**JVM垃圾回收器之CMS回收过程及优缺点**

[](https://www.jianshu.com/u/bf3f617d748a)

[是残月啊](https://www.jianshu.com/u/bf3f617d748a)关注

0.0752020.07.12 05:54:09字数 1,684阅读 2,081

**JVM垃圾回收器之CMS回收过程及优缺点**

今天给大家介绍一下CMS垃圾回收器。首先CMS是一款多线程+分段操作的一款垃圾回收器。其最大的优点就是将一次完整的回收过程拆分成多个步骤，并且在执行的某些过程中可以使用户线程可以继续运行。也就是说CMS可以减少STW的时间对我们的用户体验会非常好，所以CMS垃圾回收器也是现在大部分企业所用到的最多的一款垃圾回收器。（CMS只管老年代）  
接下来我会给大家从浅到深讲解一下CMS垃圾回收器！

**一.CMS垃圾回收器的使用与执行过程**

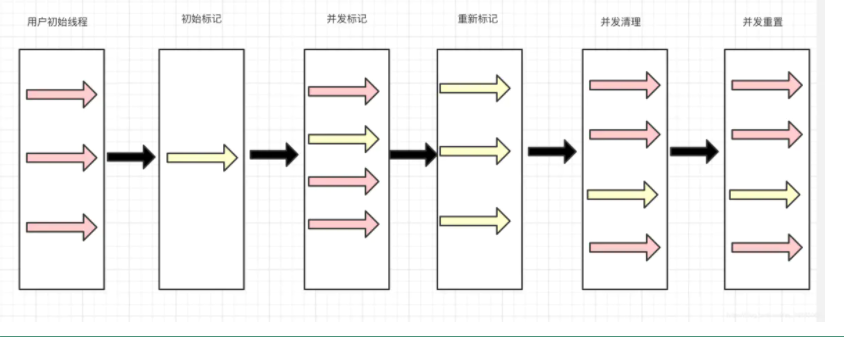
想要使用CMS垃圾回收器非常简单，我们只需要在JVM启动参数上加上一条

-XX:+UseConcMarkSweeoGC

即可让JVM使用CMS作为老年代的垃圾管理器。

接下来是我们的CMS执行过程，下面给大家放一张图

粉色代表用户线程，黄色代表CMS线程

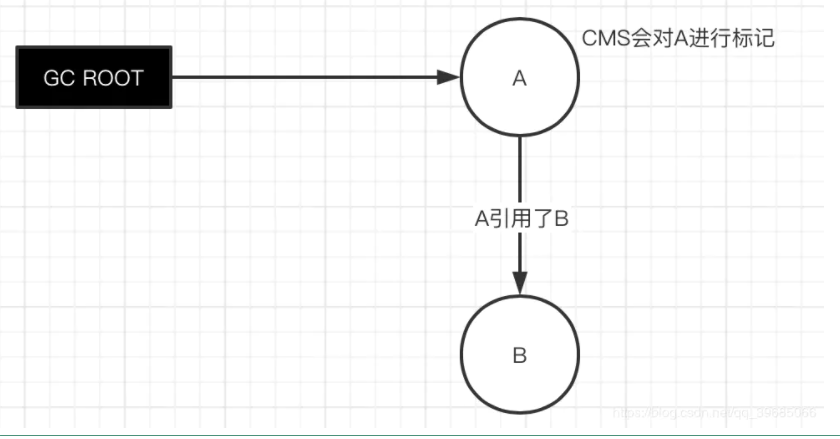


在这里插入图片描述

在上图中我们可以看到在CMS回收器对垃圾回收的时候一种经过了5个步骤（用户线程除外），分别有初始标记，并发标记，重新标记，并发清理和并发重置。那么他们是什么意思又分别做了哪些事情呢？别着急，接下来我就给大家解释一下每个步骤之间所做的事情

**1.初始标记：**

初始标记其实就是对被我们GC ROOT直接引用的对象做一个标记，而在这个过程中将会触发一次STW机制。  
如下图所示：



在这里插入图片描述

在这个过程中，虽然会触发STW机制，但是时间会非常的短

**2.并发标记**

在进行并发标记的过程中，我们的用户线程和CMS线程会一起执行。CMS所做的一件事情就是把堆里的所有引用对象全部找到并做标记。  
但是在这个过程中可能会发生对象状态被改变的问题。  
比如我的一个对象的引用链已经断开，变成了垃圾对象，但是CMS已经对他做过标记判断为非垃圾对象了怎么办？（多标问题）  
又比如本来一个对象在CMS标记的过程中把他标记成了垃圾对象但是后来我们有引用了，结果在我们用的时候垃圾对象已经被干掉了，那我们是不是在引用这个对象的时候就会找不到这个垃圾对象。这时候我们的第三步就产生了。（漏标问题）

**3.重新标记**

在这一步，CMS会触发STW机制，并修复并发标记状态已经改变的对象，但是这个过程会比较漫长。他利用三色标记和增量更新来解决我们的漏标问题，之后我会讲讲三色标记和增量更新并通过Hotspot源码来让大家对底层有一个更加深入的了解。

**4.并发清理**

那这一步就很好理解了，所谓的并发清理其实就是对没有被做标记的对象进行一个清理回收，在这个过程中同样不会产生STW。

**5.并发重置**

重置本次GC过程中的标记数据

通过上面的介绍之后，相信大家对CMS垃圾回收器的回收步骤有了一个清晰的认知。虽然CMS看着很牛逼，例如多线程操作，例如STW时间短，但是不管是什么东西有优点就有缺点，那CMS的缺点有哪些呢？我来跟你聊聊

**二.CMS垃圾回收器的缺点**

**1.回收时间长，吞吐量不如Parallel**

由于我们在执行CMS垃圾回收器的过程中有一部分资源让给了用户线程，那就会导致回收时间长，内存空间没有被及时释放掉也就会导致吞吐量不如Paralle

**2.无法处理浮动垃圾**

还记得刚刚给大家埋了两个坑吗，我们刚刚在重新标记里讲过，三色标记和增量更新可以解决漏标问题。那么多标实际上到最后都变成了浮动垃圾。浮动垃圾只有在执行下一次垃圾回收的时候才会被真正回收掉。

**3.会产生大量的空间碎片**

为什么这么说呢？因为我们的CMS垃圾回收器的算法采用的是标记清除算法，这个算法的弊端就是会产生空间随便，造成空间不连续。但是没关系CMS提供了两个参数，解决了这个问题

-XX:+UseCMSCompactAtFullCollection

这个参数可以开启我们的内存空间整理，使我们的空间整齐划一。通常情况下我们会配合另一个参数来使用

-XX:CMSFullGCsBeforeCompaction

这个参数的意思是，在CMS执行多少次full gc之后进行空间整理，这个参数不填也没关系，他默认的是0次，也就是说，在每一次垃圾回收之后，都会整理一次内存空间。

**4.并发模式失败：Concurrent model failure**

什么意思呢？大家思考一下，当我们的老年代内存空间满了之后，虽然这时候在做FULL GC，但是在FULL GC的过程中有一个新的对象进来了怎么办？  
此时会进入STW状态，并且CMS会自动切换到用Serial old垃圾收集器来回收。Serial我们都知道，它是一个单线程的垃圾回收器。那这种情况出现是不是会严重降低我们的执行效率？  
那么我们为了解决这个问题，可以通过调整老年代空间被占满了多少之后触发FULL GC

-XX:CMSInitiatingOccupancyFraction

参数如上，通过这个参数可以调整触发full gc的百分比,默认是92%

好了，文章到这里就结束了，在下一篇中我们会给大家讲到CMS底层的实现原理。喜欢文章的