高并发&压力测试&JVM 优化实战

课程回顾：

（1）、实现高并发需要考虑的问题：12 条金字规则

实现高并发需要考虑：

（1）系统的架构设计，如何在架构层面减少不必要的处理（网络请求，数据库操作等）--- 亿级流量的网站

例如：使用 Cache 来减少 IO 次数，使用异步来增加单服务吞吐量，使用无锁数据结构来减少响应时间；

（2）网络拓扑优化减少网络请求时间（rt）、如何设计拓扑结构，分布式如何实现（单体架构，分布式架构，微服务，service mesh, serverless）？

读缓存，写异步（数据一致性）

（3）系统代码级别的代码优化，使用什么设计模式来进行工作？哪些类需要使用单例，哪些需要尽量减少 new 操作?

（4）提高代码层面的运行效率、如何选取合适的数据结构进行数据存取？如何设计合适的算法?

（5）任务执行方式级别的同异步操作，在哪里使用同步，哪里使用异步?

（6）JVM 调优，如何设置 Heap、Stack、Eden 的大小，如何选择 GC 策略,控制 Full GC 的频率?

（7）服务端调优（线程池，等待队列）

（8）数据库优化减少查询修改时间。数据库的选取？数据库引擎的选取？数据库表结构的设计？数据库索引、触发器等设计？是否使用读写分离？还是需要考虑使用数据仓库?

（9）缓存数据库的使用，如何选择缓存数据库？是 Redis 还是 Memcache？如何设计缓存机制?

（10）数据通信问题，如何选择通信方式？是使用 TCP 还是 UDP，是使用长连接还是短连接？NIO 还是 BIO？netty、mina 还是原生 socket?

（11）操作系统选取，是使用 winserver 还是 Linux？或者 Unix?

（12）硬件配置？是 8G 内存还是 32G，网卡 10G 还是 1G？例如：增加 CPU 核数如 32 核，升级更好的网卡如万兆，升级更好的硬盘如 SSD，扩充硬盘容量如 2T，扩充系统内存如 128G；

（2）、服务端调优

线程池调优：最大队列数，等待队列，最大线程数，最大队列空闲数

```
server:
  tomcat:
    uri-encoding: UTF-8
```

```
#最大工作线程数，默认 200, 4 核 8g 内存，线程数经验值 800
#操作系统做线程之间的切换调度是有系统开销的，所以不是越多越好。
max-threads: 1000
# 等待队列长度，默认 100
accept-count: 1000
max-connections: 20000
# 最小工作空闲线程数，默认 10，适当增大一些，以便应对突然增长的访问量
min-spare-threads: 100
```

keepalive

防止频繁的建立连接，释放连接，造成资源的浪费，我们使用了 keepalive 的方式，保持客户端和服务端的长连接。但是连接太多也不行，因为每一个长连接都会会占用很多资源，必须综合考虑一个 balance 的值，同时设置好阀值。

（3）、jmeter 压力测试

性能参数指标：QPS，TPS，RT ，平均响应时间，中位数，90 百分位，95 百分位，99 位，吞吐量。

Jvm 调优

一、 为什么要进行 JVM 调优

1、内存耗尽

2、OOM

（1）、垃圾太多，内存占满了，程序跑不动了； 卡死

（2）、Java 程序提供自动的垃圾回收器（c,c++）,垃圾回收的线程太多，频繁的垃圾收集

目标只有一个： 回收垃圾。

思考问题：JVM 可利用多大的内存空间?

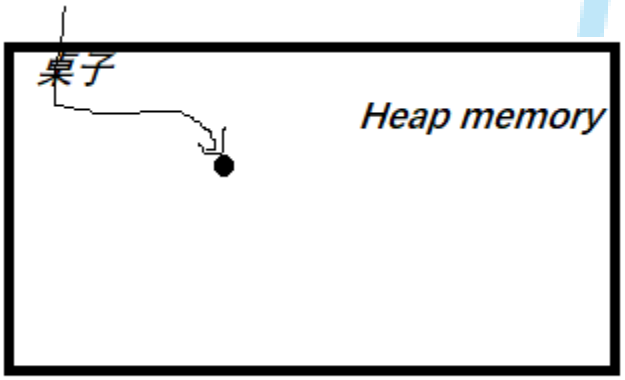
操作系统：

32 位： 2 的 32 次方 = 4G (操作系统一半空间) 2g --- jvm

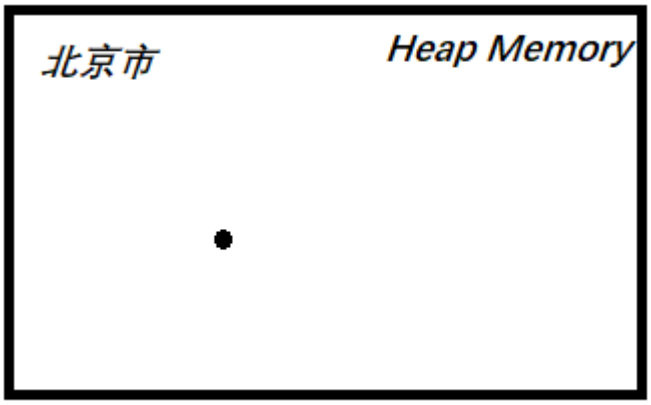
64 位： 2 的 64 次方 = 16384PB （操作系统：2G） 剩下的空间，jvm 都可以使用。

二、 JVM 调优原则

1、gc 时间足够小 （堆内存设置小一些）----



2、gc 次数足够少 （堆内存设置大一些）



内存装满的时候，开始触发垃圾回收

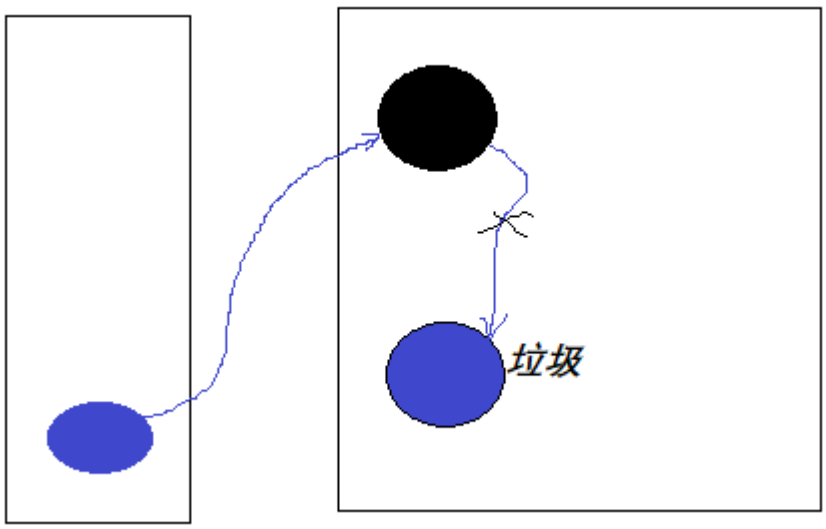
如果发现 JVM gc 垃圾回收次数比较多，heap memory 设置小了，如果垃圾回收时间变长了，说明 heap memory 设置太大。

3、发生 full gc 的周期足够长

老年代内存大小设置要合理，要设置大一些。

三、 Jvm 调优原理

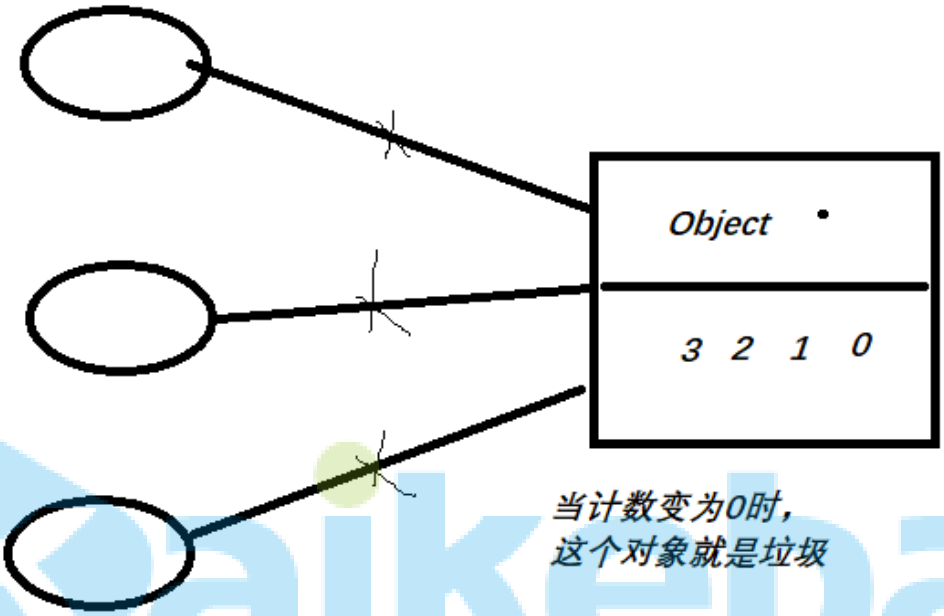
1. 什么是垃圾?



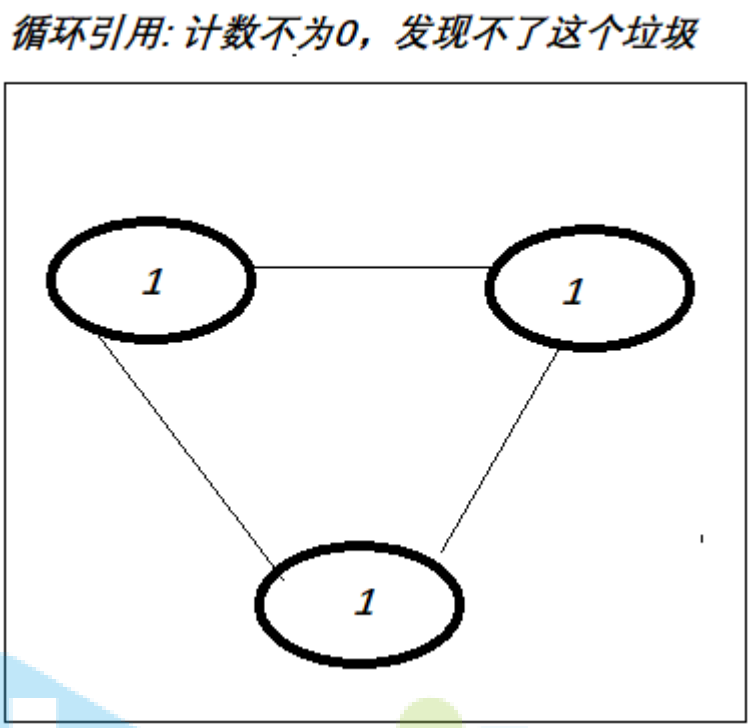
没有被引用的对象，就是垃圾。

2. 如何找到这些垃圾？

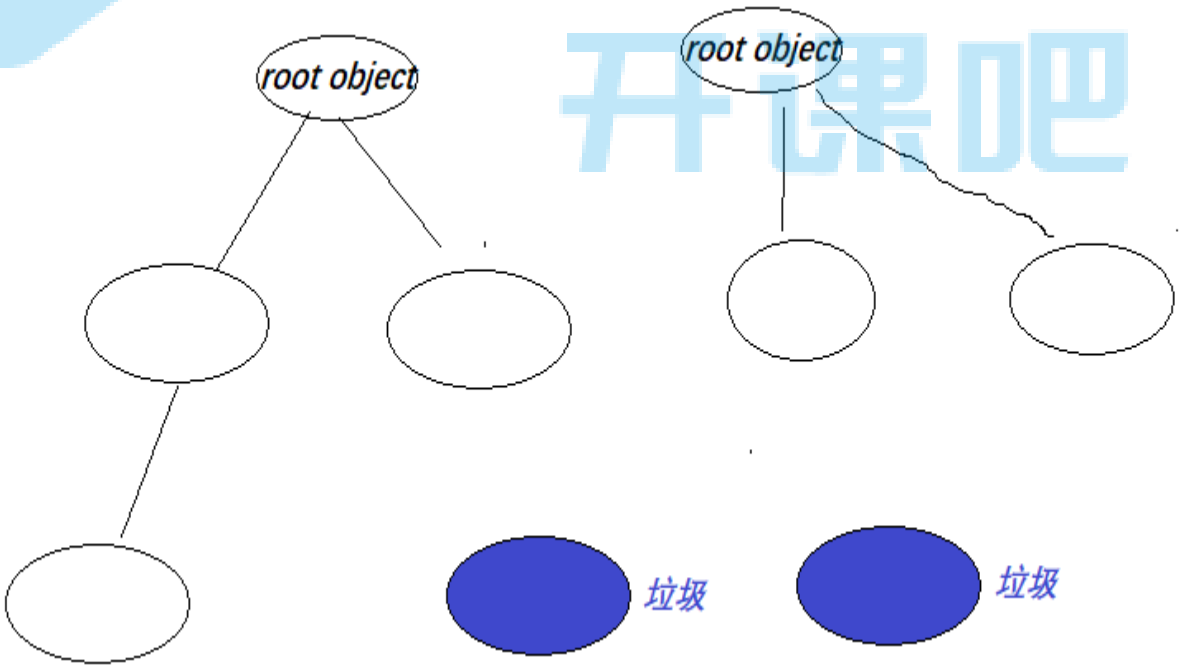
1）引用计数法



引用计数法，没有办法解决循环引用的问题。存在循环引用的话，那么就无法发现这个垃圾。

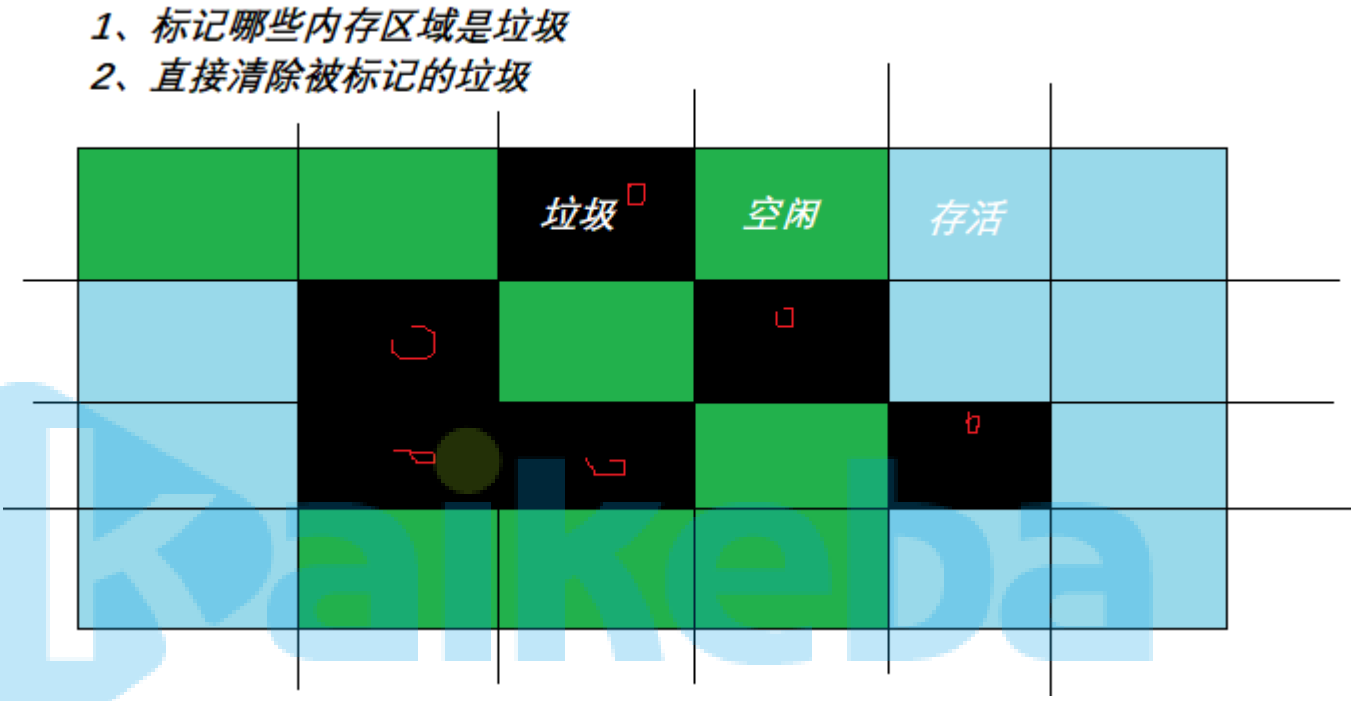


2）根可达算法



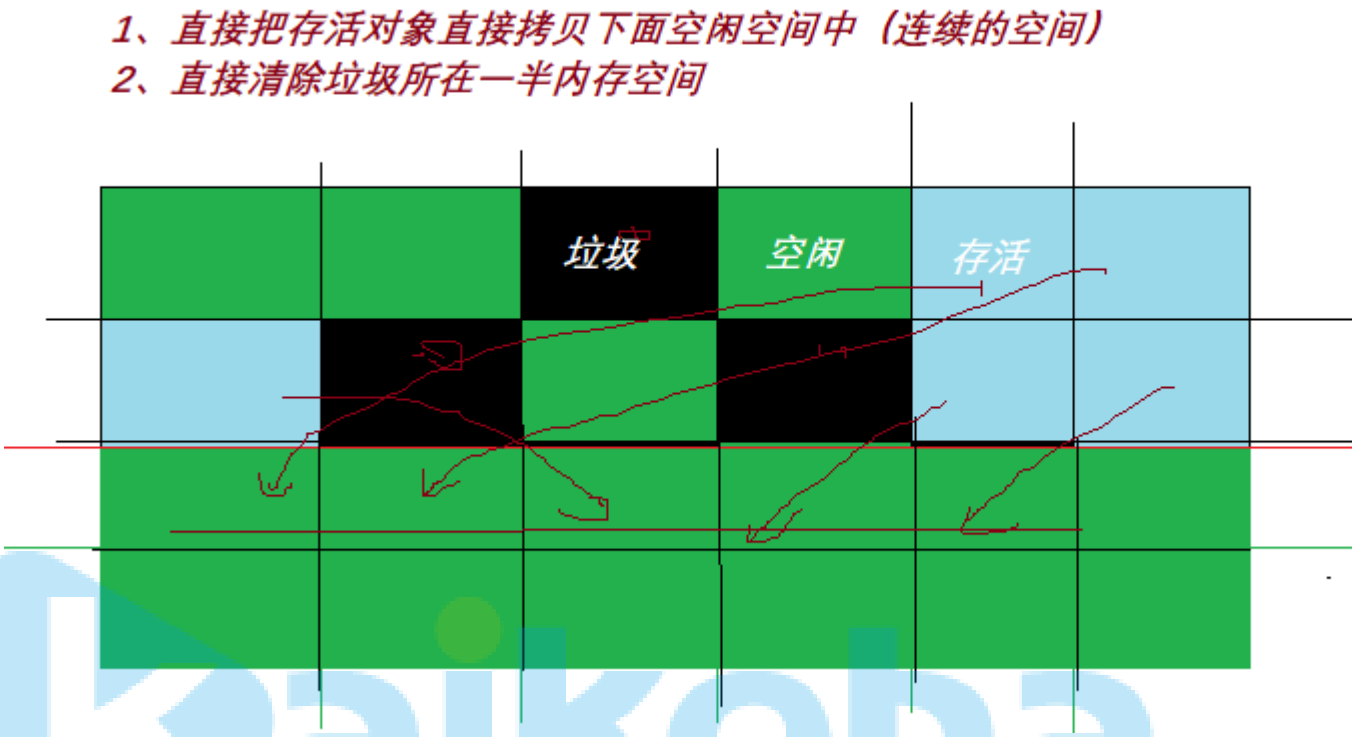
3． 垃圾回收算法

3.1． Mark – sweep 标记清除算法



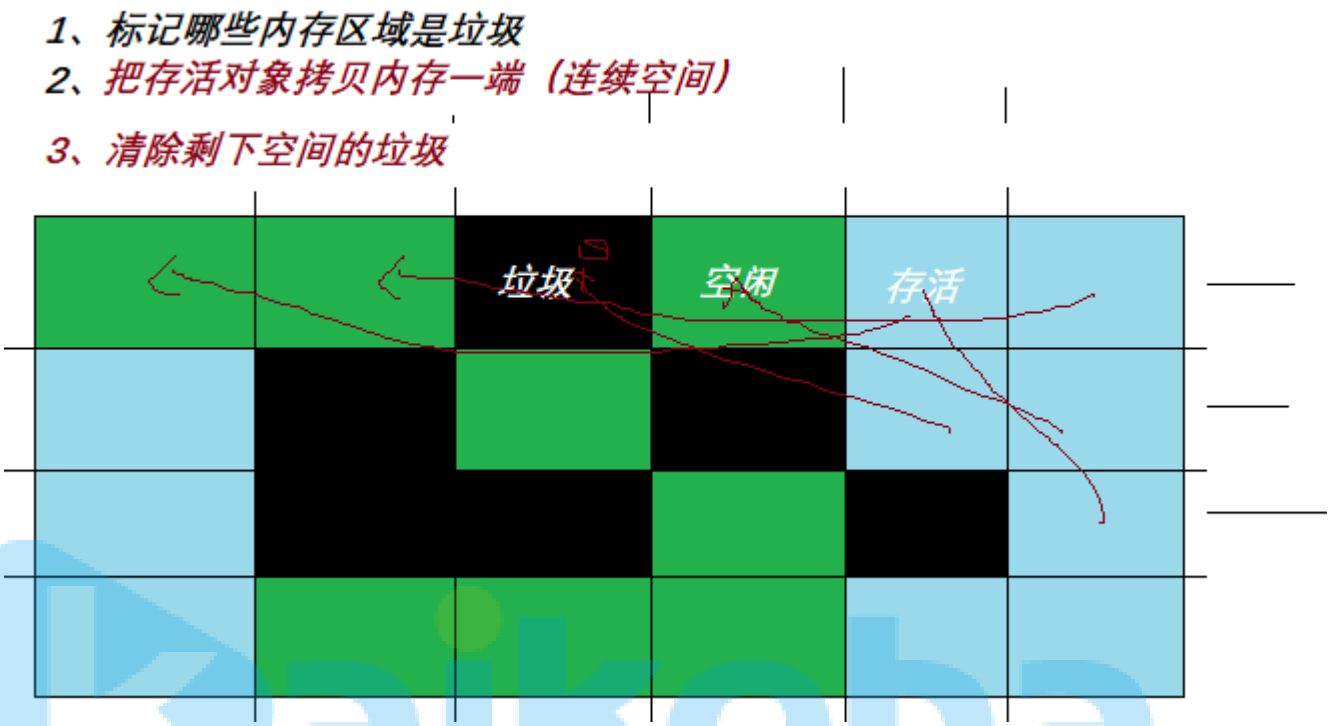
优点： 性能好； 缺点：会产生内存碎片。

3.2. Copying（拷贝）



优点：没有内存碎片，缺点：存在内存空间的浪费。

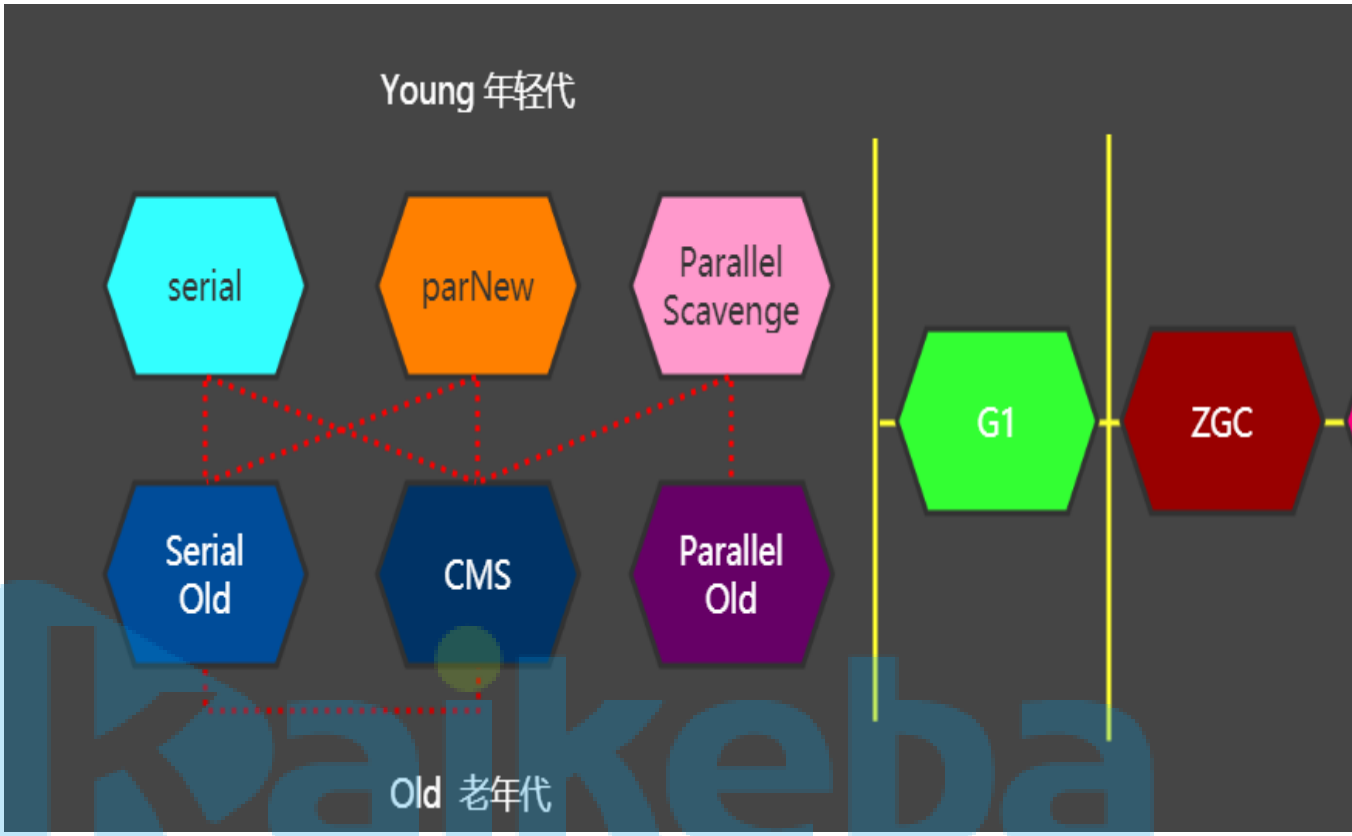
3.3. Mark-compact



优点：没有内存碎片，使用内存是一个连续的空间。

缺点：性能比较低下，拷贝操作，压缩，性能比较低下。

4. 垃圾回收器



串行化垃圾回收器:

Serial(年轻代垃圾回收) + serial old ： 支持单核心 cpu 垃圾回收

并行垃圾回收器:

Parnew （年轻代垃圾回收）：是 Parallel scavenge 增强型垃圾回收器，主要是为了更好和 cms 垃圾回收器相结合而开发的垃圾回收器。

Cms（老年代垃圾回收）：垃圾回收时候和线程执行交叉进行。STW 耗时更少。

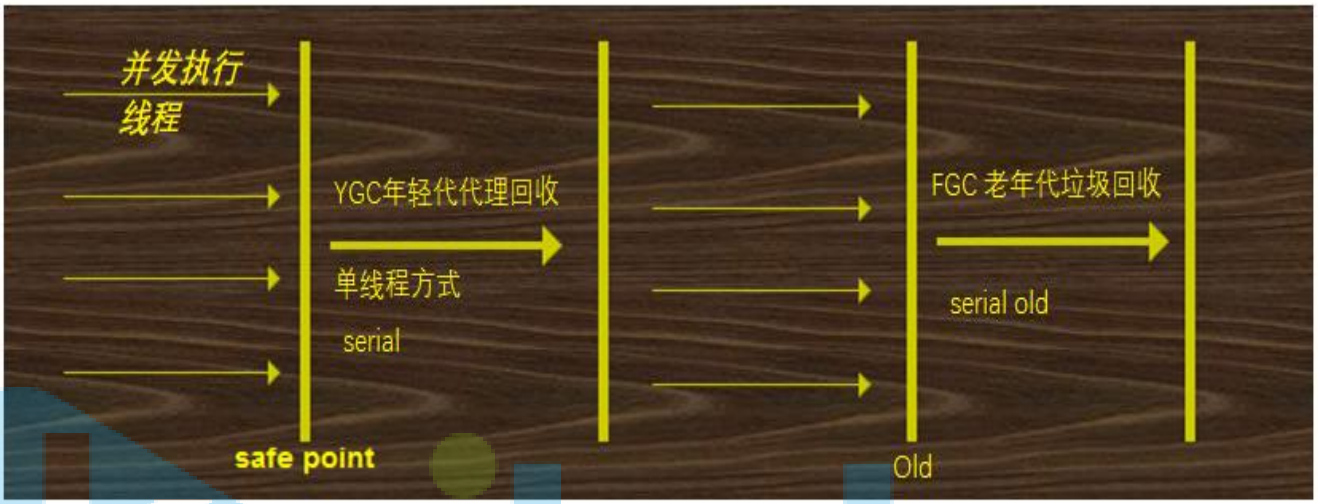
Parallel scavenge（年轻代垃圾回收）：并行垃圾回收器，特点：stop the world gc 暂停时间稍微长

Parallel old（老年代垃圾回收）： 并行垃圾回收器

G1: 同时支持年轻代，老年代垃圾回收。

5． Serial 垃圾回收器

Stop The World：整个世界都停止了！！（代码执行都停止了，开始垃圾回收）

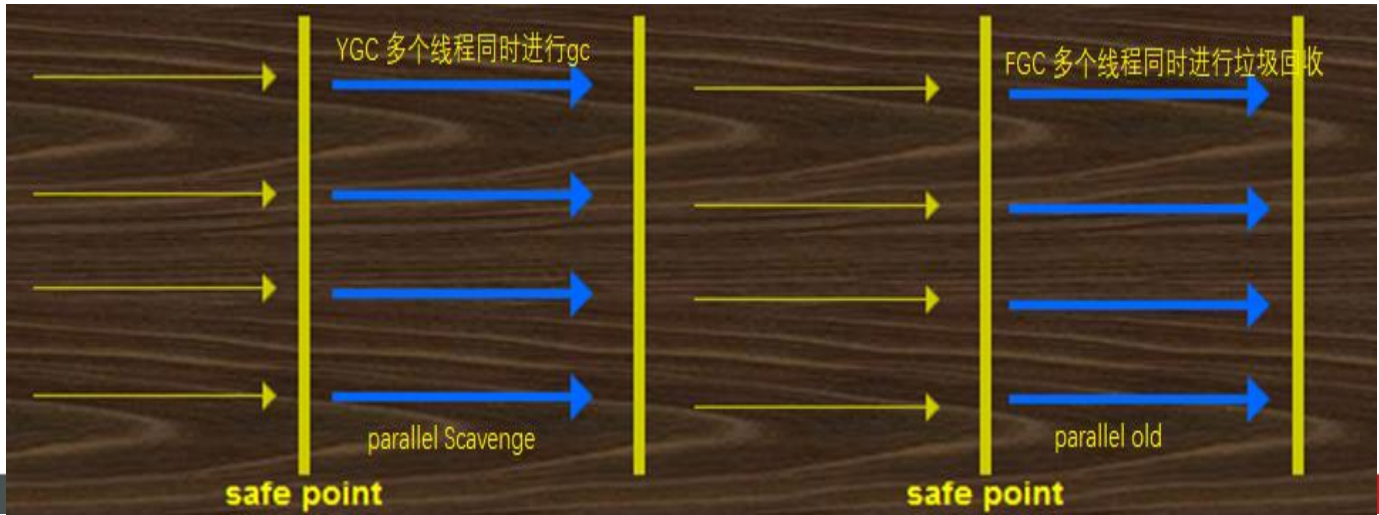


safe point：让程序暂停，进行垃圾回收！

1、循环末尾
2、方法返回前
3、抛出异常的时候

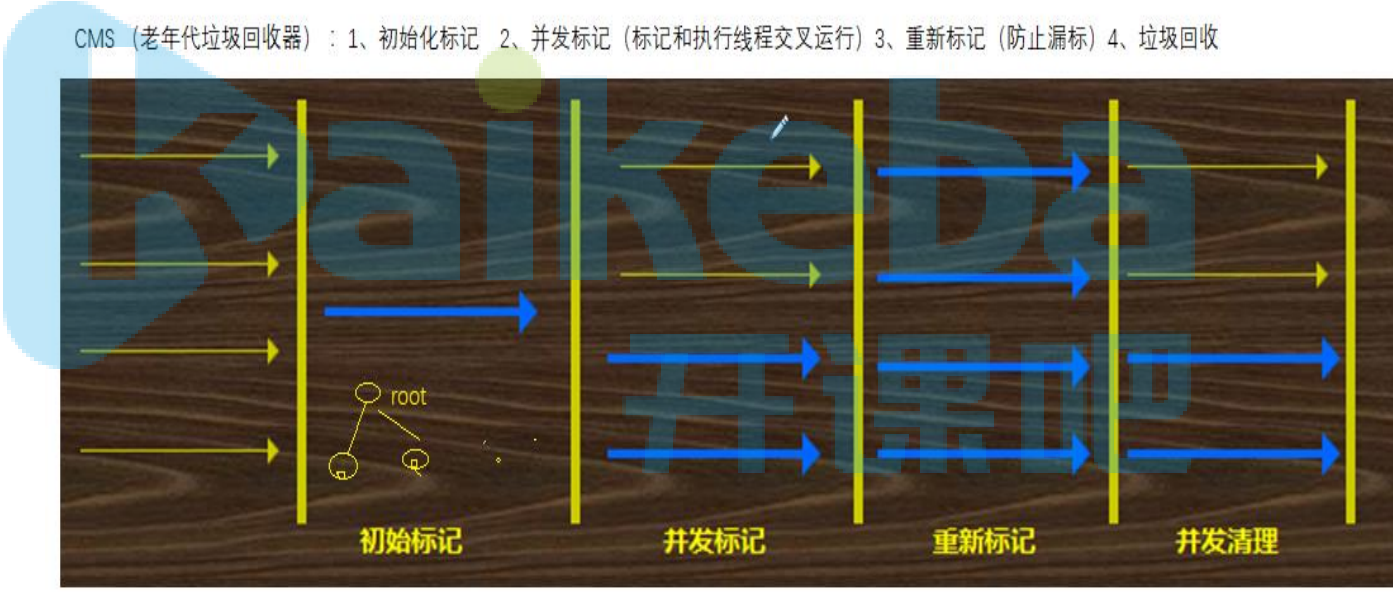
Gc 的时候会 STW,进行垃圾回收，gc 会消耗时间，程序执行时间变长，QPS,TPS

6． Ps PO

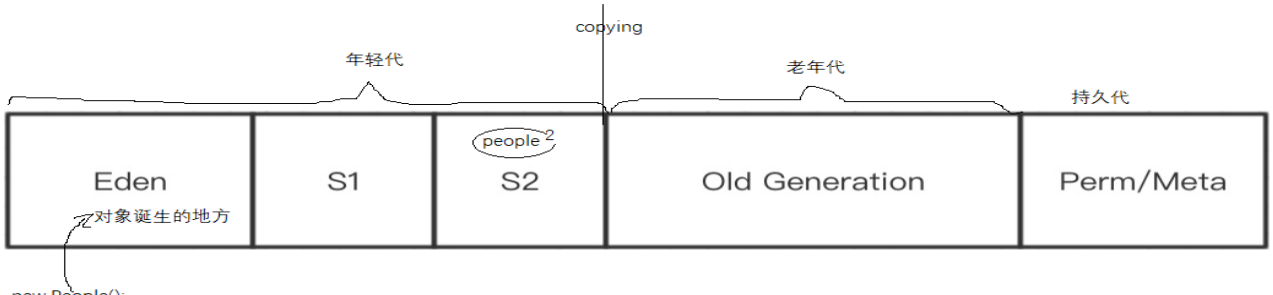


Ps ,po 都采用并行方式进行垃圾回收，适用吞吐量优先的调优方式。

7． Cms



四、 分代模型



当这个对象年龄变大了，默认age=15，对象就会进入老年代区域。

年龄修改： -XX:MaxTenuringThreshold = 20 , 对象年龄到20岁时间，进入老年代。

4、当老年代空间存储满了，触发full gc

ps，ps ： 当我们的年轻代存储满了，触发垃圾回收，老年代： 存储满了，触发垃圾回收

cms垃圾回收器： 老年代

`-XX:CMSInitiatingoccupanyFraction` 设置堆内存使用率的阈值，一旦达到该阈值，便开始进行回收。

- JDK5及以前版本的默认值为`68%`，即当老年代的空间使用率达到68%时，会执行一次CMS回收。
- JDK6及以上版本默认值为`92%`
- 如果内存使用率增长缓慢，则可以设置一个稍大的值，**大的阀值可以有效降低CMS的触发频率**，减少老年代回收的次数可以较为明显地改善应用程序性能
- 反之，如果应用程序内存使用率增长很快，则应该降低这个阈值，以**避免频繁触发老年代串行收集器**。因此通过该选项便可以有效降低 Full GC 的执行次

五、 Jvm 实战调优

1、gc 时间足够小 （堆内存设置小一些）

2、gc 次数足够少 （堆内存设置大一些）

3、发生 full gc 的周期足够长(full gc 次数要少)

把调优原则和垃圾回收器结合起来，可以进行 jvm 调优了。

Jvm 调优几个参数：

-Xmx 最大堆内存 ,一般情况下内存设置不能超过内存 80%,一般设置为 50% -- 80%

-Xms 初始化堆内存 ，这个内存一般情况下都必须和 Xmx 设置为一致大小。

-Xmn 设置年轻代大小

-Xss 每一个线程堆栈大小，这个大小可以限制线程数量，默认大小为 1M

入门参数设置：

```
nohup java -Xmx3550m -Xms3550m -Xmn2g -Xss256k -jar jshop-web-1.0-SNAPSHOT.jar --spring.config.addition-location=application.yaml > log.log 2>&1 &
```

gc 日志：使用 gceasy.io 网站分析 gc 情况：

-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:gc.log

MetaspaceSize 默认大小： 20m (方法，常量池，class) -→ 扩容，进行一次 full gc，扩容：full gc

吞吐量优先：

Ps po

```
nohup java -Xmx3550m -Xms3550m -Xmn2g -Xss256k -XX:+UseParallelGC -XX:MetaspaceSize=200m -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:gc.log -jar jshop-web-1.0-SNAPSHOT.jar --spring.config.addition-location=application.yaml > log.log 2>&1 &
```

响应时间优先：

Cms + paraNew

```
nohup java -Xmx3550m -Xms3550m -Xmn2g -Xss256k -XX:+UseParNewGC
```

```
-XX:+UseConcMarkSweepGC -XX:MetaspaceSize=200m -XX:+PrintGCDetails
-XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC
-Xloggc:gc.log -jar jshop-web-1.0-SNAPSHOT.jar
--spring.config.addition-location=application.yaml > log.log 2>&1 &
```