图文 042、第6周答疑：本周问题答疑汇总！

1671 人次阅读 2019-08-11 07:00:00

详情 评论

第6周答疑：本周问题答疑汇总！

学员总结：

g1和pn+cms调优原则都是尽可能ygc，不做老年代gc。 g1相对而言更加智能，也意味着jvm会用更多的资源去判断每个region的使用情况。

而pn+cms也更加纯粹和直接，虽然g1在gc时不会产生碎片，但是由于每个region存在存活率85%不清理的机制，会导致内存没有充分释放问题。

因此，对于cpu性能高的，内存容量大的，对应用响应度高的系统推荐使用g1。 而内存小，cpu性能比较低下的系统也可以使用pn+cms会更合适。

回答：分析的很好

问题：

复习的总结： 脑子里一定要有一个会动的图：

1、启动一个线程执行业务代码（执行main方法就是开启一个main线程）；

2、线程对应的程序计数器PC来记录程序执行到哪行字节码指令（线程与PC是1:1关系）；

3、调用方法时会创建一个“栈帧”，放入线程对应的栈中（线程与栈、方法和栈帧都是1:1关系）

4、代码运行中创建的对象放在java堆内存（堆与JVM进程是1:1关系，堆是所有进程共享）

回答：很好，可以把后面学习的各种gc原理都总结一下

问题：

老师，这里为什么说G1就适合大堆的情况呢？说对实时性要求高一点的可以理解，因为它有Maxpause的时间限制，但这个适合大堆的情况是怎么来理解呢

回答：假设你有32G内存，如果用ParNew+CMS，必须等待你的内存填满了才会触发GC，此时一GC就会回收几十G的垃圾，那么速度会很慢，可能导致你的系统停顿时间多达几十秒都有可能。

但是用了G1之后，他会更加频繁的回收Region，每次就回收一部分Region，保证停机时间不会太长。所以G1其实更加适合大内存的机器

问题：

动态对象年龄判断我有点疑惑，假设年龄为3的对象大小超过了survivor 区域的一半，年龄都为3，说明它们都是经过了3次gc存活下来的，都是从年龄为2的时候经过gc存活下来的，那在年龄为2的时候这些对象大小就应该超过survivor的一半了

同理年龄为1的时候也一样，也就是说如果有同龄的对象大小超过了survivor的一半，就只能是年龄为1的对象，而根据规则，年龄大于等于1的就得转移到老年代去了

那根据以上的推论，根本就不会出现年龄大于1的同龄对象总大小超过survivor一半的，如果有，早在年龄为1的时候这些对象就已经转移到老年代了。

然后在一篇博客上看到说这是一个误区，是年龄从小到大进行累加，当加入某个年龄段后大小超过一半了，就从这个年龄段往上的年龄转移到老年代，而不是某个年龄的大小。看上去好像也挺有道理的，所以就有点疑惑。

回答：其实是年龄1+年龄2+年龄3的对象占据了超过50%的Survivor，就会让年龄3以上的对象进入老年代，动态年龄判定规则应该是这样的

问题：

请问老师，动态年龄判断算法是这样么：Survivor区的对象年龄从小到大进行累加，当累加到 X 年龄时的总和大于50%（可以使用-XX:TargetSurvivorRatio=? 来设置保留多少空闲空间，默认值是50），那么比X大的都会晋升入老年代

回答：对的，就是你说的这样

学员思考题回答：

老师，请您耐心帮我看下，如果有错误的话帮我指点下，谢谢啦。

系统采用是g1回收器。

1.如果新生代未达60%，老年代未达45%，系统照常运行，不会触发回收

2.如果新生代达60%，此后如果如有新对象生成，跑到新生代，会触发ygc.

（1）开启了空间担保机制，ygc先判断是否需要fgc,如果每次回收后对象少于老年代空闲大小，则不用fgc,否则要

（2）不用触发，但ygc后的对象大于老年代空闲大小，无法直接进入老年代，触发fgc.

（3）触发混合回收，先通过gcroot初始标记哪些不是垃圾对象(此过程会stw,不过很快)，然后并发标记(用户线程和标记线程并行)，接着最终标记(会stw，标记并发标记过程中可能新产生的垃圾对象)，最后混合回收(此过程采用复制算法，不会产生垃圾碎片，所以不用在回收完去整理内存碎片

g1会按照我们给定的时间去stw并回收，争取回收性价比的对象，如果回收次数少于8次，则再次混合回收。不过，在回收中空闲region大小达到堆5%，会提前结束。)如果回收失败，则转换采用serialold回收器。 3.当老年代代达45%会触发上面那个混合过程。

回答：分析的非常好

问题：

老师，请教下，是否有方法可以根据设定的xss的大小推算可以支撑多少线程的方法?

是不是可以理解为整个jvm的大小减掉堆和方法区的余下额度/xss的大小呢？谢谢

回答：对的，就是这样子，但是一般单个JVM内部也就最多几百个线程，其实不会太多的

学员评价：

感谢老师的这篇，这样螺旋上升可以说是非常的值得了。

回答：是的，我们的思路就是每周布置作业，对核心原理会贯穿全文反复的强化，最终就是让你彻底建立起来系统运行时候的jvm动态运行模型，理解如何对jvm进行优化，解决问题

学员评价：

老师深知人脑的遗望曲线啊，还专门花一周带着我们复习，太赞了，有时候学得太快反而学完就忘，只有充分消化成自己的东西才能达到学习的最佳效果。期待老师写完这个专栏继续出更多的好专栏。

回答：是的，我们很了解大部分人学习的问题，很多人也许会做作业，很多人也许不做作业，所以必须把核心知识螺旋形，反复强化，深入每个人的脑子里去，让脑子里有一个jvm运行时的动态图

学员总结：

文章总体意思，就是新生代gc一般没什么影响 ，但是大内存的堆就会导致 新生代回收很慢 。

如果在高并发的情况下，不仅慢而且还没频繁 full gc影响很大 涉及过程很复杂 ，时间也是新生代的10倍以上 。但是 full gc频繁不频繁又和新生代 gc有很大的关系 比如

1.新生代年龄躲过15次以后

2.大对象直接进入老年代 G1例外 有专门存储大对象的region

3.动态年龄规则 4.supervisor区放不下

其中 3和 4是 最值得关注的 比如supervisor区内存很小 会引发3 4 很频繁 导致很频繁的full gc

回答：分析总结的非常到位

问题：

老师，学了一个多月还积累了几个问题。

1、minor fc和full gc一样都是追踪追踪gc root，为什么full gc的root追踪就更慢呢？是因为minor fc的root更少，还是链条更短？

2、从root追踪是垃圾回收线程从扫描栈中的局部变量开始吗？

回答：

1、老年代gc，从GC Roots开始追踪，但是老年代的存活对象更多，所以追踪速度更慢；新生代存活对象极少，所以追踪速度极快

2、GC Roots就是两种，方法里的局部变量，类的静态变量，从这两个地方开始追踪扫描即可

学员学习总结：

前几天碰到了full gc非常频繁的状态。由于同事采用了默认的jvm参数，导致前期新生代不断扩容，对象直接进入老年代。

中期，s区大小不够，对象还是往老年代走。后期，老年代不断在full gc，导致cpu使用率99%，阿里云都报警了。

最后我固定了新生代2g，老年代2g，eden和s比例为2，并且使用了cms，最后屏蔽了system.gc()的full gc回收。最终可以做到只做minor gc的效果。感谢老师这一个月的教导。

回答：非常的好，足见你真的吃透了这一个月的内容了，而且对线上系统的jvm问题，可以分析出来是怎么回事，合理优化内存分配，就能提升jvm性能，继续加油

学员思考题回答：

系统创建的对象被分配到java堆内存中，要想计算创建的对象内所占的内存对象，就需要计算对象的每个部分所占的内存大小；

java对象包括对象头、实例数据以及对象填充： 对象头包括对象的基本信息以及class的指针类等相关信息占用64bit（64位机器不缩）；

数据实例包括八种数据基本类型以及引用类型，将所用的全部算起来相加； 最后再加上对象头，计算8的最低倍数进行是否填充，最终则为所占用的内存空间bit

回答：对的

学员总结：

新系统上线要估算核心业务，每次请求产品的内存垃圾

例如100k，访问量QPS为100，则每秒产生垃圾10M，每次请求响应限制在时间为200ms以内,每次回收会有2M的对象卡在内存中,这40M内存触发Minor GC的时候还有引用，是存活对象，Minor GC后进入Survivor区。

1个4和8G的服务器，可以分配永久代0.5G(512M)+新生代4G+(4G+0.5G+0.5G)+永久代1G， 那相当于平均没400秒（大约7分钟）Eden区就满了，就触发Minor GC。

如果发现有太多对象没进入Survivor区，而是直接进入老年代，频繁Full GC的情况，则可以加大Survivor区。

如果发现有对象写对象存活时间较长，进行15次GC后进入老年代，然后被Full GC清理。则可以加大MaxTenuringThreshold的参数值。

大致思路就是尽量减少对象进入老年代，让年轻对象在MInor GC就被清理。 如果访问量暴增，则可以用多态机器进行负载均衡。

回答：分析的非常好

问题：

老师：一直有一个疑问，就是对象在minor gc后，survivor无法容纳就会进入老年代。疑问：gc剩余的对象年龄各不相同，应该是部分进入老年代？还是全部进入老年代

回答：要是Survivor无法容纳， 那就全部进入老年代

问题：

假如我们设置GC停顿时间设置为20ms，新生代默认最大占比60%，当G1监控到Region区域的对象的回收预期时间满足20ms，是否会马上进行一次minor GC？此时新生代的占比仅为20%。

老师请问下，上述表述是否正确，或者说另一个触发G1 minor gc的条件是新生代监控到的可回收象的回收预期时间满足MaxGCPauseMills就立马触发，进行垃圾回收，释放空间?

回答：不会的，其实他很可能让你用到50%的时候，然后回收掉里面一小部分Region，保证gc停顿时间在20ms

学员动手之后的总结：

在自己的阿里云上跑了一个开源的后台管理系统。 最开始启动后，我随便刷了几个网页，在服务器上用命令一查看GC情况，YGC 100多次，FullGC 6次。然后看JVM内存，才100M。

然后我就调大了 JVM 堆内存，新生代内存，还没有调 Eden 区的比例。然后我又刷了20多次页面，一直追踪 JVM 内存占用情况。此时才发生了6次YGC，3次FullGC，而且在刷新网页的时候，没有发生 FullGC 的情况。所以3次 FullGC 应该是启动的时候发生的。

一次简单的实际操作，感觉对所谓的 JVM 优化，理解又上了一个小层次。

正好呼应了老师今天的课程，为什么要 JVM 优化，就是为了避免系统频繁的卡顿，因为如果内存大小、参数设置不合理，就会让对象频繁进入老年代，老年代对象迅速增多，频繁触发 FullGC。

回答：分析的非常棒

问题：

老师，感觉jvm的优化就在于尽量降低老年代gc，而最根本的就是对于新生代的内存分配，同时如果是大内存的场景对于垃圾回收器就可以考虑G1.平衡垃圾回收的资源数和时间关系。

这么理解合适吗？谢谢

回答：理解的很到位

学员总结：

今天对一些概念理清总结:minor gc/young gc即年轻代垃圾回收，old gc单纯指老年代垃圾回收，full gc指年轻代，老年代，永久代等整个JVM的内存回收，mixed gc 指年轻代和老年代回收。建议用以上名词，少用major gc。

回答：对的

问题：

可以不可以这样理解采用G1回收刚开始新生代是不会回收的即便是最初的eden满了也不会gc此时因为还有好多region可以分配所以继续扩展region直到堆内存的60%或者你设置的置才不会扩展region但是回收 region还是遵循规定时间内回收一部分region

回答：理解正确

学员总结：

哟，又是沙发。 早上到公司，开机，开发工具全部启动。然后开网页，看老师的课程，如果有些比较不太理解的知识点，或者比较重要的知识点，就记录在笔记里。 谢谢老师的课程，对我来说，起到了查漏补缺，并且深入理解原理的作用。 跑了个开源项目在阿里云，也可以实际操作，跟着老师课程走。 肯定物有所值。

回答：多谢支持，继续加油

学员总结：

温故了一下：jvm内存优化就是 了解自己的系统运行的内存模型，估算合理的单位时间内产生的对象大小，然后分配合理的新生代和老年代内存空间，尽可能让普通的对象在新生代中折腾，不进入老年代.

回答：对的，完全get到我们的核心要点了

学员评价：

一路跟学下来感觉受益匪浅 老师内功深厚 毫无保留的授课 真的非常让我们感激 讲解思路清晰 希望以后老师能多出些课程 这样高含金量的专题太少了

回答：多谢支持，我和我的朋友未来会出更多类似这样的专栏的

学员思考题回答：

什么时候尝试触发Minor GC：

1，检查老年代的可用内存大于新生代的所有存活对象的大小直接触发Minor GC；

2，前1中检查结果相反，如果设置了handlerPromotionFailure然后检查老年代的可用内存大于历次转移过来的对象平均大小，则会尝试MinorGC，如果没设置或者检查相反也会触发，不过是先FullGC。

什么时候回提前触发Full GC：在设置了handlerPromotionFailure下，老年代的可用内存小于历次转移过来的对象平均大小或者没设置了handlerPromotionFailure，可用内存小于新生代转移对象总大小则提前触发Full GC。

Full GC算法：标记整理，需要标记存活对象并且移动排序到连续内存，耗时较长，这应该是就是为什么新生代不适用一块内存使用这个算法的原因吧。

Minor GC之后对应集中情况：

1，Minor GC 之后存活的对象可以放入Survivor中。

2，Minor GC 之后存活对象较多但可以放入老年代。

3，老年代依然存不下转移对象产生OOM。

哪些情况Minor GC 后会进入老年代：

1，每次Minor GC 存活对象年龄超过默认15岁（可设置）。

2，通过判断动态年龄达到的进入老年代。

3，大对象直接进入老年代

回答：分析的很好

学员总结：

看了大家的回复，都提到了spring 的bean。除此之外，我觉得还有以下几类长生命周期对象： 线程池的核心线程，coreSize那些，以及他们引用的对象，包括threadlocal引用的对象（除非手动清理） tomcat的各类组件，包括connector和container部分，比如filter，servlet，listener；

同时，classloader， Class对象这些也是长生命周期； 各类池化技术，比如线程池，连接池等等。 暂时就知道这些吧

回答：是的，很好

学员思考：

打卡。大内存机器用g1，一般机器用parnew+cms就可以了。思考题容我去看看线上机器内存，我们一个机器都部署了好几个系统。还没有用容器技术。有的时候卡的不行，估计就是因为stop the world 导致的。

回答：是的，一般系统卡顿就是gc导致的

学员思考：

打卡。我们线上系统现在日活才1000左右，所以部署在4核8G机器上基本没压力，更多压力来自于定时任务，这个是每分钟就要读数据然后计算一下

但是相对于来说每秒吃的内存也很少，触发Young GC后，基本都回收完了，就是一些托管给spring的类进入老年代。所以这样的系统主要还是保证业务正常就行了，性能怎么样不太关心。

回答：很好，起码你对自己系统有个底了