

Java性能优化之JVM GC（垃圾回收机制）



韦庆明

认真，你就赢了

[关注他](#)

258 人赞同了该文章

Java的性能优化，整理出一篇文章，供以后温故知新。

JVM GC（垃圾回收机制）

在学习Java GC 之前，我们需要记住一个单词：**stop-the-world**。它会在任何一种GC算法中发生。stop-the-world 意味着JVM因为需要执行GC而停止了应用程序的执行。当stop-the-world发生时，除GC所需的线程外，所有的线程都进入等待状态，直到GC任务完成。GC优化很多时候就是减少stop-the-world 的发生。

JVM GC回收哪个区域内的垃圾？

需要注意的是，JVM GC只回收**堆区和方法区**内的对象。而栈区的数据，在超出作用域后会被JVM自动释放掉，所以其不在JVM GC的管理范围内。

JVM GC怎么判断对象可以被回收了？

- 对象没有引用
- 作用域发生未捕获异常
- 程序在作用域正常执行完毕
- 程序执行了System.exit()
- 程序发生意外终止（被杀线程等）

在Java程序中不能显式的分配和注销缓存，因为这些事情JVM都帮我们做了，那就是GC。

有些时候我们可以将相关的对象设置成null 来试图显示的清除缓存，但是并不是设置为null 就会一定被标记为可回收，有可能会发生逃逸。

将对象设置成null 至少没有什么坏处，但是使用System.gc() 便不可取了，使用System.gc() 时候并不是马上执行GC操作，而是会等待一段时间，甚至不执行，而且System.gc() 如果被执行，会触发Full GC，这非常影响性能。

JVM GC什么时候执行？

eden区空间不够存放新对象的时候，执行Minor GC。升到老年代的对象大于老年代剩余空间的时候执行Full GC，或者小于的时候被HandlePromotionFailure 参数强制Full GC。调优主要是减



按代的垃圾回收机制

新生代 (Young generation)：绝大多数最新被创建的对象都会被分配到这里，由于大部分在创建后很快变得不可达，很多对象被创建在新生代，然后“消失”。对象从这个区域“消失”的过程我们称之为：**Minor GC**。

老年代 (Old generation)：对象没有变得不可达，并且从新生代周期中存活了下来，会被拷贝到这里。其区域分配的空间要比新生代多。也正由于其相对大的空间，发生在老年代的GC次数要比新生代少得多。对象从老年代中消失的过程，称之为：**Major GC** 或者 **Full GC**。

持久代 (Permanent generation) 也称之为 **方法区 (Method area)**：用于保存类常量以及字符串常量。注意，这个区域不是用于存储那些从老年代存活下来的对象，这个区域也可能发生GC。发生在这个区域的GC事件也被算为 Major GC。只不过在这个区域发生GC的条件非常严苛，必须符合以下三种条件才会被回收：

- 1、所有实例被回收
- 2、加载该类的ClassLoader 被回收
- 3、Class 对象无法通过任何途径访问（包括反射）

可能我们会有疑问：

如果老年代的对象需要引用新生代的对象，会发生什么呢？

为了解决这个问题，老年代中存在一个 **card table**，它是一个512byte大小的块。所有老年代的对象指向新生代对象的引用都会被记录在这个表中。当针对新生代执行GC的时候，只需要查询 card table 来决定是否可以被回收，而不用查询整个老年代。这个 card table 由一个**write barrier** 来管理。write barrier给GC带来了很大的性能提升，虽然由此可能带来一些开销，但完全是值得的。

默认的新生代 (Young generation)、老年代 (Old generation) 所占空间比例为 1 : 2。

新生代空间的构成与逻辑

为了更好的理解GC，我们来学习新生代的构成，它用来保存那些第一次被创建的对象，它被分成三个空间：

- 一个伊甸园空间 (Eden)
- 两个幸存者空间 (From Survivor、To Survivor)

默认新生代空间的分配：Eden : From : To = 8 : 1 : 1

每个空间的执行顺序如下：

- 2、在伊甸园空间执行第一次GC（Minor GC）之后，存活的对象被移动到其中一个幸存者空间（Survivor）。
- 3、此后，每次伊甸园空间执行GC后，存活的对象会被堆积在**同一个幸存者空间**。
- 4、当一个幸存者空间饱和，还在存活的对象会被移动到另一个幸存者空间。然后会清空已经饱和的哪个幸存者空间。
- 5、在以上步骤中重复N次（N = MaxTenuringThreshold（年龄阈值设定，默认15））依然存活的对象，就会被移动到老年代。

从上面的步骤可以发现，**两个幸存者空间，必须有一个是保持空的**。如果两个两个幸存者空间都有数据，或两个空间都是空的，那一定是你的系统出现了某种错误。

我们需要重点记住的是，**对象在刚刚被创建之后，是保存在伊甸园空间的（Eden）**。那些长期存活的对象会经由幸存者空间（Survivor）转存到老年代空间（Old generation）。

也有例外出现，对于一些比较大的对象（需要分配一块比较大的连续内存空间）则直接进入老年代。一般在Survivor 空间不足的情况下发生。

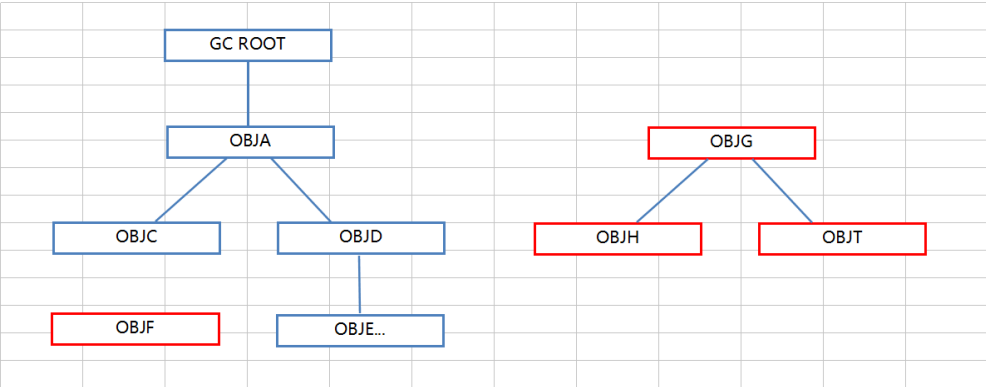
老年代空间的构成与逻辑

老年代空间的构成其实很简单，它不像新生代空间那样划分为几个区域，它只有一个区域，里面存储的对象并不像新生代空间绝大部分都是朝闻道，夕死矣。**这里的对象几乎都是从Survivor 空间中熬过来的，它们绝不会轻易的狗带**。因此，Full GC（Major GC）发生的次数不会有Minor GC 那么频繁，并且做一次Major GC 的时间比Minor GC 要更长（约10倍）。

JVM GC 算法讲解

1、根搜索算法

根搜索算法是从离散数学中的图论引入的，程序把所有引用关系看作一张图，从一个节点GC ROOT 开始，寻找对应的引用节点，找到这个节点后，继续寻找这个节点的引用节点。当所有的引用节点寻找完毕后，剩余的节点则被认为是没有被引用到的节点，即无用的节点。

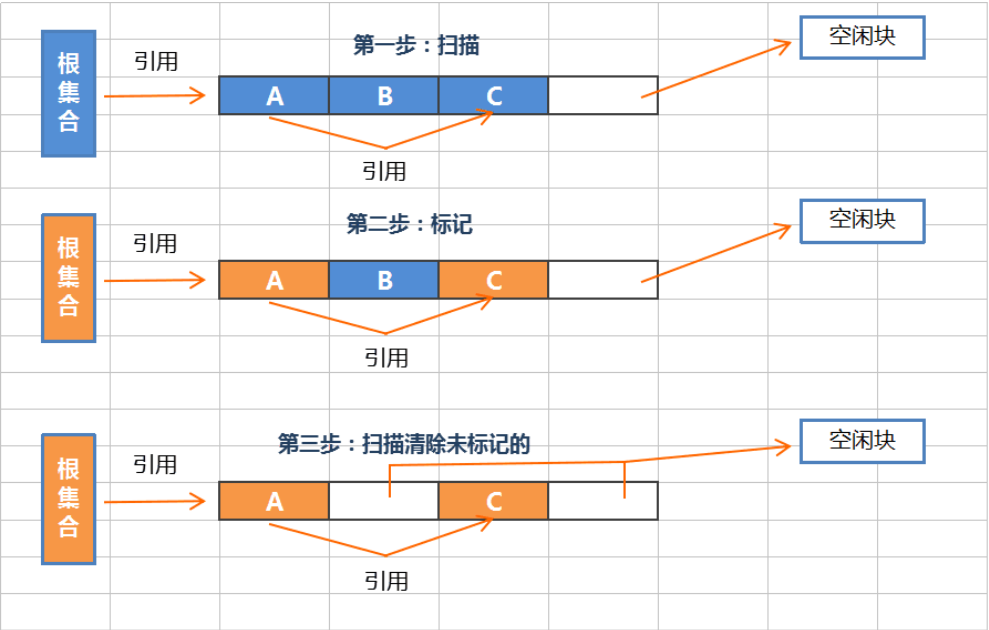


上图红色为无用的节点，可以被回收。

- 2、方法区中静态属性引用的对象
- 3、方法区中常亮引用的对象
- 4、本地方法栈中引用的对象（Native对象）

基本所有GC算法都引用根搜索算法这种概念。

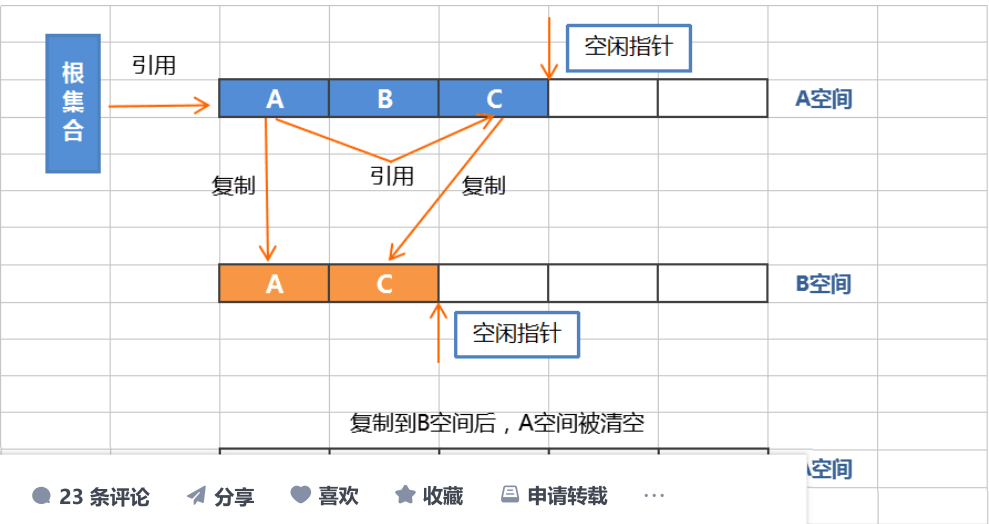
2、标记 - 清除算法



标记-清除算法采用从根集合进行扫描，**对存活的对象进行标记**，标记完毕后，再扫描整个空间中未被标记的对象进行直接回收，如上图。

标记-清除算法不需要进行对象的移动，并且仅对不存活的对象进行处理，在存活的对象比较多的情况下极为高效，但由于标记-清除算法直接回收不存活的对象，并没有对还存活的对象进行整理，因此会导致内存碎片。

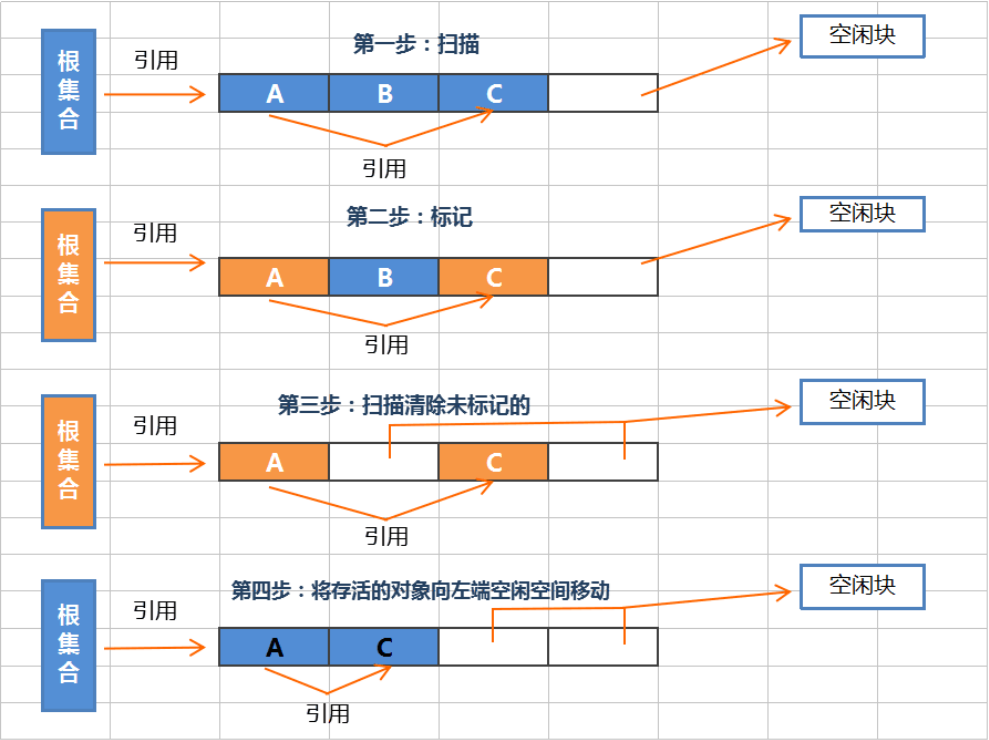
3、复制算法



复制算法采用从根集合扫描，将存活的对象复制到空闲区间，当扫描完毕活动区间后，会将活动区间一次性全部回收。**此时原本的空闲区间变成了活动区间。**下次GC时候又会重复刚才的操作，以此循环。

复制算法在存活对象比较少的时候，极为高效，但是带来的成本是牺牲一半的内存空间用于进行对象的移动。**所以复制算法的使用场景，必须是对对象的存活率非常低才行，**而且最重要的是，我们需要克服50%内存的浪费。

4、标记 - 整理算法



标记-整理算法采用 标记-清除 算法一样的方式进行对象的标记、清除，但在回收不存活的对象占用的空间后，会将所有存活的对象往左端空闲空间移动，并更新对应的指针。标记-整理 算法是在标记-清除 算法之上，又进行了对象的移动排序整理，因此成本更高，但却解决了内存碎片的问题。

JVM为了优化内存的回收，使用了分代回收的方式，对于新生代内存的回收（Minor GC）主要采用**复制算法**。而对于老年代的回收（Major GC），大多采用**标记-整理算法**。

垃圾回收器简介

需要注意的是，**每一个回收器都存在Stop The World 的问题**，只不过各个回收器在Stop The World 时间优化程度、算法的不同，可根据自身需求选择适合的回收器。

1、Serial (-XX:+UseSerialGC)

知乎

器并不是只能使用一个CPU进行收集，而是当JVM需要进行垃圾回收的时候，需暂停所有的用户线程，直到回收结束。

使用算法：**复制算法**



JVM中文名称为Java虚拟机，因此它像一台虚拟的电脑在工作，而其中的每一个线程都被认为是JVM的一个处理器，因此图中的CPU0、CPU1实际上为用户的线程，而不是真正的机器CPU，不要误解哦。

Serial收集器虽然是最老的，但是它对于限定单个CPU的环境来说，由于没有线程交互的开销，专心做垃圾收集，所以它在这种情况下是相对于其他收集器中最高效的。

2、SerialOld (-XX:+UseSerialGC)

SerialOld是**Serial收集器的老年代收集器版本**，它同样是一个单线程收集器，这个收集器目前主要用于Client模式下使用。如果在Server模式下，它主要还有两大用途：一个是在JDK1.5及之前的版本中与Parallel Scavenge收集器搭配使用，另外一个就是作为CMS收集器的后备预案，如果CMS出现Concurrent Mode Failure，则SerialOld将作为后备收集器。

使用算法：**标记 - 整理算法**

运行示意图与上图一致。

3、ParNew (-XX:+UseParNewGC)

ParNew其实就是Serial收集器的多线程版本。除了Serial收集器外，只有它能与CMS收集器配合工作。

使用算法：**复制算法**

ParNew是许多运行在Server模式下的JVM首选的**新生代收集器**。但是在单CPU的情况下，它的效率远远低于Serial收集器，所以一定要注意使用场景。

4、ParallelScavenge (-XX:+UseParallelGC)



知乎

ParallelScavenge收集器的目标是达到一个可控件的吞吐量，所谓吞吐量就是CPU用于运行用户代码的时间与CPU总消耗时间的比值，即吞吐量 = 运行用户代码时间 / （运行用户代码时间 + 垃圾收集时间）。如果虚拟机总共运行了100分钟，其中垃圾收集花了1分钟，那么吞吐量就是99%。

5、ParallelOld (-XX:+UseParallelOldGC)

ParallelOld是并行收集器，和SerialOld一样，**ParallelOld是一个老年代收集器**，是老年代吞吐量优先的一个收集器。这个收集器在JDK1.6之后才开始提供的，在此之前，ParallelScavenge只能选择SerialOld来作为其老年代的收集器，这严重拖累了ParallelScavenge整体的速度。而ParallelOld的出现后，“吞吐量优先”收集器才名副其实！

使用算法：**标记 - 整理算法**

在注重吞吐量与CPU数量大于1的情况下，都可以优先考虑ParallelScavenge + ParallelOld收集器。

6、CMS (-XX:+UseConcMarkSweepGC)

CMS是一个**老年代收集器**，全称 Concurrent Low Pause Collector，是JDK1.4后期开始引用的新GC收集器，在JDK1.5、1.6中得到了进一步的改进。**它是对于响应时间的重要性需求大于吞吐量要求的收集器**。对于要求服务器响应速度高的情况下，使用CMS非常合适。

CMS的一大特点，**就是用两次短暂的暂停来代替串行或并行标记整理算法时候的长暂停**。

使用算法：**标记 - 清理**

CMS的执行过程如下：

• 初始标记 (STW initial mark)

在这个阶段，需要虚拟机停顿正在执行的应用线程，官方的叫法STW (Stop The World)。这个过程从根对象扫描**直接关联**的对象，并作标记。这个过程会很快的完成。

• 并发标记 (Concurrent marking)

这个阶段紧随初始标记阶段，在“初始标记”的基础上继续向下追溯标记。**注意这里是并发标记**，表示用户线程可以和GC线程一起并发执行，这个阶段不会暂停用户的线程哦。

• 并发预清理 (Concurrent precleaning)



知乎

• 重新标记 (STW remark)

这个阶段会再次暂停正在执行的应用线程，重新重根对象开始查找并标记**并发阶段**遗漏的对象（在并发标记阶段结束后对象状态的更新导致），并处理对象关联。这一次耗时会比“初始标记”更长，并且这个阶段可以并行标记。

• 并发清理 (Concurrent sweeping)

这个阶段是**并发**的，应用线程和GC清除线程可以一起并发执行。

• 并发重置 (Concurrent reset)

这个阶段任然是**并发**的，重置CMS收集器的数据结构，等待下一次垃圾回收。

CMS的缺点：

1、内存碎片。由于使用了 标记-清理 算法，导致内存空间中会产生内存碎片。不过CMS收集器做了一些小的优化，就是把未分配的空间汇总成一个列表，当有JVM需要分配内存空间的时候，会搜索这个列表找到符合条件的空间来存储这个对象。但是内存碎片的问题依然存在，如果一个对象需要3块连续的空间来存储，因为内存碎片的原因，寻找不到这样的空间，就会导致Full GC。

2、需要更多的CPU资源。由于使用了并发处理，很多情况下都是GC线程和应用线程并发执行的，这样就需要占用更多的CPU资源，也是牺牲了一定吞吐量的原因。

3、需要更大的堆空间。因为CMS标记阶段应用程序的线程还是执行的，那么就会有堆空间继续分配的问题，为了保障CMS在回收堆空间之前还有空间分配给新加入的对象，必须预留一部分空间。CMS默认在老年代空间使用68%时候启动垃圾回收。可以通过-XX:CMSInitiatingOccupancyFraction=n来设置这个阈值。

7、GarbageFirst (G1)

这是一个新的垃圾回收器，既可以回收新生代也可以回收老年代，SunHotSpot1.6u14以上EarlyAccess版本加入了这个回收器，Sun公司预期SunHotSpot1.7发布正式版本。通过重新划分内存区域，整合优化CMS，同时注重吞吐量和响应时间。杯具的是Oracle收购这个收集器之后将其用于商用收费版收集器。因此目前暂时没有发现哪个公司使用它，这个放在之后再去研究吧。

整理一下新生代和老年代的收集器。

新生代收集器：

Serial (-XX:+UseSerialGC)

ParNew (-XX:+UseParNewGC)

ParallelScavenge (-XX:+UseParallelGC)

G1 收集器



SerialOld (-XX:+UseSerialOldGC)

ParallelOld (-XX:+UseParallelOldGC)

CMS (-XX:+UseConcMarkSweepGC)

G1 收集器

目前了解的GC收集器就是这么多了，如果有哪位有缘的朋友看到了这篇文章，恰好有更好的GC收集器推荐，欢迎留言交流。

参考文章：

[成为JavaGC专家Part I -- 深入浅出Java垃圾回收机制](#)

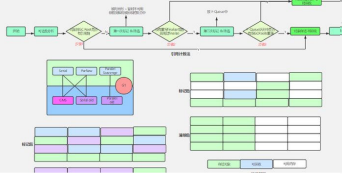
[jvm垃圾回收是什么时候触发的？垃圾回收算法？都有哪些垃圾回收器](#)

[了解CMS\(Concurrent Mark-Sweep\)垃圾回收器](#)

编辑于 2017-03-07 13:47

[Java](#) [Java 虚拟机 \(JVM\)](#) [java GC](#)

推荐阅读



谈到JVM的垃圾回收(GC)，可能这才是你想了解的！

不秃顶的程... 发表于Java架...

JVM系列(四) - JVM垃圾回收算法

前言前面介绍了 Java内存运行时区域，其中程序计数器、虚拟机栈、本地方法栈 三个区域随线程而生，随线程而灭；栈中的栈帧随着方法的进入和退出而有条不紊地执行着出栈和入栈操作。每一个栈...

零壹技术栈 发表于JAVA虚...



Java GC回收算法-判定一个对象是否可以回收

Mr羽墨青... 发表于Java生...

23 条评论

切换为时间排序

写下你的评论...



木女孩

2017-03-27

你这是自己写的？能不能稍微更新一下？比如 HandlePromotionFailure 参数早就废了，比如G1收集器 “Sun公司预期SunHotSpot1.7发布正式版本” 别预期了，这玩意儿JDK7u4正

赞同 258

23 条评论

分享

喜欢

收藏

申请转载

...

知乎

跟我之前翻译的一篇很相似。个人觉得还是声明下版权的比较好。

1

韦庆明 (作者) 回复 木女孩 2017-03-27

谢谢你的补充修正，这是我整理网上的一些文章结合自己的理解。近期在看JVM的相关书籍，发现文章中有一些缺漏和过期之处，我会尽快的再次整理更新。

1

展开其他 1 条回复

爱吃小肥羊 2017-03-27

和《深入理解 JVM》极度相似

3

Blank 2017-03-27

干货满满，挺适合需要快速概括了解JVM垃圾回收的人，比如我

2

懒散的猫 2017-03-27

复制收集算法实际并不是浪费50%空间吧..比如eden:survivor 8:1嘛

2

李浩宇Alex 回复 懒散的猫 2021-04-17

按8：1的比例其实只是百分之10的浪费

赞

planenalq 2017-03-26

要声明这是hotspot的内存划分

2

乌合之众 2018-10-12

还有就是标记整理算法，并不是和标记清除算法一样标记废弃对象并直接清除，而是标记废除对象后，直接对存活对象向内存中一段移动，然后找到该移动后存活对象的连续地址空间的边界值，并更新地址指针，然后清除掉该指针另一边界内存空间的所有对象。

1

乌合之众 2018-10-12

由于赞很多，所以还是建议修改一下文中的错误地方，比如标记清除算法，标记的是垃圾对象。而不是存活对象

1

Dustin Gao 2017-08-30

4、当一个幸存者空间饱和，还在存活的对象会被移动到另一个幸存者空间。然后会清空已经饱和的哪个幸存者空间。

被清空的空间里的引用去哪了？是去了另一个幸存者空间吗？那另一个幸存者空间不是会一直都满的吗？

1

丹丹 回复 Dustin Gao 2020-01-17

一个幸存者A饱和了 就需要把这个饱和的幸存者A进行挪动 挪动到另外一个空的幸存者B 挪动期间 清空了已经狗带的 保留下来还没狗带的到B中 A就清空了 后续就往B中继续加 幸存下来的



知乎

千白 J

1



丁春秋乐队

2017-03-28

表示看着看着就不懂了

1



林小虎

2017-03-28

点个赞顺便收藏了吧，假装看懂了

1



Littlerunner

2017-03-27

很全，给力

1



猫头鹰

2017-03-26

马克

1



丹丹

2020-01-15

mark一下吧

赞



时空禁区

2019-03-03

楼主好，我有两个小疑问。

1:文章里说的标记--清除算法是标记存活对象，清除没标记的死亡对象，我的理解是标记不可达的对象，且不会立即回收，有必要执行finalize方法则被标记的对象放到回收队列等待执行回收，期间该对象如果又和GC root建立引用则不会被回收，否则回收。

2:标记整理算法，文章说是先回收再整理。我的理解是先把存活对象往一边移，再从存活对象边界把剩余的部分清空。

赞



fjsh大鹏展翅

2017-06-18

最近正好整理了下这方面的资料mp.weixin.qq.com/s/NJm2...

拿走不谢，如果是土豪给个赏赞

赞



kilig

2021-05-18

你这是误人子弟啊兄弟，赶紧改了吧

标记-清除”（Mark-Sweep）算法，如同它的名字一样，算法分为“标记”和“清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收所有被标记的对象

“标记-整理”（Mark-Compact）算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存

赞



无情便是清明

2020-12-13

Minor gc

赞

