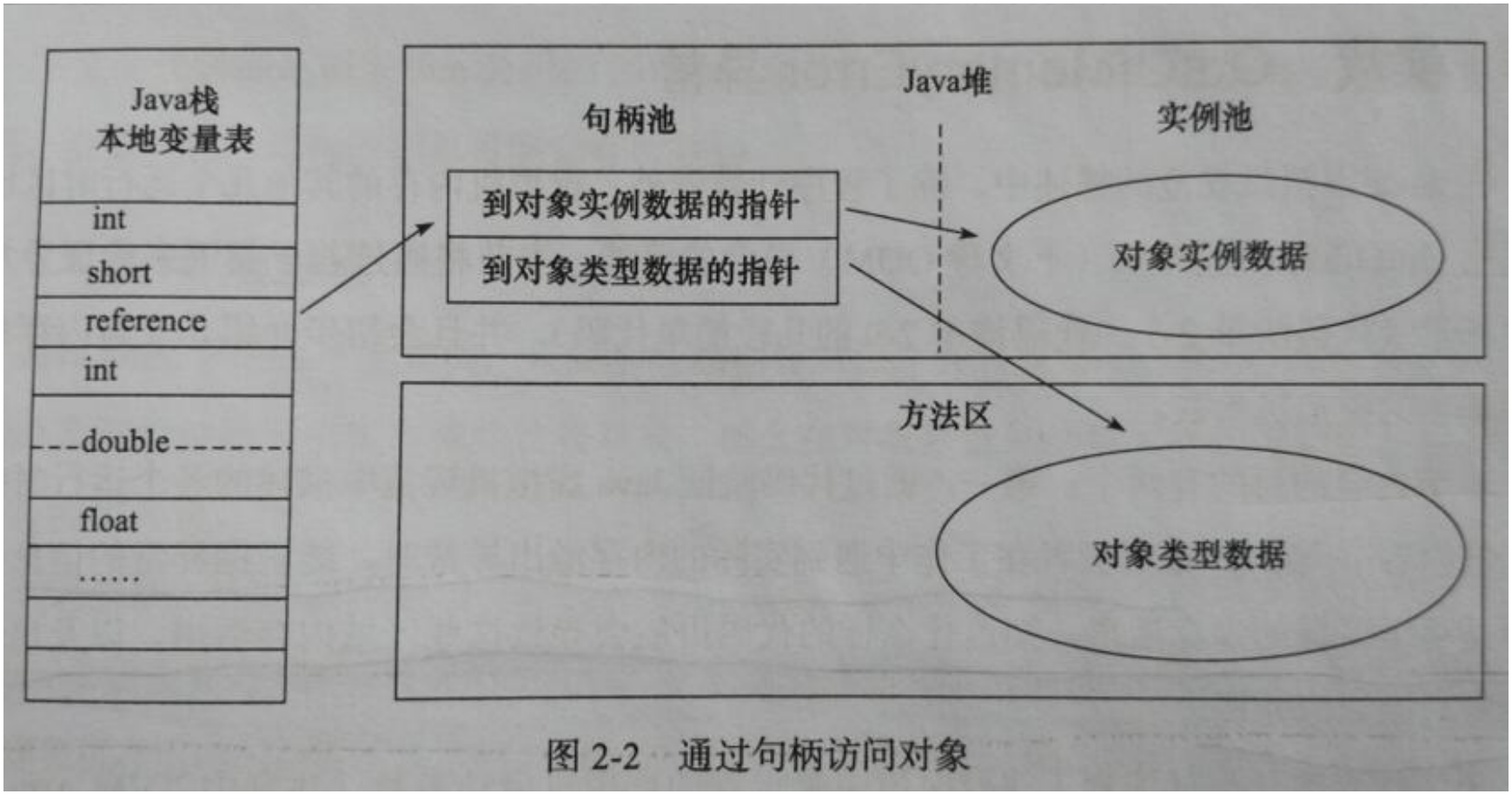


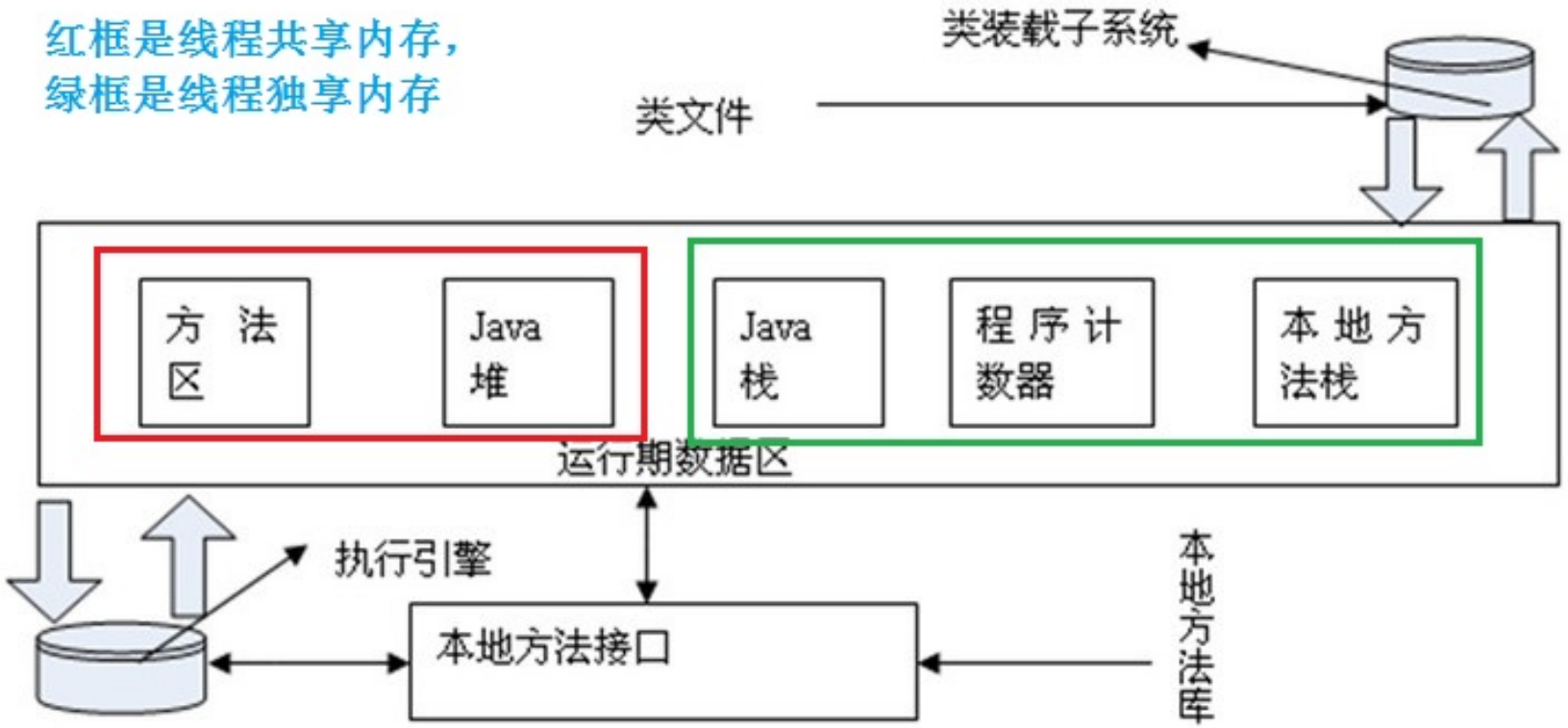
图 2-2　通过句柄访问对象

目录

- JVM内存布局
- 对象内存分配
- 不同内存区域的异常
- 内存回收策略
- JVM性能监控与故障处理工具
- 实战

不同内存区域的异常

-Xmx -Xms



程序计数器

唯一一块没有OutOfMemoryError的内存

HeapOutOfMemoryErrorClient

目录

- JVM内存布局
- 对象内存分配
- 不同内存区域的异常
- 内存回收策略
- JVM性能监控与故障处理工具
- 实战

内存回收策略

- 什么区域可以GC?
- 什么时候该去GC?
- 怎样GC?

内存回收策略

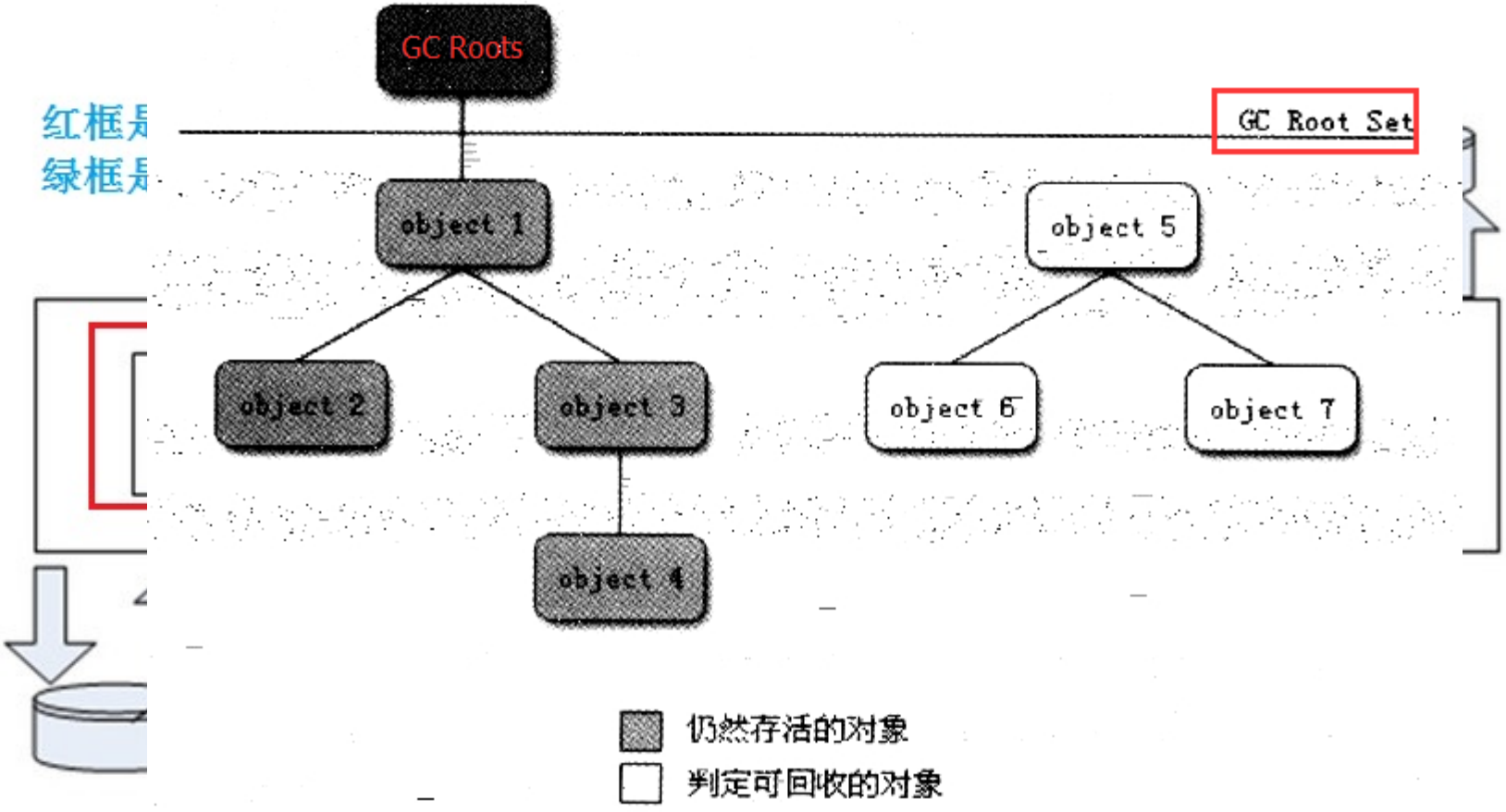


图 3-1 可达性分析算法判定对象是否可回收


新生代
标记-清除算法
标记-整理算法
老年代
复制算法（需要分配担保空间）
HotSpot
分代收集算法，就是根据新生代和老年代采用不同的垃圾收集算法
枚举根结点
出发点：为了快速获取GC Roots，不用每次GC前去扫描GC Roots，采用OopMap记录对象引
无法解决：GC过程需要Stop The World，停止一切用户线程
出发点：解决了GC过程，运行中的线程在何处stop的问题
无法解决：始终sleep或blocked的线程，GC的问题
主动式中断
设置中断标志位，线程检测到这个中断标志位后，就自动中断。HotSpot采用
抢先式中断
先中断线程，若有线程没有停在安全点，则resume，让该线程跑到安全点再sto
安全区域
出发点：解决始终sleep或blocked的线程，GC的问题
Serial
单线程，复制算法，默认jvm -client模式的收集器
ParNew
Serial的多线程版本，默认jvm -server模式的收集器
Parallel Scavenge
吞吐量优先收集器，多线程，复制算法，适合后台程序，不适合用户交互程序
G1
用于替换CMS收集器
Serial Old（MSC）
单线程，标记-整理算法
Parallel Old
吞吐量优先收集器，多线程，标记-整理算法，适合后台程序，不适合用户交互程序
目标是获取最短停顿时间
多线程，标记-清除算法
真正意义的并发收集器，在GC线程运行时，并不会停止用户线程，但CPU资源有限，系统只
慢，但至少还有响应。GC线程和用户线程同时运行，因此会产生"浮动垃圾"

安全点
- 主动式中断 ⊖ 设置中断标志位，线程检测到这个中断标志位后，就自动中断。HotSpot采用
- 抢先式中断 ⊖ 先中断线程，若有线程没有停在安全点，则resume，让该线程跑到安全点再stop

安全区域 ⊖ 出发点：解决始终sleep或blocked的线程，GC的问题

新生代
- Serial ⊖ 单线程，复制算法，默认jvm -client模式的收集器
- ParNew ⊖ Serial的多线程版本，默认jvm -server模式的收集器
- Parallel Scavenge ⊖ 吞吐量优先收集器，多线程，复制算法，适合后台程序，不适合用户交互程序
- G1 ⊖ 用于替换CMS收集器

Serial Old（MSC） ⊖ 单线程，标记-整理算法

Parallel Old ⊖ 吞吐量优先收集器，多线程，标记-整理算法，适合后台程序，不适合用户交互程序

集器

图 3-6 Serial / Serial Old 收集器运行示意图

总结
- 力求系统最短停顿时间，适用于用户交互程序 ⊖
  - 老年代 ⊖
    - G1
    - Serial Old
    - G1
- 力求系统最大吞吐量，适用于后台计算程序 ⊖
  - 新生代 ⊖ Parallel Scavenge
  - 老年代 ⊖ Parallel Old

垃圾收集算法

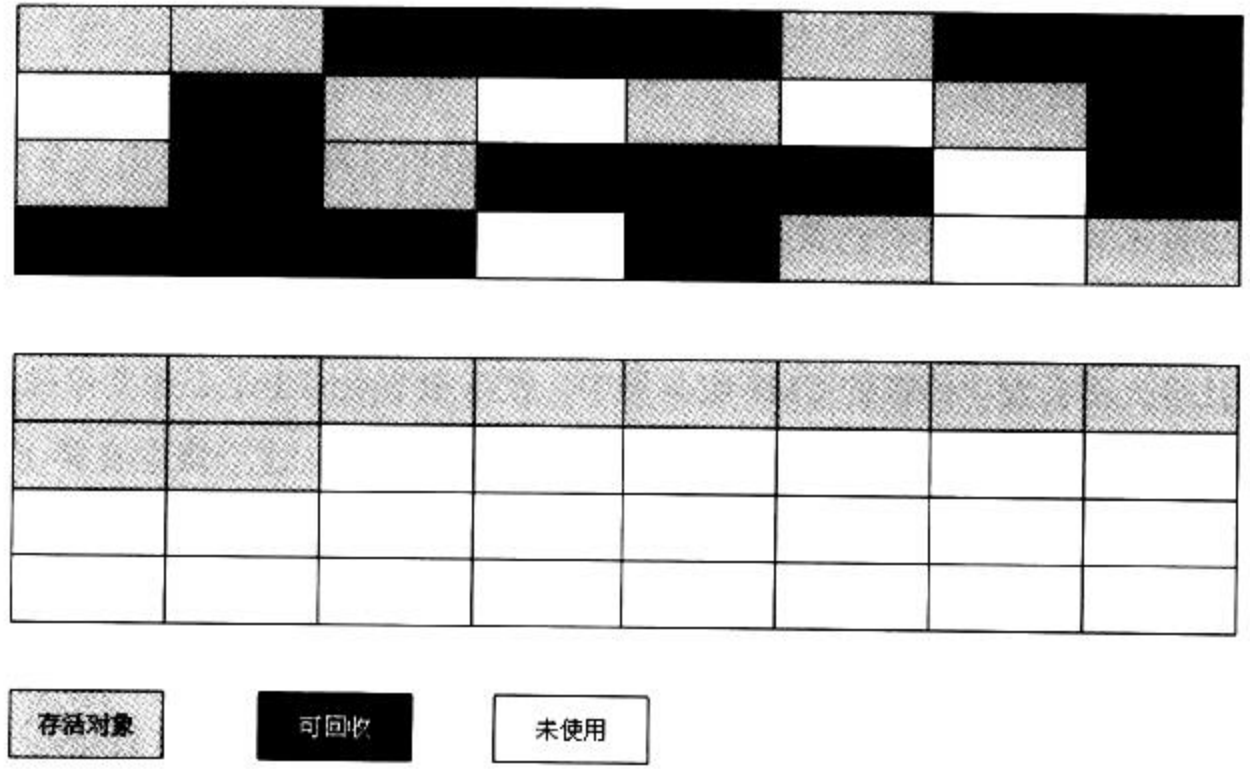
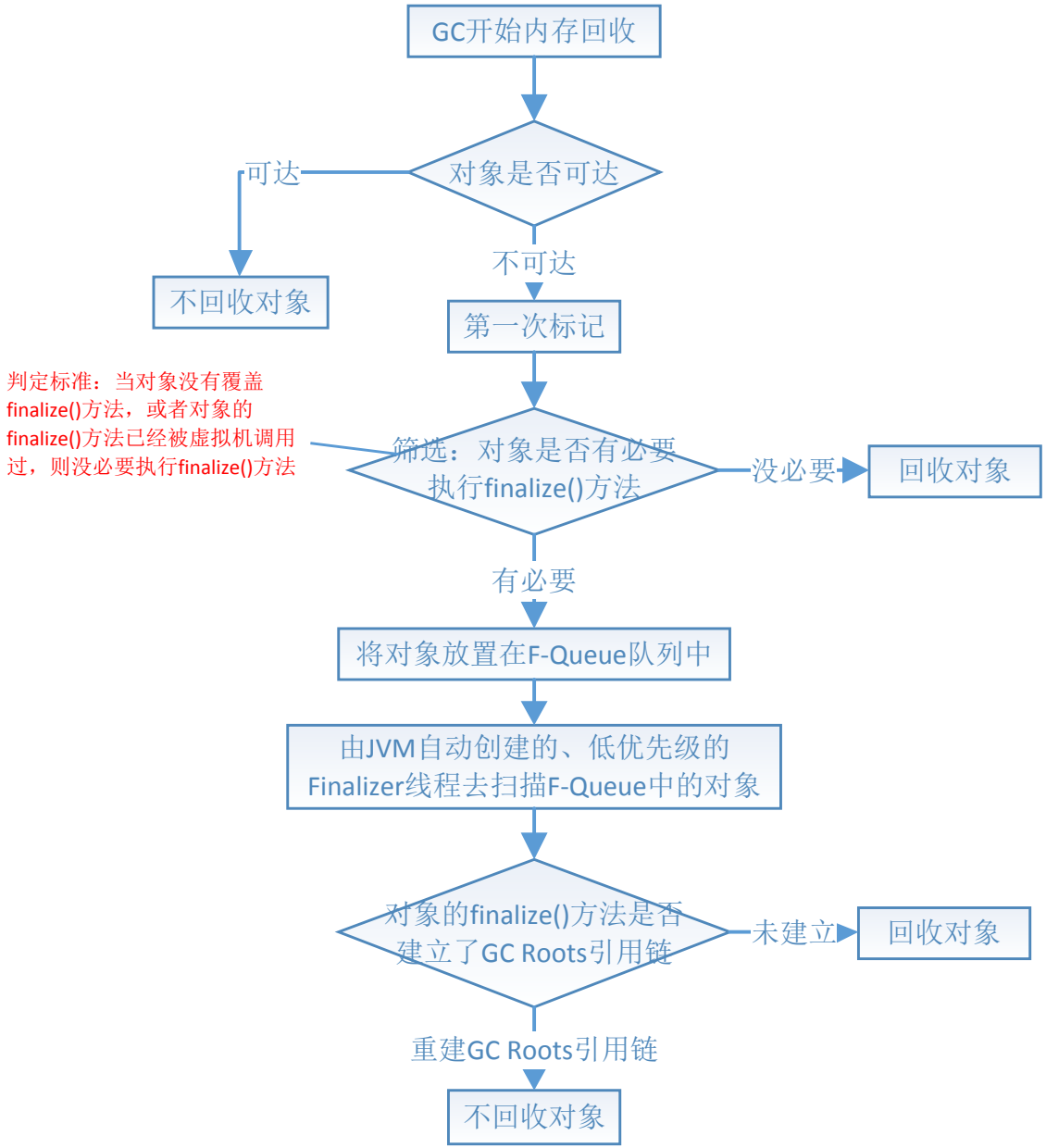


图 3-4 “标记 – 整理”算法示意图

图 3-3 “复制算法示意图

图 3-2 “标记 – 清除”算法示意图

GC Roots不可达对象的内存回收流程



目录

- JVM内存布局
- 对象内存分配
- 不同内存区域的异常
- 内存回收策略
- JVM性能监控与故障处理工具
- 实战

JVM性能监控与故障处理工具

- jps：JVM进程状况工具
- jinfo：JVM配置信息工具
- jstat： JVM运行信息监视工具
- jstack：Java线程堆栈跟踪工具
- jmap：Java内存映像工具，dump文件
- jhat：用于分析dump文件的工具

练习

JVM性能监控与故障处理工具

- Jconsole软件：集成了上述所有指令，用于监控JVM性能
- VisualVM：可扩展的Jconsole，以插件形式提供服务

1、JMX+Mbean动态修改线程bean中的属性值

2、Btrace：动态修改线上class的代码，相当于动态代理，可以用于线上debug日志打印，即时生效，不用单独上线

目录

- JVM内存布局
- 对象内存分配
- 不同内存区域的异常
- 内存回收策略
- JVM性能监控与故障处理工具
- 实战

实战

- scf项目无法启动问题排查
- web项目在windows+jetty环境正常运行，但在linux+tomcat运行异常，NoClassDefFoundError + NoSuchMethodError
- JMX+Mbean：动态修改javaBean属性
- Btrace：动态代理，免上线的线上debug输出