图文 035、第5周答疑：本周问题答疑，上周作业点评

2027 人次阅读 2019-08-04 09:55:58

详情 评论

第5周答疑：

本周问题答疑，上周作业点评

问题：

老师，我刚刚回想了一下，突然懂了。本案例是秒杀系统和高峰时期的案例情况分析。如果是平时的情况下。每秒也就是不足10个订单的请求。

那么每秒在eden区的对象10*1*20*10=2000kb大小，按照新生代区2g大小，可以保证说minor gc频率大概在20分钟左右。

回答：

是的，一般其实新生代gc没那么频繁的

问题：

我回去分析了一下系统gc情况，整个堆3600m，新生代仅仅给680m，然后平均每隔40分钟一次minor gc，即便深夜也是这样，我估计是一些定期任务在执行。

然后老年代占内存600多m，没有fullgc，好像没什么要调优的了。唯一有点诟病就是新生代给的太少的，才680，是通过xmn指定的。

我觉得起码要给个默认的1200m吧，这一块是否需要调大新生代呢？

我的观点是，调大了也就是降低一点minorgc的频率，好像没有什么太大的收益

回答：

对的，你们系统新生代那么小，还要40分钟一次minor gc，原因只有一个，那就是你们系统的访问量很低，负载很低，产生对象的速度很低

这种情况下，也可以给新生代多分配一些内存，其实说白了，一些负载很低的系统，可能jvm本身就没什么好调优的

问题：

老师，负责的系统对内存使用压力有多大，意思是每秒有多少个请求，这些请求会生成多大的对象对吗，多少个请求怎么看啊？

回答：

你完全可以根据自己写的代码来预估，基本都能估算出来 ，每秒多少请求，每个请求会连带产生多少个对象，每个对象大致多大。后面我们也会讲怎么通过工具来看

问题：

感觉还是需要系统复杂度和压力上去才能更多的暴露问题

回答：

对的，没复杂度和高负载的系统，一般JVM这块出问题的比较少

问题：

老师，是否可以监控到因为survivor区域放不下或者因为动态年龄导致进去老年代的对象大小或者次数甚至是哪些对象？ 如果能够监控到那岂不是更加准确调优参数？

回答：

刚开始开发完毕肯定得先按照估算的来进行jvm参数设置，后续在压测、生产 环境可以用工具来精准的检测，这个工具后续会讲

问题：

老师，请问一下，minor gc 之前如果老年代剩余空间小于新生代所有对象大小或者小于之前历次 minor gc 后进入老年代对象平均大小，从而会触发老年代 full gc，那么这个老年代的 full gc 使用的垃圾回收器是 CMS 还是 Serial Old？

回答：

此时垃圾回收是CMS

问题：

文章里说，老年代的回收触发有两种情况，我不理解第二种（在minorGC之后）

按文章说的，根本就不会有这种情况啊，因为在minorgc之前都会检查老年代的大小是否大于新生代的大小。

如果是小于新生代的，都先fullGC了，哪里了还有在minorgc以后的?

还有疑问就是：如果我把老年代的大小设置为小于新生代的，例如新生代设置了100m，而老年代设置了1m，岂不是基本每次minorgc都fullgc?

回答：

1、不是，你没看清楚，如果老年代可用空间大于历次minor gc后升入老年代的对象平均大小，也可以进行minor gc，但是此时可能minor gc后的存活对象特别多都要进入老年代，就会触发full gc

2、不会，你仔细看看，现在最新版本里，只要老年代可用空间大于历次minor gc后升入老年代的对象平均大小，就可以直接进行minor gc，不需要每次都full gc

问题：

老师，请问用了G1回收器，是不是我们前面的知识都用不上了?

还有G1这个停顿时间我们应该怎么预估呢？比如我们设置一小时停顿0.5秒，当它明显做不到这个停顿时间的时候，它会怎么样?

回答：

不是的，G1里很多原理都是跟之前讲解的一脉相承，都有用，具体看后续的文章

问题：

我司的广告系统，使用的就是G1垃圾回收器……因为是一个30G的大堆，传统回收器可能会造成很大的停顿，所以使用了G1

回答：

对的，G1非常适合超大内存的机器，因为内存太大，如果不用G1，会导致新生代每次GC回收垃圾太多，停顿时间太长，用了G1可以指定每次GC停顿时间，他会每次回收一部分Region

问题：

我现在有点迷糊了：jdk1.8 Minor GC之前是否还比较老年代的可用空间大于新生代存活对象的总大小？如果还比较的话，调大年轻大的大小，调小老年代的大小，那岂不是有问题?

回答：

两个条件，“老年代可用空间” > “新生代所有对象大小”，“老年代可用空间” > “新生代历次minor gc后升入老年代的平均对象大小”，满足任意一个即可

问题：

新生代和老年代各自的内存区域是不停变动的，新生代变为老年代可以理解，老年代会变为新生代么？

我猜想应该不会，老年代的Region空间全部回收完了，可能从老年代 变成 初始状态（非新非老），然后变成新生代，这样循环轮回。而不会从老年代直接变成新生代吧

回答：

会变，G1里一切都可以变，Region都是动态灵活的，一个Region回收掉垃圾空了之后，未来可以分配给新生代，也可以分配给老年代

问题：

虽然我们没有用过 G1，但是从 GC 效果上看，G1 最明显的特点就是可以预测 STW 的时间。

而 G1 为了达到这个效果，抛弃传统分代内存，把它们都变成了各个小内存块 region ，针对这些内存块 region 预测垃圾回收的性价比，然后选某些性价比最高最高的内存块 region 进行 GC，以在预先设定的 GC 时间内完成 GC。

猜测可以用到对 STW 特别敏感的业务上。这可能是实时通信之类的，也就是一些追求低延迟的响应的业务。

回答：

对的，还有就是那种大内存机器，比如16G，32G的机器部署的系统，不用G1一次回收时间太长了，内存满了对象太多了，用了G1可以控制，每次就回收部分Region即可

问题：

按照老师的介绍， G1 是不是也会给年轻代和老年代分配一定的内存比例 虽然region不一直属于年轻代活老年代， 但是也会有类似 eden区放不下的情况 这个时候触发 region gc 然后根据“回收价值”去做回收 这样猜测不知道正不正确？

回答：

对的，明天的文章里会分析这些细节

问题：

按region回收会不会形成新的内存碎片呀？

回答：

不会的，后续会分析，Region回收的时候走复制算法，存活对象挪动到其他Region，然后对一个Region直接回收掉全部垃圾

问题：

老师，有个疑问，假设新生代占55%的堆内存，老年代占30%的堆内存，大对象占15%的堆内存，这样的情况会进行垃圾回收吗？

回答：

如果新生代已经没法给eden分配更多空间了，那肯定会垃圾回收

问题：

老师，有点想不明白，G1分那么多的Region,有点像hdfs里面的小文件，小文件太多会影响性能，但是为什么G1的性能会比之前那些更好呢？

回答：

很简单，划分为很多的Region了，回收的时候比如你设定只能停顿系统20ms，那他就挑选比如几十个Region回收，可以控制垃圾回收的停顿时间。

如果按照以前那么简单粗暴的划分，必须回收整个新生代，岂不是每次gc都要停顿更多时间？

思考题：从新生代的垃圾回收来看，大家觉得G1垃圾回收器在新生代垃圾回收过程中，相比之前的ParNew而言，最大的进步在哪里？

学员回答：最大进步就是STW可控，但是，虽然各个Region所属区域是动态变化的，但不是随意变化的，还是会为Eden、Survivor、老年代保留各自需要的空间。例如不会让Eden空间的分配超过系统设定的值

回答：

是的，这点非常关键，其实G1很适合大内存机器，因为比如你给JVM分配32G内存，要是用ParNew+CMS，每次gc都是内存快满了，此时一下子要回收对象太多了，就会导致gc停顿时间很长，所以针对那种大内存机器，用G1是很合适的

问题：

咦，我还是沙发！ 终于追上了进度，每天早上到公司开了电脑，启动了工作要用的所有程序，就开始看书。 哈哈，早上记忆里好，看了顺便做点简单笔记，然后晚上回去复习下笔记，基本就记住了。 就是实操的比较少，才入新公司，还对项目的各种不熟悉。自己的阿里云又没有项目在跑，感觉时机正不好。

回答：

非常好，后续会有很多的实操环节的

问题：

文末给出的案例非常好，我在上上周已动手调试了我自己的项目 ，就是一个单体的spring boot 在8g内存台式机上跑， 需要加载的类特别多 ，根本没有什么百万流量。

jvm一启动 直接每秒10m的内存增长速度，光是启动要花十分钟。启动期间就进行了两次full gc，半小时走了十几次ygc

当时我就调了新老比例为2:1 ，共分配4g， 之后 fgc一直为0， ygc半小时只有两次， 启动降为1分钟以内

回答：

是的，这就是典型的新生代内存不足，导致你启动系统创建一堆对象，然后新生代不够，频繁young gc进入很多对象到老年代，然后老年代又不足了，又要对老年代full gc，最后就是十多次young gc+几次full gc，因为gc太多还导致启动速度很慢。

你优化了比例之后，新生代内存充足，很多对象都直接进入新生代，不用进入老年代，最多就是少数young gc回收一部分对象，也不会有full gc。gc次数减少了，那启动速度立马就快了

问题：

老师，网上大部分资料设置新生代和老年代都有一个比率(1:2)，这个比率是正确的吗？并且和你说的不一样，新生代一般小于老年代。都不知道以那个为准了。

回答：

不对，那都是错误引导，完全没从根上来说明怎么优化jvm参数，可以跟着我们的文章来学，是最科学的设置方法

问题：

我觉得G1垃圾回收也应该合理分配年轻代的占比，保证 supervisor区放得下 或者 动态年龄不会那么快进入老年代，让老年代很快占到百分之45。

如果老年代不那么快占到百分之45自然就可以减少 混合回收

回答：

分析的非常的好

问题：

1. G1回收器的4个阶段是只针对混合回收的吗？那么新生代的回收过程又是如何的？ 假如老年代总量没有达到45%,是不是不会进行混合回收，只会进行新生代的回收，顺带回收大对象Region

2. G1混合回收、第四个阶段会进行多次混合回收，那么这个多次混合回收的间隔是如何定义的？

3. 空闲的Region数量达到堆内存的就会停止回收，也就是说正常是8次混合回收，但是可能到了4次，发现空闲Region达到5%了，就不进行后续的混合回收

4. 回收失败时Full GC，应该是采用Serial Old回收器吧。

回答：

1、新生代垃圾回收策略也说了，基本和之前一致

2、间隔由g1自己控制

3、对的

4、是的，直接就是serial old

学员思考题回答：使用G1垃圾回收的时候，值得优化的地方：

对于新生代，和之前的理论差不多，主要目标是避免短期存活的对象进入老年代。

1. 预估系统每次GC后存活对象，确保Survivor能放得下。

2. 避免高峰期间，新生代对象满足动态年龄判断条件，导致短期存活对象进入老年代。

3. 大对象有大对象Region，不占用老年代空间，基本不用考虑。

对于老年代；

1. 对于可预测停顿时间，需要合理设置，并不是越小越好，如果过小，有可能多次回收效果不大，最终导致回收失败FullGC，停顿系统线程。

2. G1HeapWastePercent 这个参数，我觉得应该可以适当提高，避免万一真的遇到了高峰期，短期存活对象进入老年代，但是回收的时候，进行了几次混合回收的时候， 刚好达到了5%,但是在老年代Region中可能还是存在某些短期存活对象没有被回收。过早结束混合回收。

3. -XX:G1MixedGCLiveThresholdPercent这个参数，暂时不用管，降低这个参数可能会让Region的回收效率更高，但是也可能导致短期存活对象驻留内存时间过长，进入老年代的风险。

回答：

分析的很好

问题：

这篇有个疑问，JVM能保证无论如何都有一个Survivor区是空着的吗？

会不会存在这一个情况，第一次在Survivor1区有存活对象，然后第二次GC的时候，又放到了Survivor2区(同时s1也有对象)，第三次GC的时候，s1和s2都有存活对象，这样就不能保证无论如何都有一个Survivor区存在的情况。

回答：

不会的，一次gc，会把eden和s1里的存活对象都转移到s2，然后清空eden和s1

学员思考题回答：

1. 如果使用G1垃圾回收的时候，应该值得优化的是什么地方？

感觉jvm优化的主旋律就是，尽量让短命对象在新生代回收掉，长期存活对象早进入老年代，G1的优化思路亦是如此。

首先是根据具体业务系统，合理分配老年代和新生代的大小、新生代Eden和Survivor区大小 其次是根据具体业务系统，合理设置G1的MaxGCPauseMills大小。

太小容易造成回收频繁，影响系统的吞吐量。太大会增大系统的停顿时间，影响用户体验

2. 什么时候可能会导致G1频繁的触发Mixed混合垃圾回收？

1、InitiatingHeapOccupancyPercent设置的值太小

2、新生代和老年代空间设置不合理，导致进入老年代对象太多，频繁达到MixedGC的条件 上面两个原因导致频繁的MixedGC

3. 如何尽量减少Mixed GC的频率？

首先看触发MixedGC的条件是什么，触发MixedGC条件是到老年代达InitiatingHeapOccupancyPercent设置的值，这就会回到如何让老年代尽量不达到这个值的问题。

1、让垃圾对象尽量在新生代就被回收掉，尽量让短命对象不进老年代。这就要求根据具体应用系统来合理设置新生代Eden大小和Survivor的大小。

2、将老年代设置为较小的值或者提高InitiatingHeapOccupancyPercent的值，这样就可以使达到触发MixedGC概率降低。但这样可能会存在一些问题：

（1）设置老年代为较小的值存在的问题：如果有较多的需要长期存活的对象的情况下，容易FullFC或直接OOM了.

（2）提高InitiatingHeapOccupancyPercent存在的问题：虽然降低了MixedGC的频率，但导致老年代存在过多的对象，增加了每次老年代回收时‘并发标记’阶段的计算负担和‘MixedGC’阶段计算和预估的负担。不适合CPU负载较高的计算型业务的系统

回答：非常好的分析解答

问题：

坚持，加油，老师讲得很好!我小白同时跟这门和石杉老师的课，有点困难，时间消耗比较多，坚持加油!

回答：

一定要坚持，学习技术就是这样的，长期跟着坚持下来，收获就会很大

问题：

一如既往的分析流程，总结一下：分析系统的背景和核心业务流程，预估高峰时长及当时的活跃用户，预估产生多少请求及每个请求处理时长，预估需要部署多少台机器及每台机器每秒抗多少请求，预估一个对象的内存大小及每秒产生对象总大小，然后对JVM参数进行调优。老师看是否有遗漏的?

回答：

相当好，这套理论掌握好了，以后才能从根上玩儿转jvm优化

问题：

我自己先分析了一下，按照老师的思路接着往下走：

比如每台机器堆内存5G，新生代最多占堆内存的60%，即3000MB左右，1秒3MB对象，60秒180MB对象

Eden区域和survivor是默认的8:1:1比例来计算，Eden大概有2400M，要占满需要大概14分钟就得去回收一次

每个survivor区有300MB内存，每次回收大概有180MB对象会卡在内存中，因为1分钟请求一次，一分钟用户才点击一次按钮，所以一次请求大概得一分钟才能处理完（我是这么想的你比如讲个故事大概讲50多秒，快60秒的时候点一下按钮）

所以大概有200MB左右的垃圾不会被回收进入了survivor区，此时超过了survivor的50%了，会进入老年代

也就是每隔14分钟就会有200MB的垃圾进入老年代，快一个小时的时候有800MB会进入老年代，老年代达到45%会进行混合垃圾回收，那么大概1个小时零10分时候就进行一次混合回收，我觉的如果是这样的话应该还可以，系统性能不会太差

因为我这里还没有考虑G1在规定的时间内尽可能的去回收更多的垃圾，每次回收还要分8次，在这8次中的空隙还让用户线程去工作，性能应该不会差。看着今天的文章，心痒痒了，自己大概练了一下，不知道对不对，抛砖引玉吧。

回答：分析的非常好

问题：

刚看完，简单记录一下笔记。调优思路还是减少耗时的垃圾回收，也就是避免对象进入老年代。

传统的垃圾收集器是通过调整s区和e区的大小来控制，而G1则比较先进，直接指定一个期望的停顿时间，选择停顿时间的标准是既不能频繁触发minor gc，也不能一次回收过多的对象，所以还得通过工具来调试出一个最适合自己系统的停顿时间。

通过工具检测，要得出一个停顿多久可以回收多少的内存大小的指标。根据这个指标和业务系统生成垃圾的速率设置合理的停顿时间。之后等作业里面再整理一个详细的。

回答：是的，分析的很好，都吸收了

问题：

老师的意思 G1就是调最大停顿时间 过大新生代 存活对象太多 可能新生代gc过后 存活对象太多 supervisor放不下 进入老年代 或者 因为动态年龄进入老年代 其实mix gc也是新生代的最大停顿时间设置的不合理

回答：

是的，其实核心的调优没那么复杂，关键是明白背后的原理，调节几个关键参数即可，不是说优化jvm就一定要优化一大堆的参数

学员思考题回答：

到底为止，大家已经基本学明白了G1的运行原理以及基本的优化思路，那么我想问大家两个问题：
G1这种垃圾回收器到底在什么场景下适用呢?
有了G1以后，是不是还有一些场景采用“ParNew+CMS”垃圾回收器也可以呢?

1、G1压缩内存空间会比较有优势，适合会产生大量碎片的应用；

2、G1能够可预期的GC停顿时间，对高并发应用更有优势

3、其他垃圾收集器对大内存回收耗时较长，G1对内存分成多块区域，能够根据预期停顿时间选择性的对垃圾多的区域进行回收，适用多核、jvm内存占用大的应用

4、parNew+cms回收器比较适用内存小，对象能够在新生代中存活周期短的应用

回答：分析很好

学员思考题回答：

到底为止，大家已经基本学明白了G1的运行原理以及基本的优化思路，那么我想问大家两个问题：
G1这种垃圾回收器到底在什么场景下适用呢?
有了G1以后，是不是还有一些场景采用“ParNew+CMS”垃圾回收器也可以呢?

g1适合 大堆的场景 或者有业务不能有太高的延时 如果业务上不需要特别大的堆 或者 业务属于不需要及时反馈用户的比如贷款业务 申请额度之后就后台处理了 有额度以后 在通知你这个时候 par new 加 cms可以用的

回答：分析的非常到位

问题：

请问老师我的理解对么：使用系统的用户其实并不关心什么gc频率，但他们关心的是我使用的系统卡不卡，处理速度快不快。系统卡不卡是受-XX:MaxGCPauseMills影响的。系统处理速度快不快是受gc的频率影响的。

“gc的频率高”翻译一下应该是“系统的吞吐量低”，gc频率高说明cpu用来处理垃圾回收的时间比例变多了，自然用来处理业务的cpu时间变少了。比如原先每秒可以取出1w条数据，现在只能取出2k条。

回答：理解正确