图文 058、案例实战：垂直电商APP后台系统，如何对Full GC进行深度优化?

详情 评论

案例实战：
垂直电商APP后台系统，如何对Full GC进行深度优化?

狸猫技术窝专栏上新，基于**真实订单系统**的消息中间件（mq）实战，重磅推荐：

未来3个月，我的好朋友原子弹大侠将带你一起，全程实战，360度死磕MQ

（点击下方蓝字进行试听）

从 0 开始带你成为消息中间件实战高手

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我都会逐一答疑

（**ps**：评论区还精选了一些小伙伴对**专栏每日思考题的作答**，有的答案真的非常好！大家可以通过看别人的思路，启发一下自己，从而加深理解）

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**。

（群里有不少**一二线互联网大厂的助教**，大家可以一起讨论交流各种技术）

具体**加群方式**请参见文末。

（**注**：以前通过其他专栏加过群的同学就不要重复加了）

1、垂直电商业务背景

今天给大家讲的这个案例，是我协助另外一个朋友的公司进行全公司级别的JVM性能优化的案例，很多核心的思想其实也跟之前是相同的，只不过在优化的过程中会带出来一些比较高级的参数的调优。

先给大家说一下这个公司的大致业务背景，这是一个垂直电商公司，做的是一个垂直电商APP

之前的文章里给大家提过，其实现在除了淘宝、京东、天猫、唯品会这些超大型的电商平台之外，国内还是有很多中小型的垂直类电商公司的。

他们做的主要是一些细分领域的电商业务，比如说有的APP专门做消费分期类的电商业务，在他们的APP里你主要是进行购物，然后可以分期付费。

还有的APP，他专门是做服装定制的，他可能是会在APP里让你选购商品，然后有人上门给你定制化的测量身体，然后给你做定制的衬衫或者西装之类的。

还有的APP，他是做时尚潮流服饰的，就是专门售卖针对年轻人的一些潮牌、设计师的品牌，等等。

所以这个朋友的公司就是类似上述的那种垂直电商APP，专门做某个细分领域的。

说实话，这个垂直电商APP大致注册用户量有就数百万的规模，不算多大，而且每日活跃用户数量也就几十万而已，每天APP的整体请求量也就小几千万的级别，也并不大。高峰期的QPS也就每秒数百请求罢了。

但即使如此的一个普通APP的后台系统，感觉上压力一点儿都不大，是不是真的就没有JVM的性能问题呢?

当然不是了，这个APP虽然不大，但是他同样有JVM相关的性能问题，而且也需要一些细致的优化才可以。

2、垂直电商APP的JVM性能问题

那么类似这样的一个垂直电商APP，他的JVM性能问题在哪里呢?

很简单，问题就出在类似这样的一个创业型互联网公司，虽然有少数几个技术比较好的架构师，但是架构师往往没那么大精力把控到特别细节的地方

所以大部分的一线普通工程师可能都对JVM这块没有那么的精通，起码说对我们的专栏里讲解的JVM相关原理和优化手段，都没了解。

所以这直接导致一个很大的问题，那就是大部分的一线工程师开发完一个系统之后，部署生产环境的时候往往就不会对JVM进行什么参数的设置，可能很多时候就是用一些默认的JVM参数。

默认的JVM参数绝对是系统负载逐渐增高的时候一个最大的问题

如果你不设置-Xmx、-Xms之类的堆内存大小的话，你启动一个系统，可能默认就给你几百MB的堆内存大小，新生代和老年代可能都是几百MB的样子。

所以当时这个垂直电商APP的很多后台系统，基本都是用的默认JVM参数部署启动的，前期是没什么问题，但是中后期开始，当有一定用户量，有一定负载了，此时就会出现一些问题了。

大家通过之前大量的学习，哪怕现在不画图，脑子里都有一个概念了，那就是新生代内存过小，会导致Survivor区域内存过小，同时Eden区域也很小。

Eden区域过小，自然会导致频繁的触发Young GC，Survivor区域过小，自然会导致经常在Young GC之后存活对象其实也没多少，但就是Survivor区域放不下。

此时必然会导致对象经常进入老年代中，因此也必然会导致老年代过一段时间就放满了，然后就会触发Full GC。

所以当时这个垂直电商APP的各个系统通过jstat分析JVM GC之后发现，基本上高峰期的时候，Full GC每小时都会发生好几次。

Full GC一般在正常情况下，都是以天为单位发生的，比如每天发生一次，或者是几天发生一次Full GC。

要是每小时都发生几次Full GC，那么就会导致系统每小时都卡顿好几次。这个时候必然是不行的。

在这个背景下，当时我的朋友是那家公司的架构师，找到我帮忙，在分析系统情况过后，定制了一套公司级别的JVM参数模板

在大部分工程师都对JVM优化不是很精通的情况下，通过推行一个JVM参数模板，让各个系统短时间内迅速就优化了JVM的性能。

3、公司级别的JVM参数模板

其实这个公司级别的或者团队级别的JVM参数模板，是一个很有用的东西，因为大家要知道，并不是每个人都会来学习我们的这个JVM专栏，也就意味着并不是每个人都是可以比较精通JVM的核心运行原理和性能优化的。

所以如果你是一个团队的leader，或者是一个中小型公司的架构师，那么必然是需要为团队或者公司定制一套基本的JVM参数模板的

然后尽量让大部分系统套用这个模板，基本保证JVM性能别太差，避免很多初中级工程师直接使用默认的JVM参数，可能一台8G内存的机器上，JVM堆内存就分配了几百MB。

下面我们可以来看看当时我和那位朋友一起定制出来的适合他们公司的JVM参数模板：

```
-Xms4096M -Xmx4096M -Xmn3072M -Xss1M  -XX:PermSize=256M -XX:MaxPermSize=256M -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFaction=92 -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=0
```

为什么如此定制JVM参数模板呢?

首先，8G的机器上给JVM堆内存分配4G就差不多了，毕竟可能还有其他进程会使用内存，一般别让JVM堆内存把机器内存给占满。

然后年轻代给到3G，之所以给到3G的内存空间，就是因为让年轻代尽量大一些，进而让每个Survivor区域都达到300MB左右。

根据当时对这个业务系统的分析，假设用默认的JVM参数，可能年轻代就几百MB的内存，Survivor区域就几十MB的内存

那么每次垃圾回收过后存活对象可能会有几十MB，这是因为在垃圾回收的一瞬间可能有部分请求没处理完毕，此时会有几十MB对象是存活的，所以很容易触发动态年龄判定规则，让部分对象进入老年代。

所以在分析过后，给年轻代更大内存空间，让Survivor空间更大，这样在Young GC的时候，这一瞬间可能有部分请求没处理完毕，有几十MB的存活对象，这个时候在几百MB的Survivor空间中可以轻松放下，绝对不会进老年代。

基本上在这个内存分配之下，对于这个垂直电商APP的大部分后台业务系统，都是可以轻松hold住的

不同的系统运行时的情况略有不同，但是基本上都是在每次Young GC过后存活几MB~几十MB的对象，所以此时在这个参数模板下，都可以抗住。

只要把内存分配完毕，那么对象进入老年代的速度是极慢极慢的，经过这个参数模板在朋友公司全部系统的重新部署和上线，各个团队通过jstat观察，基本上发现各个系统的Full GC都变成了几天才会发生一次。

此时在参数模板里还会加入Compaction相关的参数，保证每次Full GC之后都会执行一次压缩，解决内存碎片的问题。

关于内存碎片的影响和优化，上一篇文章刚刚分析过，那是为另外一个朋友的公司专门做优化的时候调整的参数。

4、如何优化每次Full GC的性能?

这里给大家再介绍一下当时帮那位朋友做优化的时候调整的另外两个参数，这个两个参数可以帮助优化FUll GC的性能，把每次Full GC的时间进一步降低一些。

一个参数是“-XX:+CMSParallelInitialMarkEnabled”，这个参数会在CMS垃圾回收器的“初始标记”阶段开启多线程并发执行。

大家应该还记得初始标记阶段，是会进行Stop the World的，会导致系统停顿，所以这个阶段开启多线程并发之后，可以尽可能优化这个阶段的性能，减少Stop the World的时间。

另外一个参数是“-XX:+CMSScavengeBeforeRemark”，这个参数会在CMS的重新标记阶段之前，先尽量执行一次Young GC。

这样做有什么作用呢?

其实大家都记得，CMS的重新标记也是会Stop the World的，所以所以如果在重新标记之前，先执行一次Young GC，就会回收掉一些年轻代里没有人引用的对象。

所以如果先提前回收掉一些对象，那么在CMS的重新标记阶段就可以少扫描一些对象，此时就可以提升CMS的重新标记阶段的性能，减少他的耗时。

所以当时在JVM参数模板中，同样加入了这两个参数：

-Xms4096M -Xmx4096M -Xmn3072M -Xss1M -XX:PermSize=256M -XX:MaxPermSize=256M -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFaction=92 -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=0 -XX:+CMSParallelInitialMarkEnabled -XX:+CMSScavengeBeforeRemark

5、垂直电商APP全公司采用JVM参数模板之后的效果

上述JVM参数模板，推广到了朋友公司的全部系统中，因为当时公司里几乎一线工程师对JVM优化的理解都达不到这个专栏讲解的水准，很多人都理解非常粗浅的，所以这套JVM模板参数是全部推行的。

经过各个团队采用jstat观察JVM GC情况，发现明显有了很大的好转，基本上各个系统的Young GC都在几分钟一次，或者十几分钟一次，每次耗时就几十毫秒而已。

Full GC基本都在几天一次，每次耗时在几百毫秒的样子。

基本上各个系统的JVM达到这个性能，就对线上系统没多大影响了。哪怕是不太懂JVM优化的普通工程师只要套用这个模板，对一些普通的业务系统，都能保证其JVM性能不会出现大的问题，比如频繁的Young GC和Full GC导致的系统频繁卡顿。

6、今日思考题

今天让大家思考一下：

> 你们公司有没有类似这里讲的JVM参数模板？
>
> 假如你是公司的架构师，结合你们公司的大部分业务系统的实际情况，会如何定制一套JVM参数模板？
>
> 是否你们公司有各种不同配置的机器？
>
> 针对不同配置的机器如何定制JVM参数模板？
>
> 你们公司有没有那种特例的系统，比如并发量特别高或者数据量非常大？
>
> 对特例系统该如何进行优化？

希望大家对以上问题积极思考，并将自己的答案发送到评论区交流。

有什么疑问，也欢迎大家在评论区给我留言！

End
