

大家好，上一小节中我们主要介绍了JVM内存机制的基础知识点，包括内存的分配以及回收策略等。本小节我们主要介绍垃圾回收算法以及垃圾收集器，并且对当前使用较广的CMS垃圾收集器做了较为详细的阐述。垃圾回收算法是JVM相关技术考察中的高频考点，希望大家可以有效理解与掌握，在面试中清晰阐述。

有效掌握JVM内存相关技术原理对于我们的日常开发工作也有很大的帮助。好了，让我们一起来学习吧~

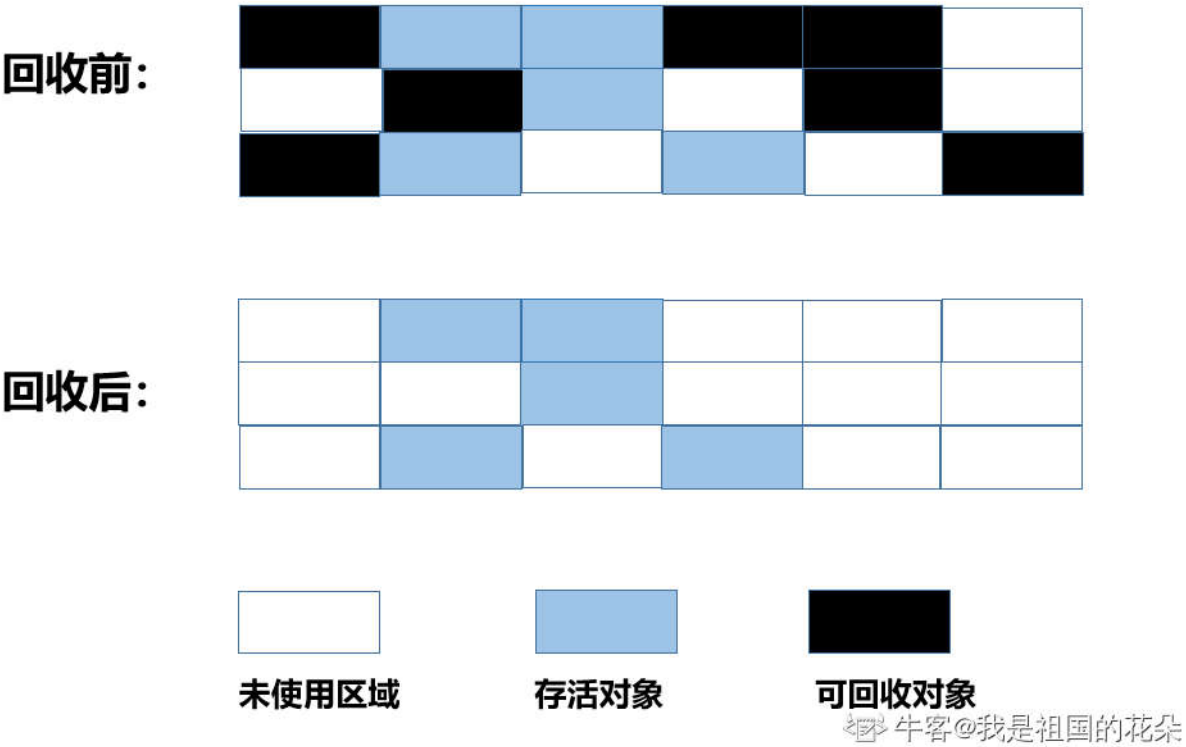
(1) JVM垃圾回收算法有哪些？（重点掌握）

答：HotSpot 虚拟机采用了root根搜索方法来进行内存回收，常见的回收算法有**标记-清除算法**，**复制算法**和**标记整理算法**。

标记-清除算法（Mark-Sweep）：

标记-清除算法执行分两阶段。第一阶段从引用根节点开始标记所有被引用的对象，第二阶段遍历整个堆，把未标记的对象清除。此算法需要暂停整个应用，并且会产生内存碎片。

标记-清除算法示意图：



复制算法：

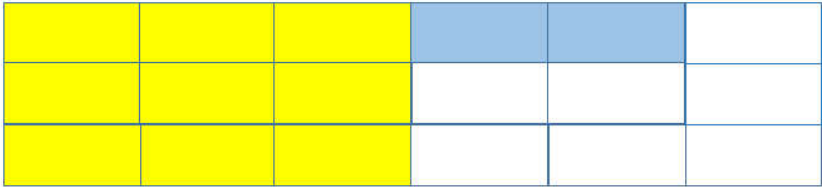
复制算法把内存空间划为两个相等的区域，每次只使用其中一个区域。垃圾回收时，遍历当前使用区域，把正在使用中的对象复制到另外一个区域中。复制算法每次只处理正在使用中的对象，因此复制成本比较小，同时复制过去以后还能进行相应的内存整理，不会出现“碎片”问题。当然，此算法的缺点也是很明显的，就是需要两倍内存空间。

复制算法示意图：

回收前：



回收后：



牛客@我是祖国的花朵

6

标记-整理算法：

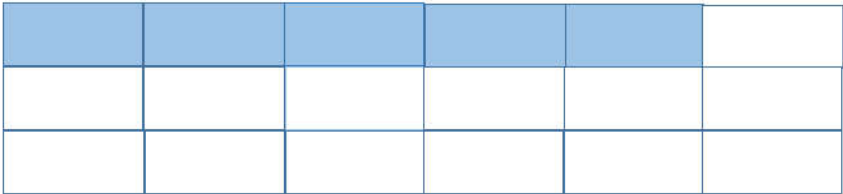
标记-整理算法结合了“标记-清除”和“复制”两个算法的优点。也是分两阶段，第一阶段从根节点开始标记所有被引用对象，第二阶段遍历整个堆，清除未标记对象并且把存活对象“压缩”到堆的其中一块，按顺序排放。此算法避免了“标记-清除”的碎片问题，同时也避免了“复制”算法的空间问题。

标记-整理算法示意图：

回收前：



回收后：



牛客@我是祖国的花朵

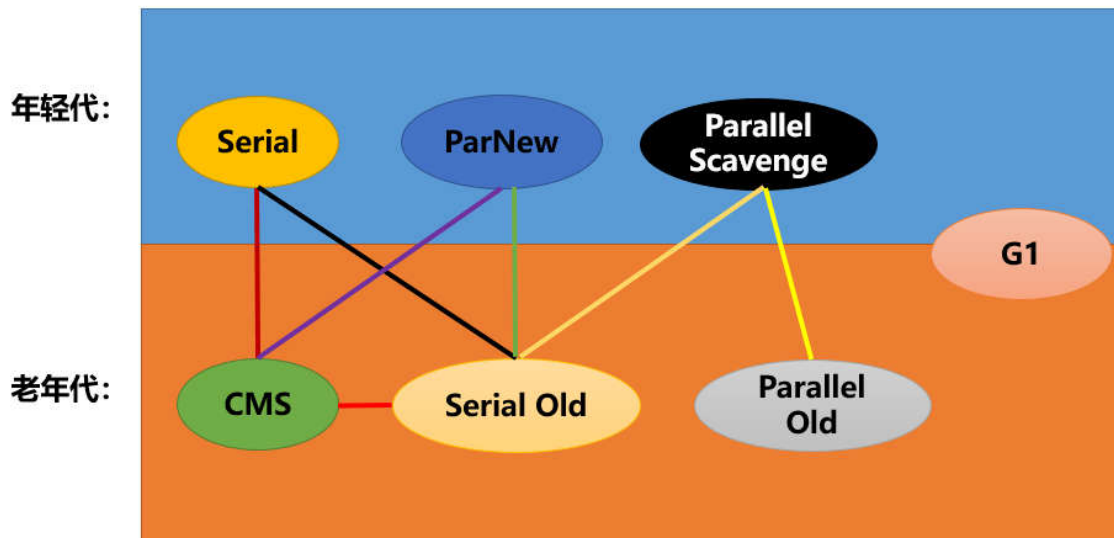
解析:

垃圾回收算法是垃圾收集器的算法实现基础，年轻代垃圾回收一般采用复制算法，老年代垃圾回收一般采用标记-清除和标记-整理算法。希望大家对照着算法示意图，对算法的原理与过程加以理解与掌握。接下来，我们一起来看看垃圾回收算法的具体实现，那就是垃圾收集器吧。

6

## (2) JVM中的垃圾收集器有了解吗？（重点掌握CMS收集器）

答：JVM中的垃圾收集器主要包括7种，即Serial, Serial Old, ParNew, Parallel Scavenge, Parallel Old以及CMS, G1收集器。如下图所示：



### HotSpot虚拟机的垃圾收集器

牛客@我是祖国的花朵

**Serial收集器：**

Serial收集器是一个单线程的垃圾收集器，并且在执行垃圾回收的时候需要 Stop The World。虚拟机运行在Client模式下的默认新生代收集器。Serial收集器的优点是简单高效，对于限定在单个CPU环境来说，Serial收集器没有多线程交互的开销。

**Serial Old收集器：**

Serial Old是Serial收集器的老年代版本，也是一个单线程收集器。主要也是给在Client模式下的虚拟机使用。在Server模式下存在主要是做为CMS垃圾收集器的后备预案，当CMS并发收集发生Concurrent Mode Failure时使用。

**ParNew收集器：**

ParNew是Serial收集器的多线程版本，新生代是并行的（多线程的），老年代是串行的（单线程的），新生代采用复制算法，老年代采用标记整理算法。可以使用参数：**-XX: UseParNewGC使用该收集器，使用 -XX: ParallelGCThreads可以限制线程数量。**

**Parallel Scavenge垃圾收集器：**

Parallel Scavenge是一种新生代收集器，使用复制算法的收集器，而且是**并行的多线程收集器**。Parallel收集器特点是更加关注吞吐量（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）。可以通过-

**XX:MaxGCPauseMillis**参数控制最大垃圾收集停顿时间；通过**-XX:GCTimeRatio**参数直接设置吞吐量大小；通过**-XX:+UseAdaptiveSizePolicy**参数可以打开GC自适应调节策略，该参数打开之后虚拟机会根据系统的运行情况收集性能监控信息，动态调整虚拟机参数以提供最合适的停顿时间或者最大的吞吐量。**自适应调节策略**是Parallel Scavenge收集器和ParNew的主要区别之一。

6

### Parallel Old收集器：

Parallel Old是Parallel Scavenge收集器的老年代版本，使用多线程和标记-整理算法。

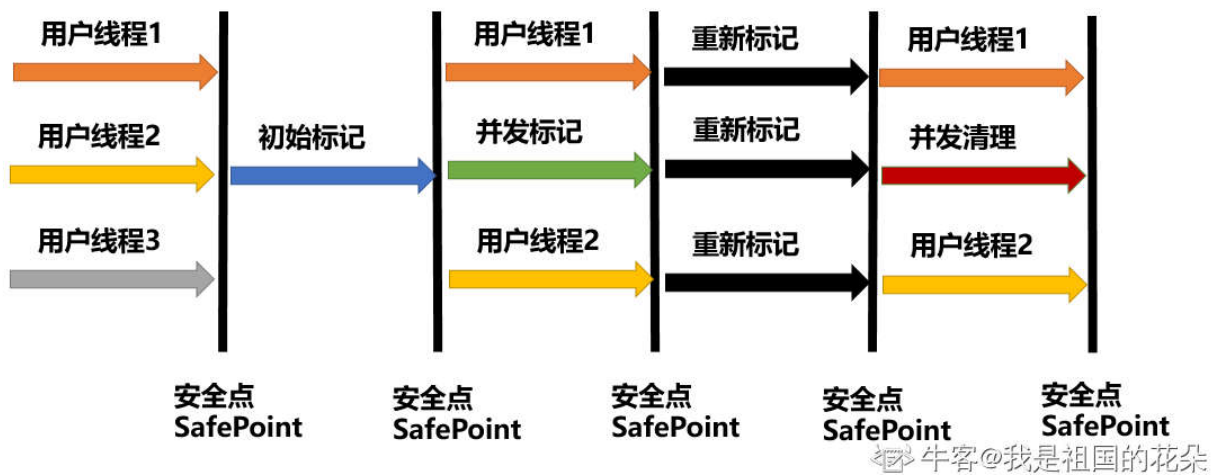
### CMS（Concurrent Mark Sweep）收集器：

CMS收集器是一种以**获取最短回收停顿时间**为目标的收集器。CMS收集器是基于**标记-清除算法**实现的，是一种老年代收集器，通常与ParNew一起使用。

#### CMS的垃圾收集过程分为4步：

- **初始标记**：需要“Stop the World”，初始标记仅仅只是标记一下GC Root能直接关联到的对象，速度很快。
- **并发标记**：是主要标记过程，这个标记过程是和用户线程并发执行的。
- **重新标记**：需要“Stop the World”，为了修正并发标记期间因用户程序继续运作而导致标记产生变动的那一部分对象的标记记录（停顿时间比初始标记长，但比并发标记短得多）。
- **并发清除**：和用户线程并发执行的，基于标记结果来清理对象。

#### CMS垃圾回收过程示意图：



那么问题来了，如果在重新标记之前刚好发生了一次MinorGC，会不会导致重新标记阶段Stop the World时间太长？

答：不会的，在并发标记阶段其实还包括了一次并发的**预清理阶段**，虚拟机会主动**等待年轻代发生垃圾回收**，这样可以将重新标记对象引用关系的步骤放在并发标记阶段，有效降低重新标记阶段Stop The World的时间。

#### CMS垃圾回收器的优缺点分析：

CMS以降低垃圾回收的停顿时间为目的，很显然其具有并发收集，停顿时间低的优点。

缺点主要包括如下：

- **对CPU资源非常敏感**，因为并发标记和并发清理阶段和用户线程一起运行，当CPU数变小时，性能容易出现问題。
- 收集过程中会产生**浮动垃圾**，所以不可以在老年代内存不够用了才进行垃圾回收，必须提前进行垃圾收集。通过参数-XX:CMSInitiatingOccupancyFraction的值来控制内存使用百分比。如果该值设置的太高，那么在CMS运行期间预留的内存可能无法满足程序所需，会出现**Concurrent Mode Failure失败**，之后会临时使用Serial Old收集器做为老年代收集器，会产生更长时间的停顿。
- **标记-清除方式会产生内存碎片**，可以使用参数-XX: UseCMSCompactAtFullCollection来控制是否开启内存整理（无法并发，默认是开启的）。参数-XX:CMSFullGCsBeforeCompaction用于设置执行多少次不压缩的Full GC后进行一次带压缩的内存碎片整理（默认值是0）。

接下来，我们先看下上边介绍的浮动垃圾是怎么产生的吧。

**浮动垃圾：**

由于在应用运行的同时进行垃圾回收，所以有些垃圾可能在垃圾回收进行完成时产生，这样就造成了“**Floating Garbage**”，这些垃圾需要在下次垃圾回收周期时才能回收掉。所以，**并发收集器一般需要20%的预留空间**用于这些浮动垃圾。

**G1（Garbage-First）收集器：**

G1收集器将新生代和老年代取消了，取而代之的是**将堆划分为若千的区域**，仍然属于分代收集器，区域的一部分包含新生代，新生代采用复制算法，老年代采用标记-整理算法。

通过**将JVM堆分为一个个的区域（region）**，G1收集器可以避免在Java堆中进行全区域的垃圾收集。G1跟踪各个region里面的垃圾堆积的价值大小（回收所获得的空间大小以及回收所需时间的经验值），在后台维护一个优先列表，每次**根据回收时间来优先回收价值最大的region**。

**G1收集器的特点：**

- **并行与并发**：G1能充分利用多CPU，多核环境下的硬件优势，来缩短Stop the World，是并发的收集器。
- **分代收集**：G1不需要其他收集器就能独立管理整个GC堆，能够采用不同的方式去处理新建对象、存活一段时间的对象和熬过多次GC的对象。
- **空间整合**：G1从整体来看是基于标记-整理算法，从局部（两个Region）上看基于复制算法实现，G1运作期间不会产生内存空间碎片。
- **可预测的停顿**：能够建立可以预测的停顿时间模型，预测停顿时间。

**和CMS收集器类似，G1收集器的垃圾回收工作也分为了四个阶段：**

- 初始标记
- 并发标记
- 最终标记
- 筛选回收

其中，筛选回收阶段首先对各个Region的回收价值和成本进行计算，根据用户期望的GC停顿时间来制定回收计划。

**解析：**

这块关于垃圾回收器的知识点相对较多，我们重点介绍了CMS垃圾收集器，并且**CMS也确实是我们服务中最常使用的垃圾收集器**。利用CMS并发标记和清理的特性，**可以有效降低用户的停顿时间，对于服务的稳定性有一个非常显著的提升**。通常，作者本人设置的JVM启动参数如下，学习了上边的内容，聪明的你一定可以明白设置的含义啦。

1

JAVA\_OPTS="-Xms4096m -Xmx4096m -XX:NewRatio=2 -XX:SurvivorRatio=8 -Xloggc:/home/work/log/ser

## 总结:

本小节在上一小节JVM相关基础技术的基础上，重点介绍了垃圾回收算法和垃圾收集器的实现。垃圾收集器需要重点掌握CMS垃圾收集器，面试中可以准确阐述CMS相关知识点是一个亮点，显示出自己对JVM内存相关技术的熟练掌握。下一小节，我们将阐述JVM相关的内存调优命令，并且给出如何排查线上服务故障的步骤，并且还会介绍类加载机制等相关技术原理。

限于作者水平，文章中难免会有不妥之处。大家在学习过程中遇到我没有表达清楚或者表述有误的地方，欢迎随时在文章下边指出，我会及时关注，随时改正。另外，大家有任何话题都可以在下边留言，我们一起交流探讨。

讨论


评论

柳杰2019050110494201#  
这篇没懂  
发表于 2020-01-01 16:17:02赞(0) 回复(0)


刘畅2019042116068162#  
打卡，这块我还要好好练一练  
发表于 2020-01-03 19:39:18赞(0) 回复(0)

牧水s N3#  
打卡  
发表于 2020-01-18 20:45:55赞(0) 回复(0)

snailClimbZG4#  
打卡  
发表于 2020-02-24 21:52:48赞(0) 回复(0)

Ruoji555555#  
垃圾收集器总结：最初使用Serial+Serial Old收集垃圾，最简单，两者都是单线程的，所以只适合少内存使用。随着内存增大，开始使用Parallel Scavenge +Parallel Old，这两个其实就是前面两个Serial的多线程版本，性能更好一点，在JDK1.6-1.8中作为默认垃圾回收器。随着内存进一步增大，出现了ParNew+CMS的组合，其中ParNew是Parallel Scavenge为了配合CMS出现的改进版本，CMS是并发标记清除。看似性能更好，实则存在巨大的缺陷：CMS会导致大量的内存碎片，而内存碎片太多的时候，会使用Serial Old这个单线程的收集器进行垃圾收集（雾）....从JDK1.7之后出现了G1垃圾收集器，在JDK1.8之后开始完善，它支持更大的内存（大概几百G），特点是逻辑分代，物理不分代，它的stop-the-world时间可以小于200ms。还有一个在JDK11中推出的垃圾回收器ZGC，还正在开发，它的逻辑、物理都不分代，而且能支持16T的内存，传说停顿时间只有1ms（\*\*\*C++!）  
发表于 2020-02-29 10:39:41赞(5) 回复(2)

Ruoji555555：牛客不让说C++的坏话2333333  
2020-02-29 10:40:31赞(1) 回复(0)

我是祖国的花朵  作者： CMS是在并发收集失败的情况下，才会使用Serial Old收集器来进行回收（stop the world），一般情况下，可以设置内存使用70%或者80%即开始老年代回收。

2020-03-15 17:44:58 赞(0) 回复(0)

请输入你的观点

6


回复



爱生活的猿 6#

最重点的是。。说这几种算法都没用，面试官问你JVM的垃圾回收算法是什么，标准答案应该是：JVM这三种都不用，用的是分代回收算法。然后再详细阐述年轻代有哪种算法，老年代有哪种算法，这样回答才是满分。

发表于 2020-03-11 09:56:17 赞(2) 回复(1)

我是祖国的花朵  作者： 分代收集是从分区对待的方面来看的，上边的三种策略是从基本的垃圾回收算法层面说的。

2020-03-15 17:40:49 赞(0) 回复(0)

请输入你的观点

回复