# 一、查看GC Roots

## 1.jmap生成转储文件

jmap -dump:format=b,live,file=1.bin 1100

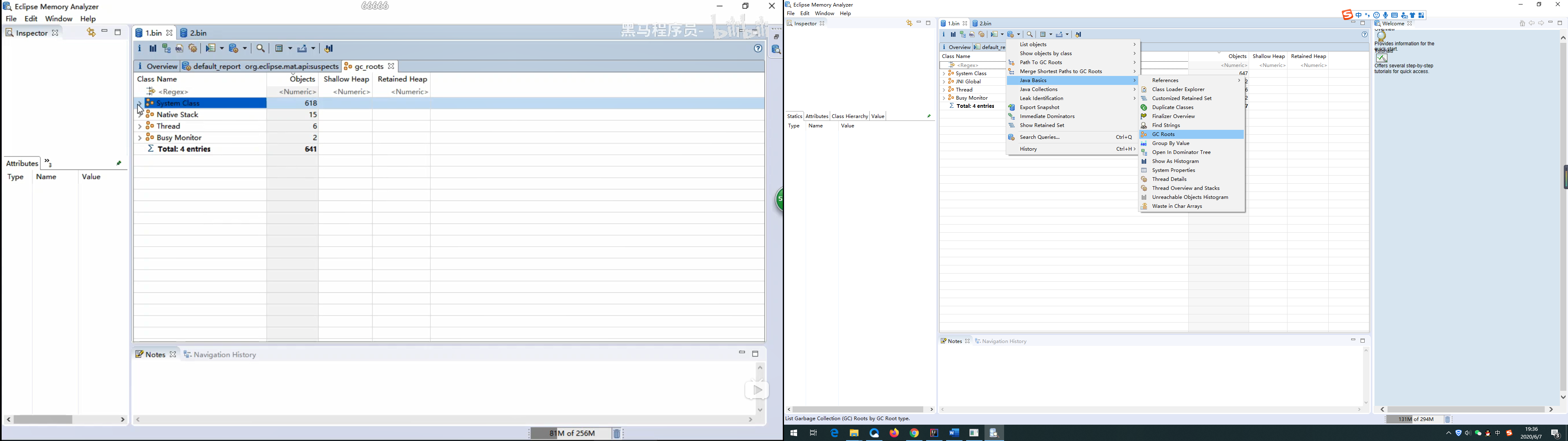
-dump:format=b表示转储文件成二进制文件

live显示存活对象，而且在快照前进行一次垃圾回收

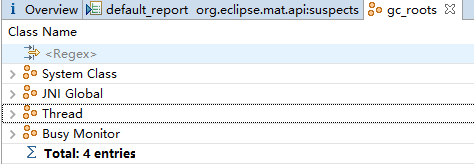
file=1.bin 为文件名

1100为jps查看的进程号

## 2.查看并分析转储文件

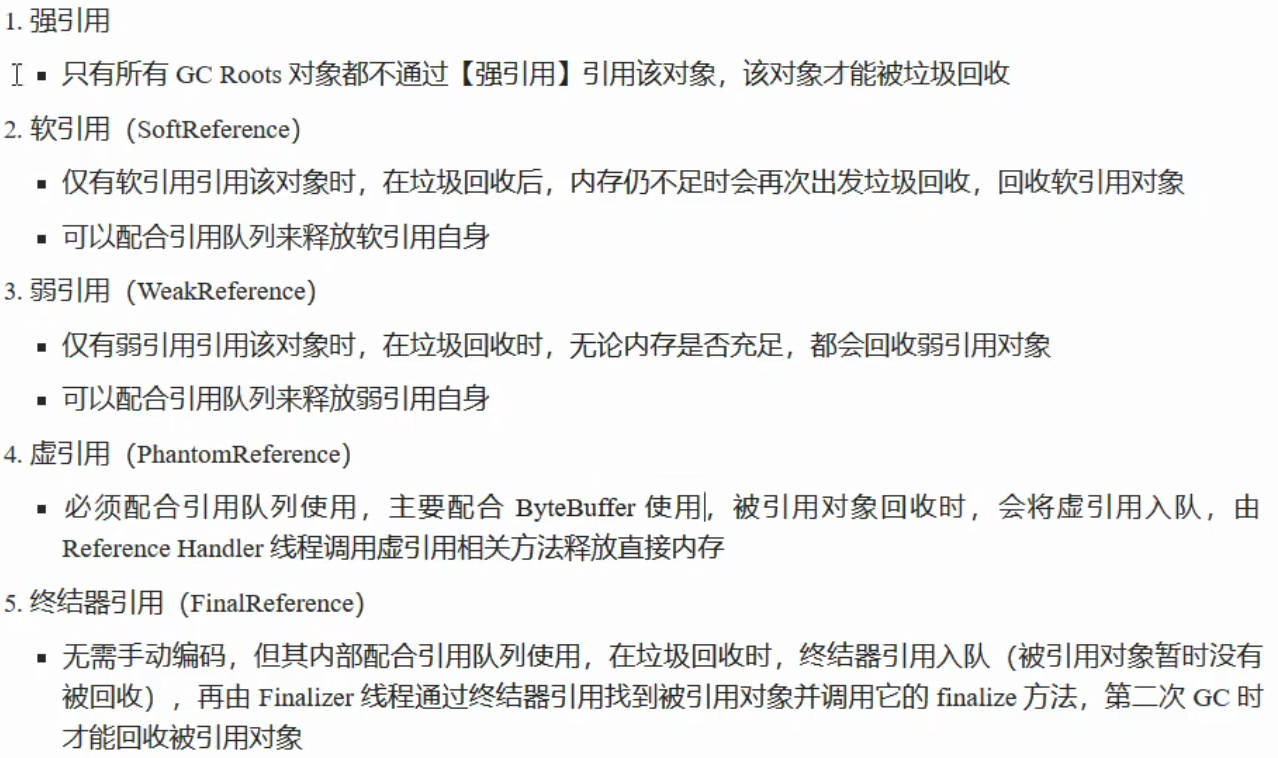


## 3.查看GC Roots



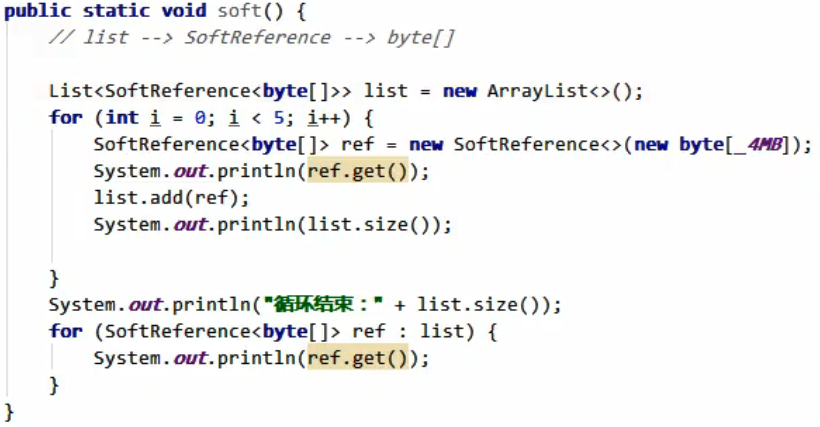
# 二、五种引用的应用

引用自身也是一个对象

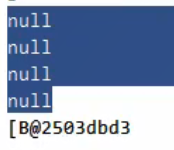


## 1.数据不够时清除软引用

若数据不够了，就清除（软/弱引用使用）

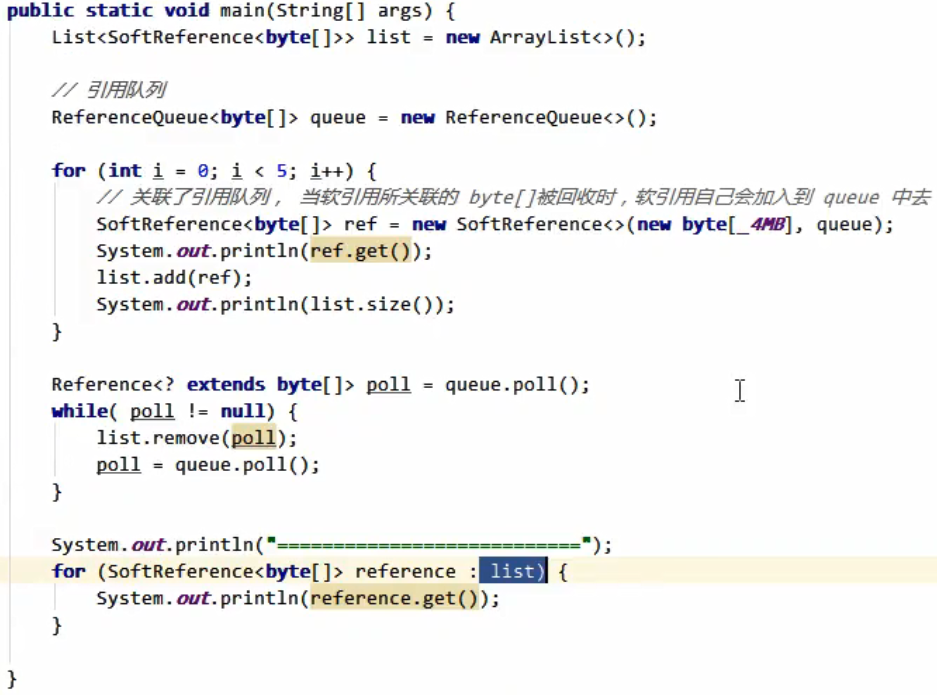


软引用也是对象，所以被清除之后ref.get()返回null，而且list中还保存着ref对象



## 2.配合引用队列使用弱引用

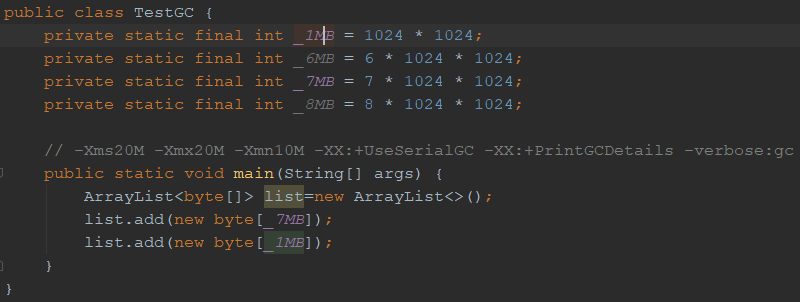
关联了引用队列，当软引用所关联的对象byte[]被回收时，软引用自己会加入到queue中去

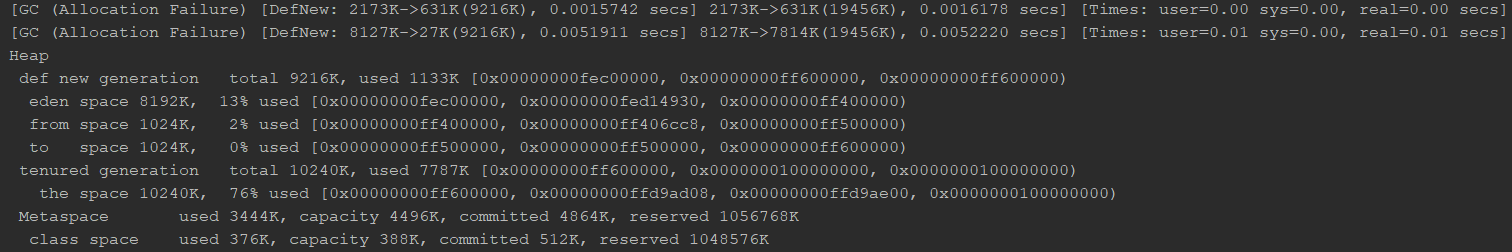


# 三、GC实战

## 1.垃圾回收

-Xms20M -Xmx20M -Xmn10M -XX:+UseSerialGC -XX:+PrintGCDetails -verbose:gc





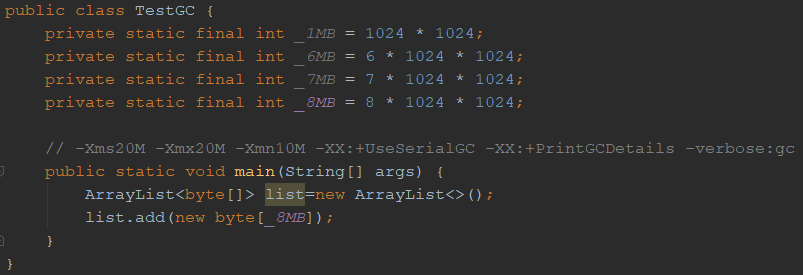
由于虚拟机参数设置堆20MB，新生代10MB，eden和from与to默认8:1:1，所以eden中为9MB。

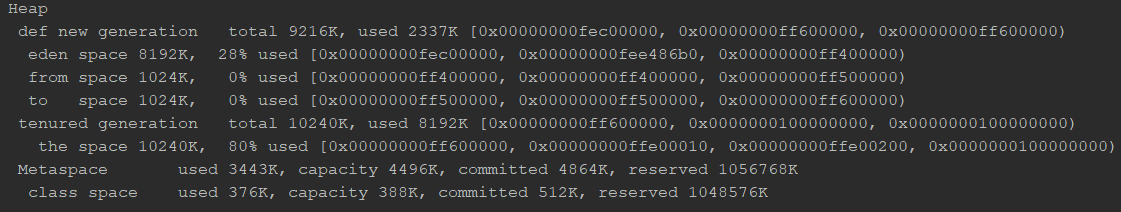
a.一开始虚拟机中会有一些默认空间被占用，当申请一个7MB的byte数组的时候，7mb已经不够了，所以进行了一次垃圾回收；垃圾回收后将7MB放到eden。

b.再申请一个1MB的byte数组时，eden又不够了，所以进行第二次垃圾回收，7MB不能被回收，所以被放到老年代中。

## 2.大对象直接晋升老年代

-Xms20M -Xmx20M -Xmn10M -XX:+UseSerialGC -XX:+PrintGCDetails -verbose:gc

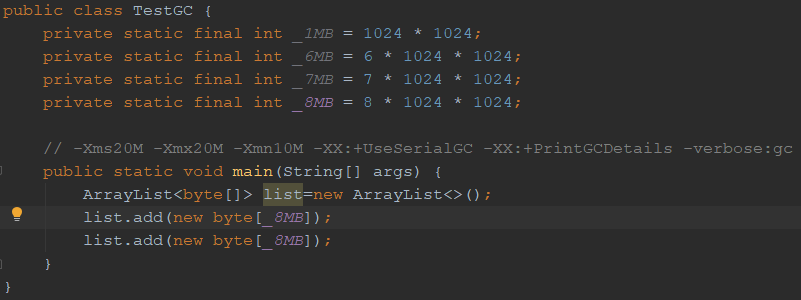


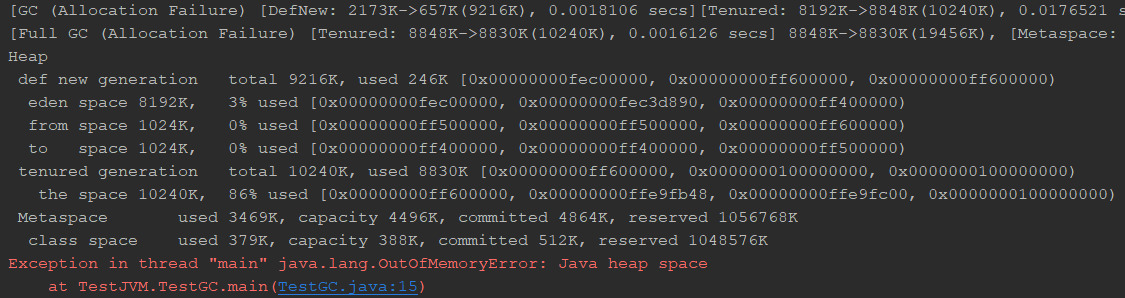


当申请一个对象的时候，这个对象大于新生代的容量了，所以直接被加入到老年代中

## 3.OOM和Full GC

-Xms20M -Xmx20M -Xmn10M -XX:+UseSerialGC -XX:+PrintGCDetails -verbose:gc

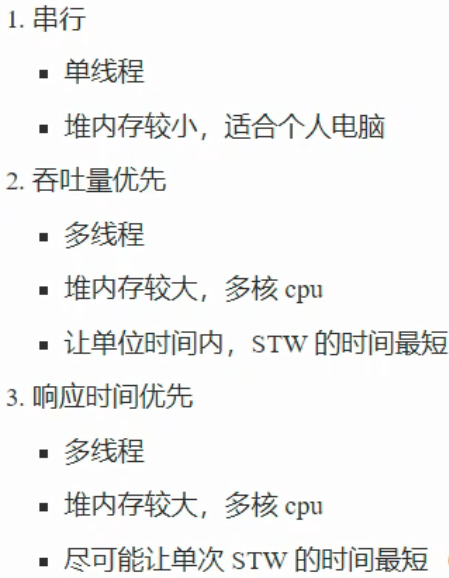




a.先放一个8MB，直接放到了老年代

b.再放一个8MB时，新生代不够，老年代也不够，触发Full GC，Full GC会先触发一次GC，还是不够，然后就报OOM 。

# 四、垃圾回收期分类

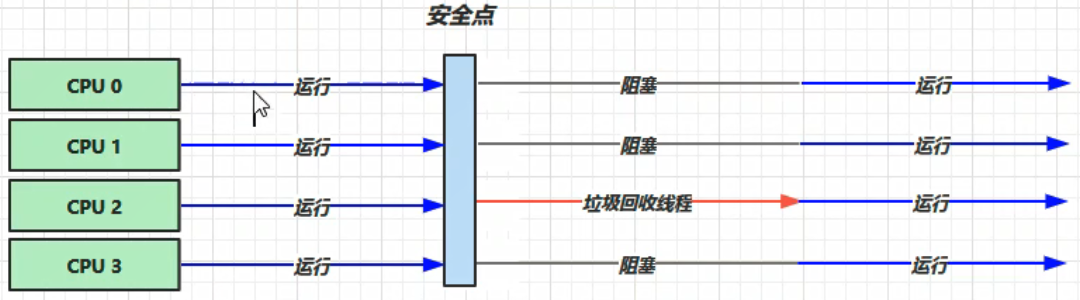


## 1.串行

### （1）虚拟机参数

-XX:+UseSerialGC （Serial(新生代、复制算法)+SerialOld（老年代、标记+整理算法））

### （2）过程



由于垃圾回收的前后对象的内存地址可能会改变，所以需要stop the world

## 2.吞吐量优先

Paralled和ParallelOld

### （1）虚拟机参数

-XX:+UseParalledGC或-XX:+UseParallelOldGC

开启并行垃圾回收（多线程）

-XX:paralledGCThreads=n

垃圾收集的线程数

-XX:+UseAdaptiveSizePolicy

自适应调整新生代的比例，eden和fromTo的比例、堆的大小，晋升阈值

-XX:GCTimeRatio=ratio

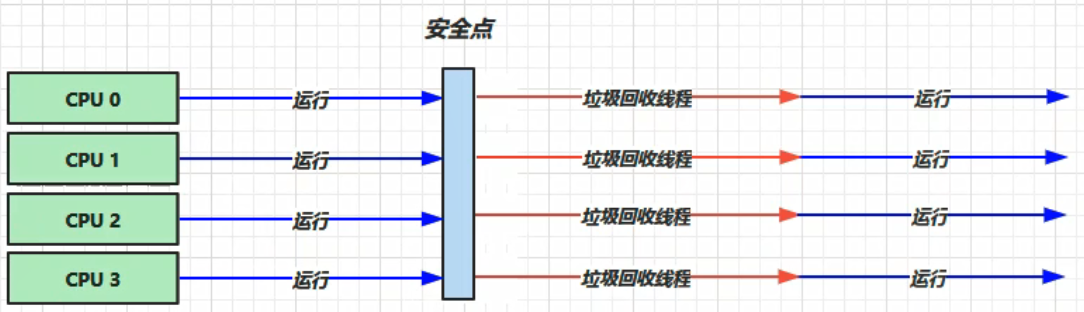
调整吞吐量（垃圾回收时间与总时间占比）若垃圾回收时间过久，会自动调整垃圾回收的时间，一般是将堆增大（可以使垃圾回收次数不频繁从而达到单位时间垃圾回收时间减少以增加吞吐量，但是单次垃圾收集时间会变长）。Ratio参数默认是99（1/(1+99)=1%,GC时间只占1%）；一般设为19（1/（1+19）=5%）

-XX:MaxGCPauseMillis=ms

最大垃圾收集暂停毫秒数，若停顿时间久，就动态将堆大小缩小从而达到单次垃圾收集时间变短。

最后两个参数是对立的目标

### （2）过程



垃圾回收时，CPU会到100%

## 3.响应时间优先

CMS（标记整理）老年代有时候会并发失败（由于碎片过多）退化成SerialOld，新生代只能试用ParNew

### （1）虚拟机参数

-XX:+UseConcMarkSweepGC与 -XX:+UseParNewGC（或SerialOld）

开启Conc（并发）+Mark（标记）Sweep（清除）算法的收集器CMS

-XX:ParallelGCThreads=n~-XX:ConcGCThreads=threads

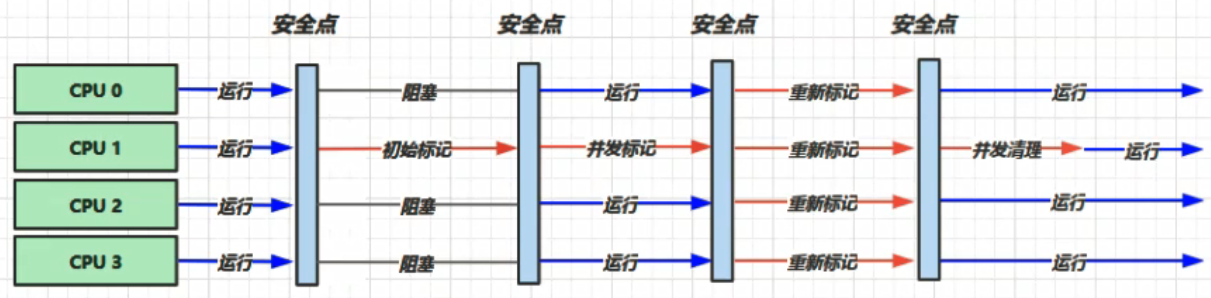
定义CMS并发线程的数量，一般为ParallelGCThreads的四分之一

-XX:CMSInitiatingOccupancyFraction=percent

因为浮动垃圾存在，当老年代到达对应百分比时，就进行垃圾回收

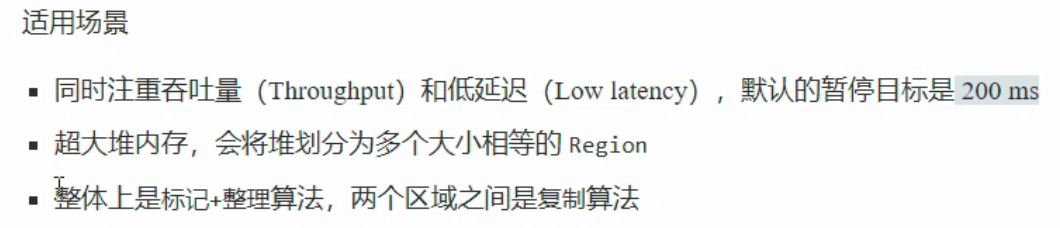
-XX:+CMSScavengeBeforeRemark

### （2）过程



CMS并发过程中由于会使用一个线程进行GC，会影响一部分吞吐量导致用户线程使用率下降。

## 4.G1收集器



### （1）虚拟机参数

-XX:+UseG1GC

使用G1垃圾收集器（jdk1.9以前需要显示的启动）

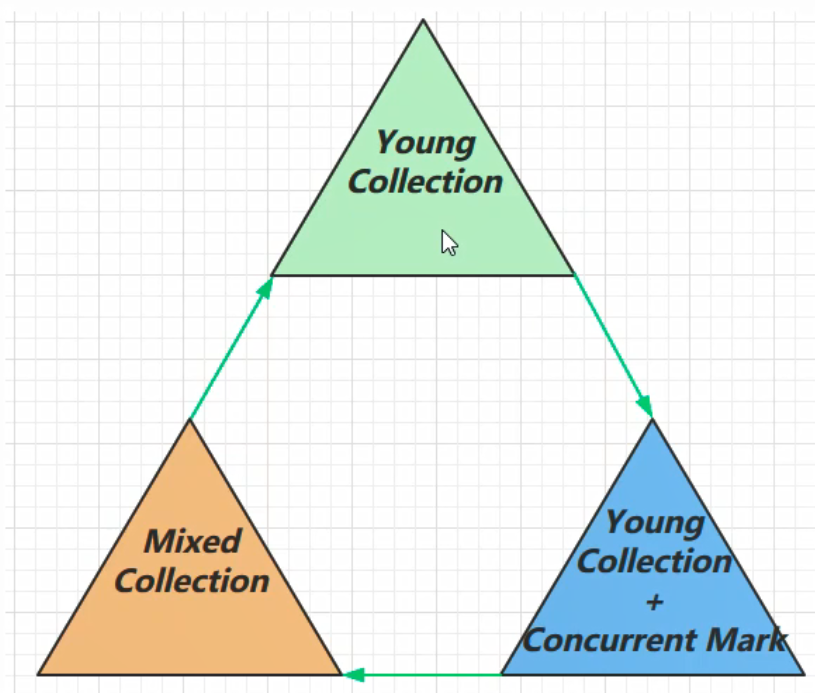
-XX:G1HeapRegionSize=size

设定每个Region的大小（2的倍数）

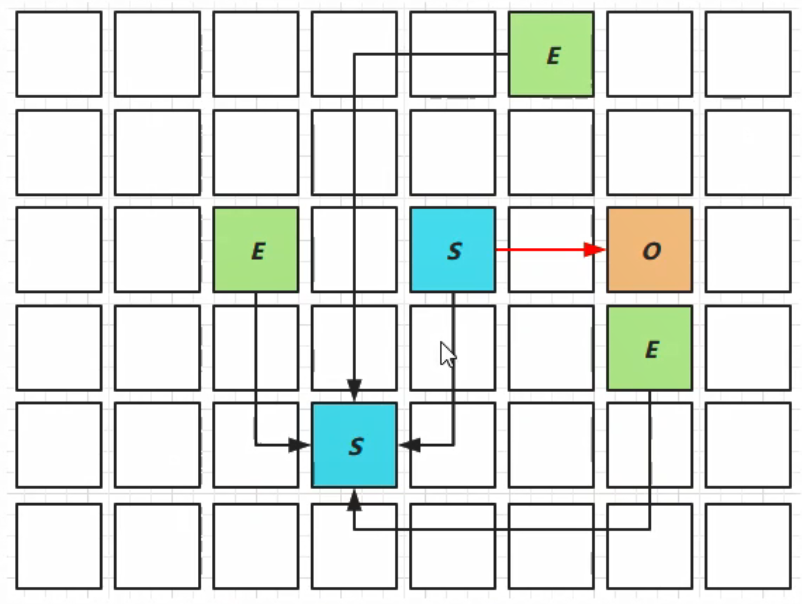
-XX:MaxGCPauseMillis=time

用户允许的收集时间

### （2）G1垃圾回收阶段



#### a.新生代收集



Eden快满的的时候会触发一次stop the world进行一次新生代垃圾回收

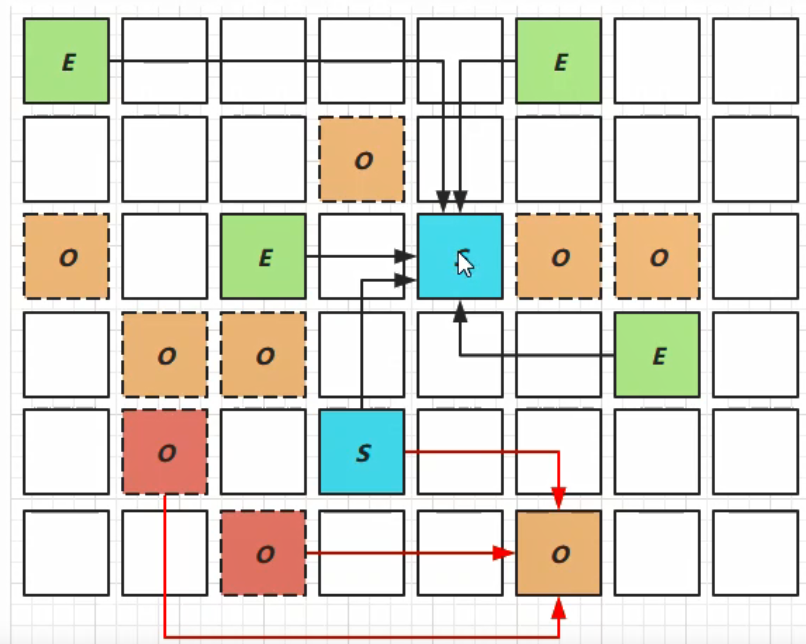
#### b.新生代收集+CM（concurrent Marking）

GC Root的初始标记在新生代GC时发生的

当老年代占用堆空间比例达到阈值时，进行并发标记（不会stop the world），由下面JVM参数决定

-XX:InitiatingHeapOccupancyPercent（默认45%） G1老年代垃圾回收的比例

#### c.Mixed Colleation



对E、S、O进行全面的垃圾回收

最终标记（Remark）会stop the world

拷贝存活 也会stop the world

-XX:MaxGCPauseMillis=ms 用户指定的停顿时间

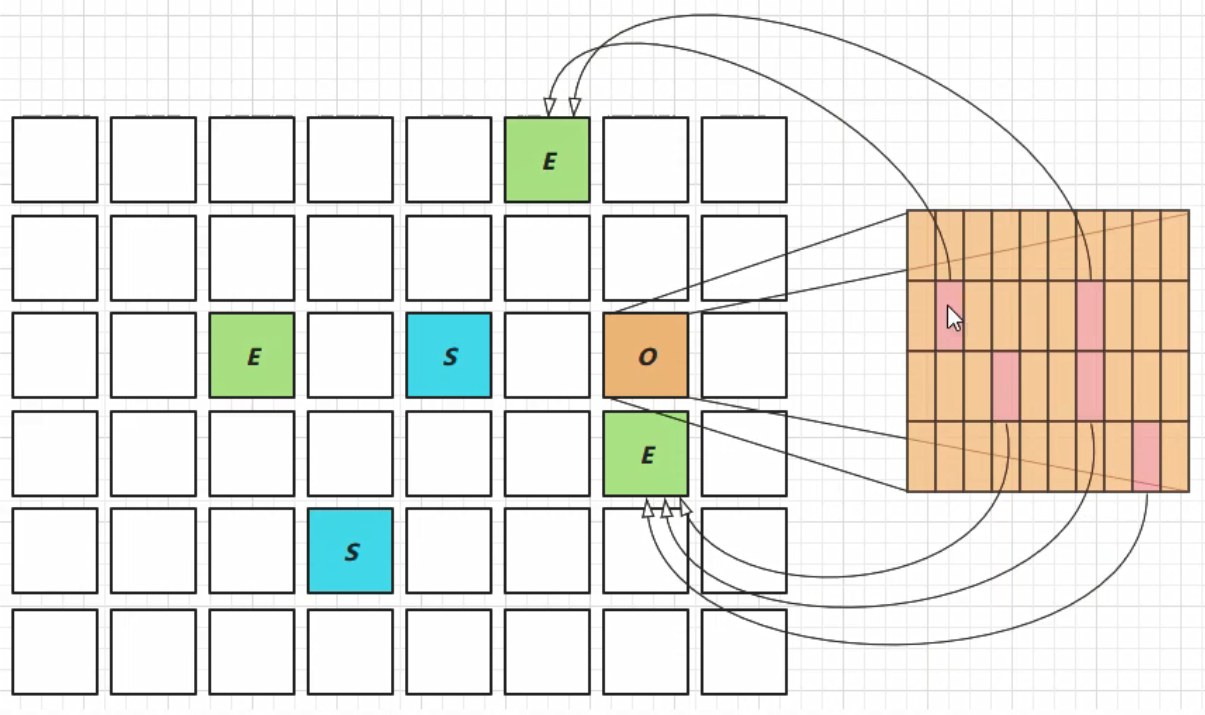
不会对所有内存进行回收，而是根据上述参数进行有选择的回收

### （3）卡表（新生代跨代引用）与写屏障

见书p84

卡表是老年代区域的一张卡，用以记录哪些对象被新生代引用，以减少搜索范围以提高效率

脏卡：老年代卡表中被新生代引用的对象



### （4）remark阶段

见书p88

### （5）jdk 8u20字符串去重（p.79）

-XX:+UseStringDeduplication 使用字符串去重

将所有新分配的字符串放入一个队列，当新生代回收的时候，G1并发检查是否有字符串是重复的，由于字符串底层使用的是char[]数组进行存储字符串，所以若有字符串重复，就让字符串引用指向同一个char[]



intern方法关注的字符串对象，而字符串去重关注的是char[]

节省一些内存但是略微会增加一些cpu占用

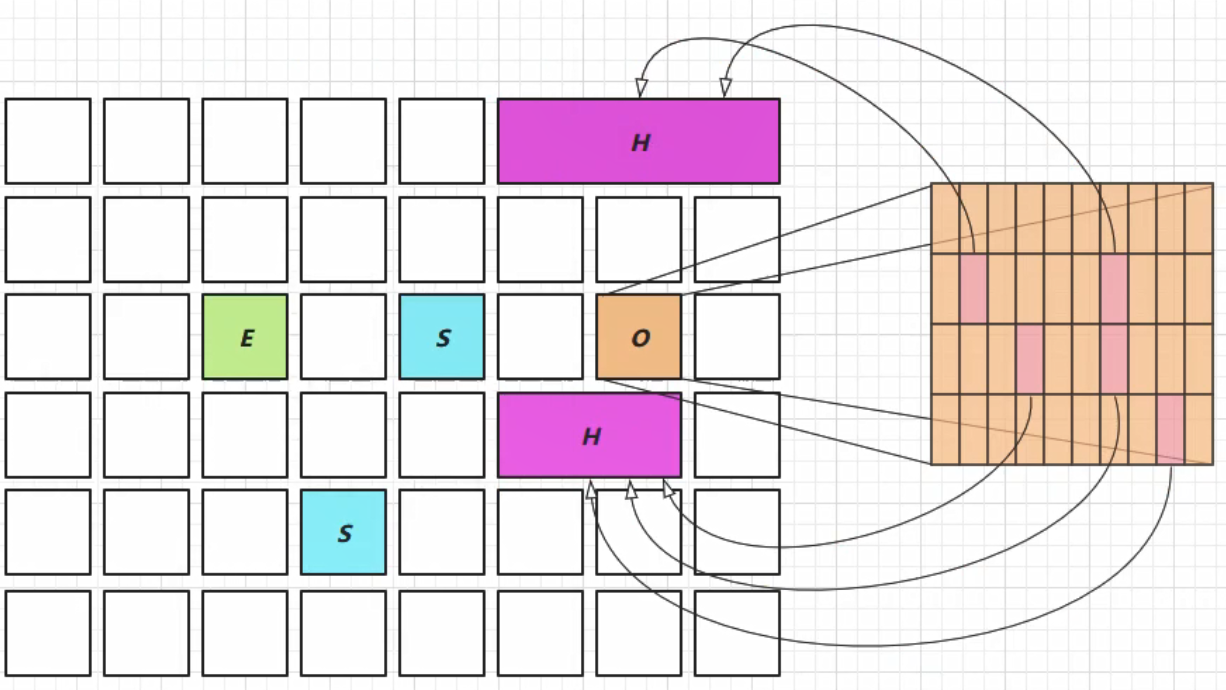
### （6）并发标记类卸载

所有对象都经过并发标记后，就知道哪些类不再使用了。当一个类加载器的所有类都不再使用，则卸载它所加载的所有类

-XX:+ClassUnloadingWithConcurrentMark 启用类卸载

### （7）回收巨型对象

超过一个Region的一半



不会对巨型对象进行拷贝，回收时优先考虑。跟踪巨型

# 五、垃圾回收调优

## 1.查看当前垃圾回收参数

java -XX:+PrintFlagsFinal -version | findstr "GC"

## 2.掌握相关工具

## 3.选择确定目标

若是以高吞吐量为目标，选择ParallelGC；

若是以低延迟（响应时间）为目标，那就用CMS、G1（JDK9中推荐G1而非CMS）

## 4.最快的GC是不发生GC

（1）若经常GC了，数据是不是太多了

resultSet=statement.executeQuery(“select \* from 大表”) （利用limit）

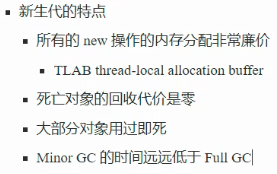
（2）数据表示是否太臃肿，一个包装类型Integer，对象头16个字节+数据值4字节+对齐（总共起码24字节），而一个int只有四个字节。（能不能对对象进行瘦身）

（3）是否存在内存泄漏

尽量不要用java来做缓存实现（例如static Map），而是使用第三方的缓存

## 5.新生代调优（TLAB）

为了防止多线程对堆的线程安全问题，所以为每个线程都会分配一块堆上的内存，叫TLAB



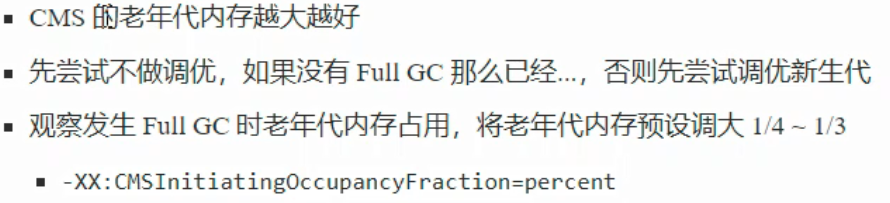
新生代大小最好是堆的四分之一到二分之一

设置合适的晋升阈值，让长期存活对象进入老年代



## 6.老年代（CMS为例）

老年代大小越快越好



# 虚拟机参数

-Xms

堆初始大小

-XX:MaxHeapSize=size 或-Xmx

堆最大大小

-XX:MaxNewSize=size或-Xmn

新生代大小

-XX:+UseAdaptiveSizePolicy和-XX:InitialSurvivorRatio

幸存区比例（动态）

-XX:SurvivorRatio=ratio

幸存区比例

-XX:MaxTenuringThreshold=threshold

晋升阈值

-XX:+PrintTenuringDistribution

晋升详情

-XX:+PrintGCDetails -verbose:gc

GC详情

-XX:+ScavengeBeforeFullGC FullGC前MinorGC

-XX:+UseSerialGC

开启串行GC

-XX:+UseParalledGC或-XX:+UseParallelOldGC

只要输入一个参数，即可开启并行垃圾回收

-XX:GCTimeRatio=ratio

调整吞吐量（垃圾回收时间与总时间占比）若垃圾回收时间过久，会自动调整垃圾回收的时间，一般是将堆增大（可以使垃圾回收次数不频繁从而达到单位时间垃圾回收时间减少以增加吞吐量，但是单次垃圾收集时间会变长）。Ratio参数默认是99（1/(1+99)=1%,GC时间只占1%）；一般设为19（1/（1+19）=5%）

-XX:MaxGCPauseMillis=ms

最大垃圾收集暂停毫秒数，若停顿时间久，就动态将堆大小缩小从而达到单次垃圾收集时间变短（G1收集器中是进行动态选择相关的块进行回收）。

-XX:+UseConcMarkSweepGC与 -XX:+UseParNewGC（或SerialOld）

开启Conc（并发）+Mark（标记）Sweep（清除）算法的收集器CMS

-XX:ParallelGCThreads=n~-XX:ConcGCThreads=threads

定义CMS并发线程的数量，一般为ParallelGCThreads的四分之一

-XX:CMSInitiatingOccupancyFraction=percent

因为浮动垃圾存在，当老年代到达对应百分比时，就进行垃圾回收

-XX:+CMSScavengeBeforeRemark

-XX:InitiatingHeapOccupancyPercent

（默认45%） G1老年代垃圾回收的比例

-XX:+UseStringDeduplication

使用字符串去重

-XX:+ClassUnloadingWithConcurrentMark

启用类卸载