# 垃圾回收

<https://juejin.cn/post/6844903651266265095>

## 如何判断垃圾：

### 0:引用计数法：

记录每个对象被引用次数

**问题**：循环引用问题无法解决

### 1:可达性分析法

从root找，查看是否有root指向此对象，如果没有，那么就是垃圾

根对象：

1:栈中的变量

2:常量池中的变量

3:static变量

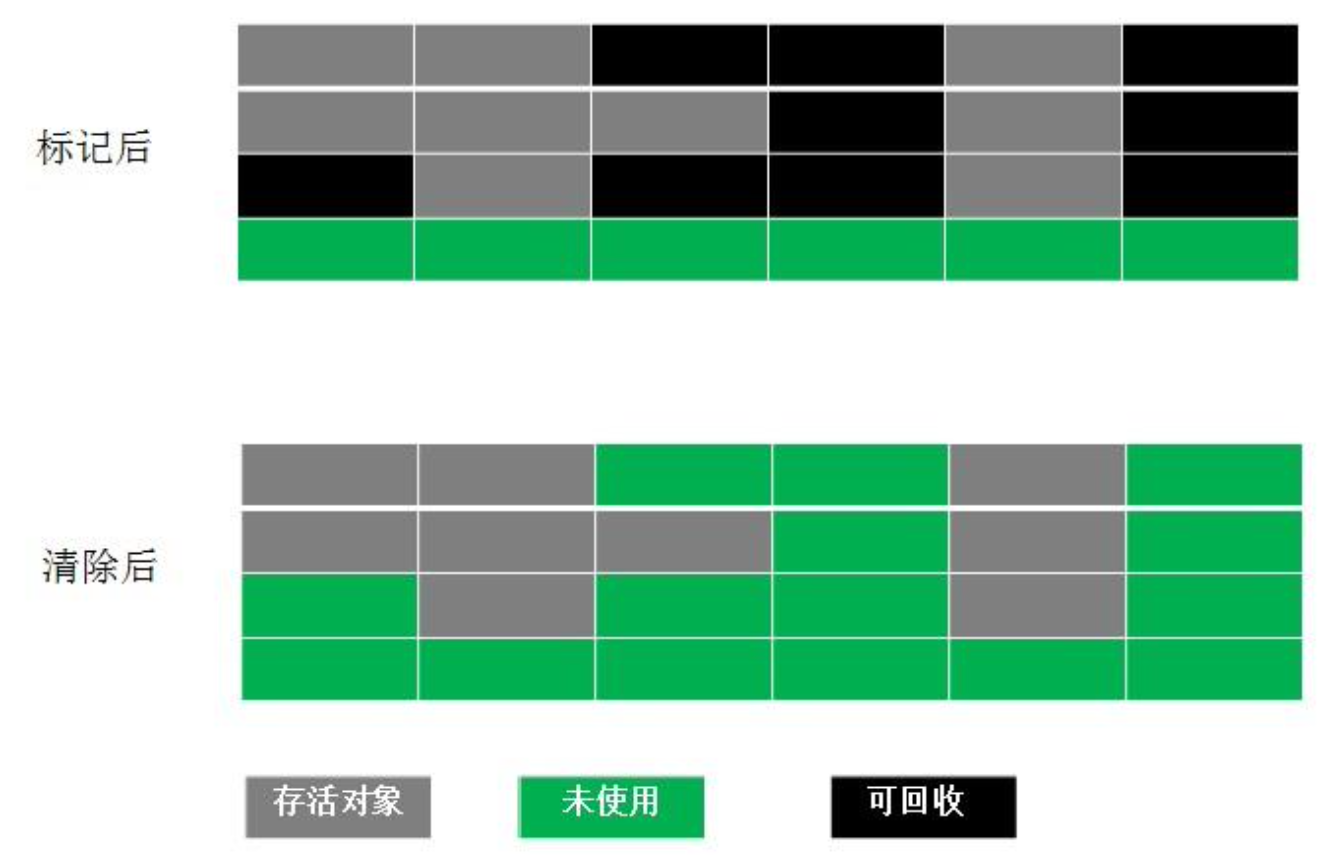
4:JNI指针

## 常见的垃圾回收算法：

### 标记-清除算法：

标记对象是否被使用，然后清除没有被使用的

问题：会产生大量碎片

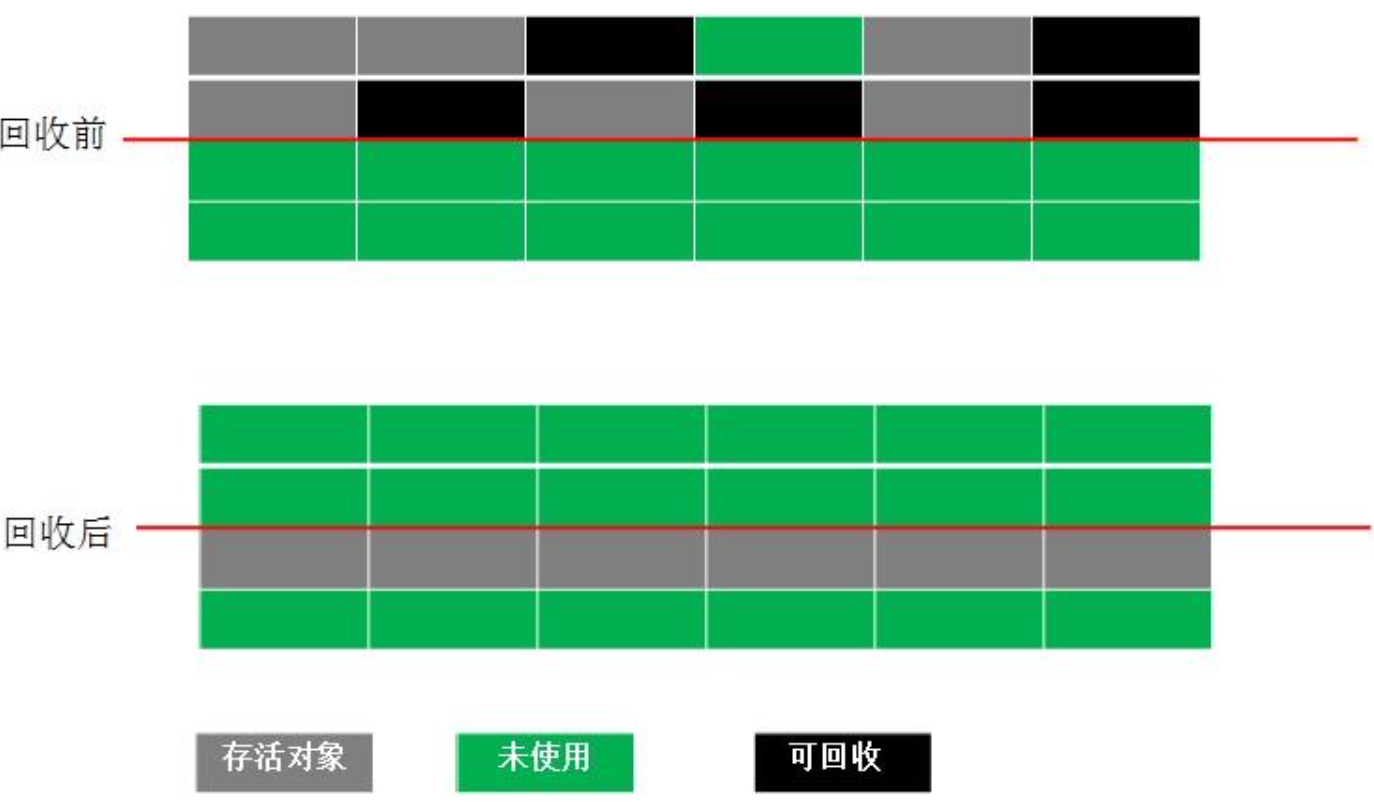


### 复制算法：

把内存分为两个区域，每次垃圾回收复制存活的对象到另一半

问题：如果对象存活多，那么效率低，并且浪费空间

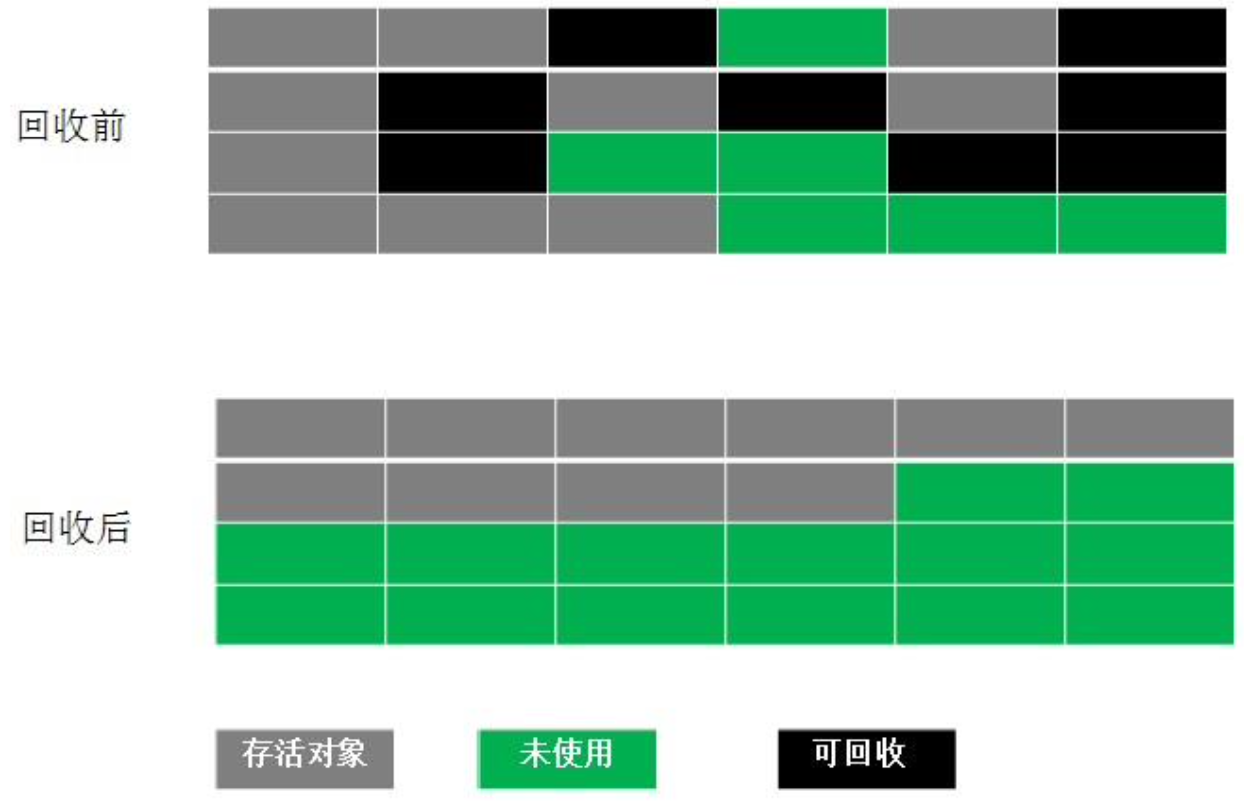
适用于对象少的场景，对象少效率高



### 标记压缩：

先进行标记，然后把数据进行压缩

效率在数据量大的时候，比较有优势



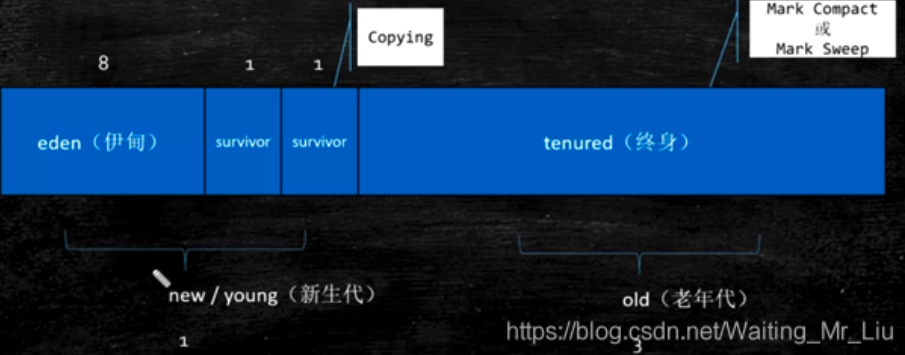
## 垃圾回收器：

### 前置知识：

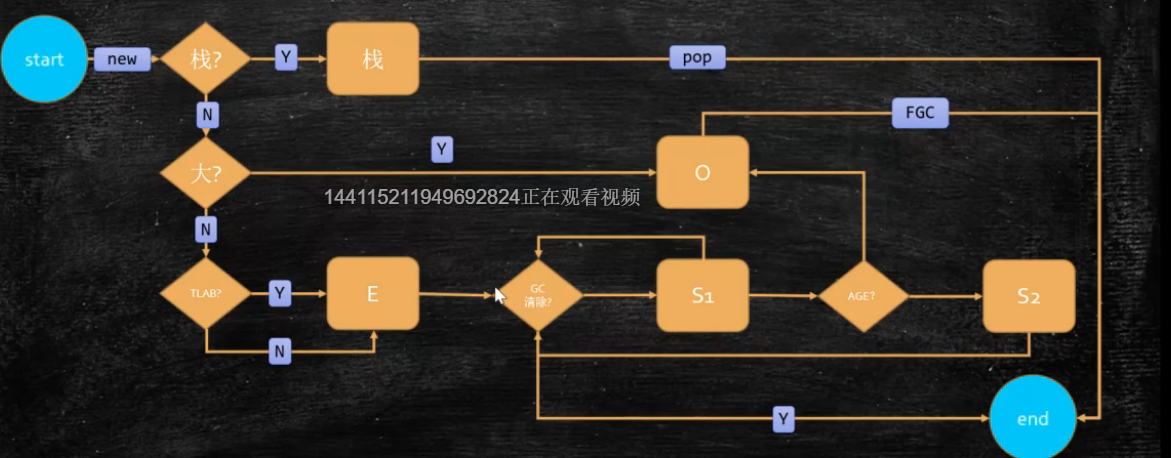
堆内存中，将对象分为了2代，方便进行管理

1:年轻代（存活的少，使用复制算法）

2:老年代（存活的多，使用标记整理算法）



### 对象生命周期：



1:首先在看是否可以在栈上创建对象

前提：空间够，对象不存在逃逸

2:如果不行，判断是否能在年轻代放的下，如果不能直接在Old区创建

3:如果可以，那么查看是否可以在 Eden中的线程专属区域分配

背景：每个线程有一块独占的区间，大概1M，可以避免Eden争抢

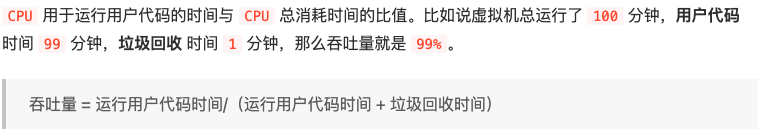
4:如果不能，那么在Eden普通区创建

5:如果在年轻代垃圾回收了15次（对象头里保存），依然存活，那么此时转移到Old区去

ps：CMS默认年龄是6，其他都是15

### 垃圾回收关键概念：

#### 1:吞吐量



#### 2:响应时间

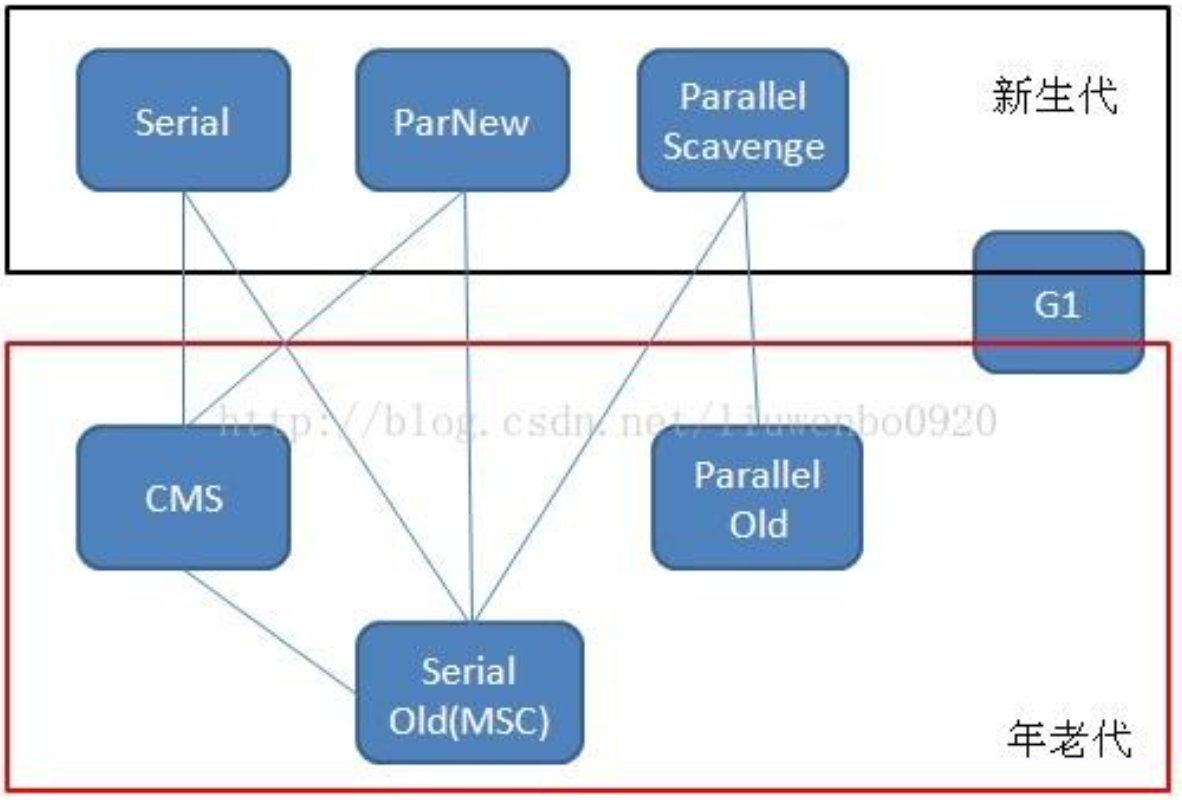
一次请求从开始到返回的时间

#### 问题点：

回收会stop\_world，那么会导致垃圾回收时间特别长，而导致响应时间长。所以如果对于实时性要求比较高的业务，那么就不能使用stop\_world类型的，或者说stop\_world比较久的。

如果是事实性要求不高的，使用stop\_world的算法，吞吐量比较高

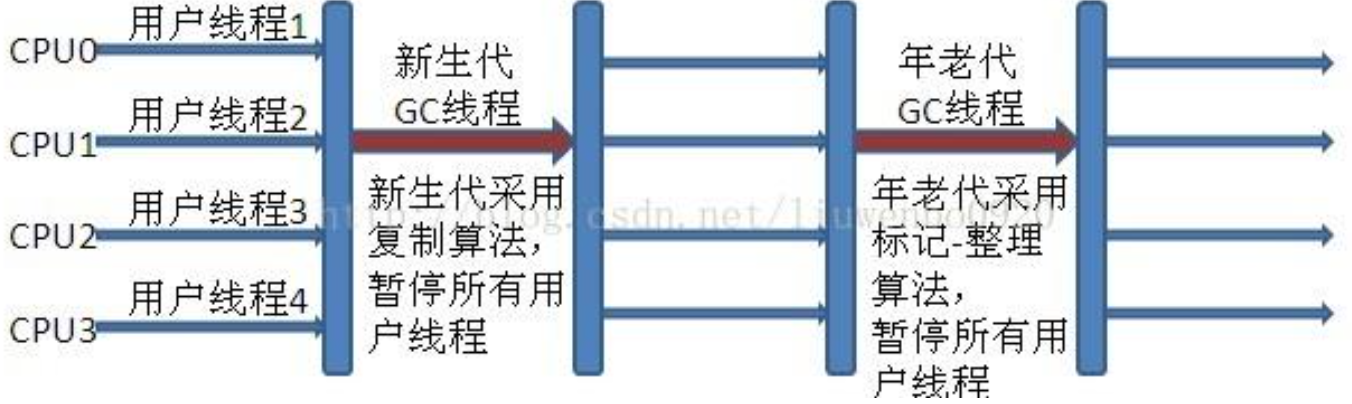
### 垃圾回收器：



#### Serial（年轻代）：

单线程/复制算法 stop\_world

没有合理使用线程，所以吞吐量和 响应时间都有问题



#### SerialOld（老年代）：

单线程/标记整理算法 stop\_world

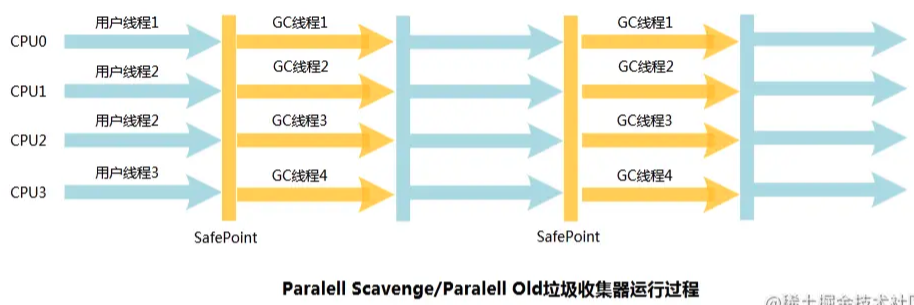
没有合理使用线程，所以吞吐量和 响应时间都有问题

同时也是CMS的后备算法解决碎片问题

#### ParallelScavenge（年轻代）：

多线程/复制算法 stop\_world

合理使用线程，相比于Serial提升了吞吐量和响应时间/但是因为还是stop-world，所以如果**内存很大**，响应时间还是会有问题



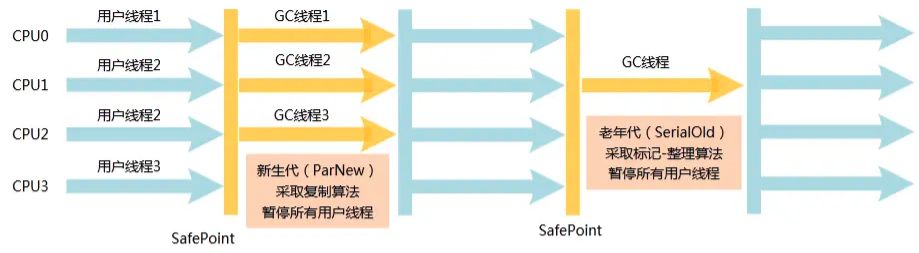
#### ParallelOld（老年代）：

多线程/标记压缩 stop\_world

合理使用线程，相比于Serial提升了吞吐量和响应时间但是因为还是stop-world，所以如果**内存很大**，响应时间还是会有问题

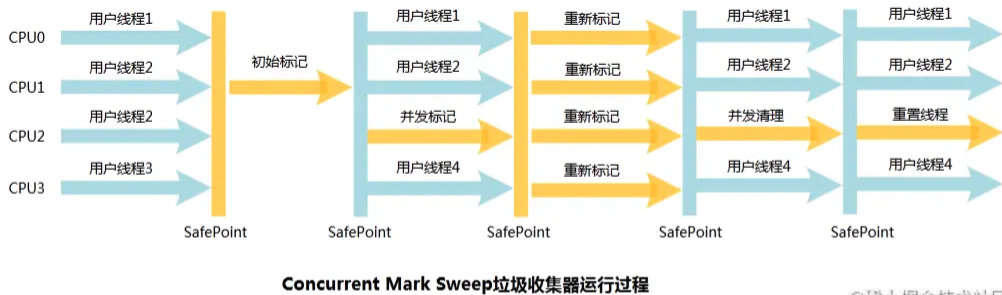
#### ParNew（年轻代）：

类似于ParallelSecavenge，主要配合老年代为CMS时使用



#### CMS（老年代）：

高级算法，一定程度解决了 po的大内存下的 响应时间的问题，但是没有PO时候的吞吐量高了（线程切换）



1:初始标记：仅仅标记**GC Roots根**对象，**Stop the world**

2:并发标记：不阻塞业务，多线程进行标记

3:重新标记：修正在并发标记过程中的引用变化的对象，**Stop the world，但不会很久**

4:并发清除：清理对象

##### CMS存在的问题：

1:严重依赖CPU资源，会拉低吞吐量

2:CMS会存在浮动垃圾（在并发清除阶段产生的垃圾）

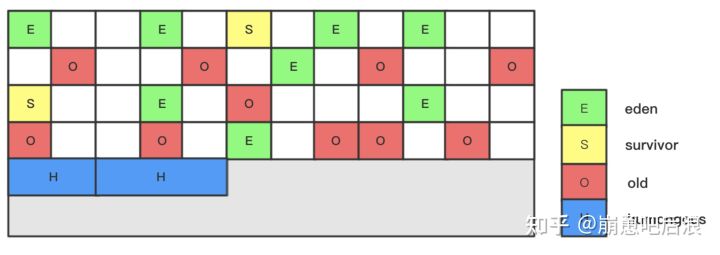
3:CMS使用标记-清除，会产生大量的内存碎片

4:当碎片过多导致没有空间分配对象时，使用SerialOld清理碎片（超级慢）

#### G1（物理不分代）:

逻辑分代，物理不分代（物理空间打散了）

G1 是 JDK 1.7 中正式投入使用的用于取代 CMS 的 **压缩回收器**。它虽然没有在物理上隔断 **新生代** 与 **老生代**，但是仍然属于 **分代垃圾回收器**。G1 仍然会区分 **年轻代** 与 **老年代**，年轻代依然分有 Eden 区与 Survivor 区。



G1首先将堆区分为大小相等的 Region，然后维护一个 优先列表（回收价值最大的Region）

1:初始标记

2:并发标记

3:最终标记

4:筛选回收

优点：

1:筛选回收，回收时间可控

2:更好的解决CMS内存碎片问题

#### 总结：

serial 几十M

PS 几个G

CMS 20G

G1 上百G

ZGC 16T

## 调优：

1:前期调优

2:运行过程中调优（慢，卡顿）

3:OOM

### 前期调优：

提前规划，预估并发量（淘宝最大54W/s，12306 100/s）

1:根据业务特点，选择垃圾回收器（PS/PO/CMS/G1/ZGC）

2:合理配置Xms Xmm Xss等大小

### 运行中调优：

#### CPU高：

top找到对应的进程

top –Hp 找到对应的线程

jstack查看线程类型，是gc还是业务

如果是业务jstack看云心在哪里

如果是gc，可以jstat –gc查看gc日志+jmap导出堆，看对象

#### 内存高：

top找到对应线程

jmap 导出heap dump

jvisualvm查看那个对象多

##### 导出heap dump问题点：

业务停摆

办法：

1:设置OOM自动倒出

2:多活，切流，jmap倒出

3:jvisivalvm在线看

## 其他知识点：

1:已经被发现没有 gc root 指向的对象，怎么恢复有gc root指向？

在对象的析构函数中，把自己又被其他引用指向了

2:TODO 三色标记

3:TODO 阿里的Arthas