OOM模拟

1. JVM参数配置

```
java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xmx128m
GCLogAnalysis
```

2.GC情况分析

```
youngGC或者MinorGC次数10次
fullGC或者MajorGC次数19次
```

```
老年代经过gc后内存反而增加情况:
(1)新生存活对象经过gc后过渡到了老年代
(2)老年代对象只是被少量回收,大部分对象还是存活状态
(3)新生代内存空间不足,经过gc后内存空间仍然不足,对象直接进入老年代。
```

```
中间过程:
(1)首先会进行YoungGC,部分对象会过渡到老年代
(2)经过几次YoungGC后发现,old区也满了,会触发FullGC
(3)触发FullGC后发现,很多新生代对象进入old区,但是old区由于大部分对象处于存活状
态,并没有被回收,最终导致经过FullGC后old区不降,反而增加
(4)持续一段时候后,很快old区也满了,这时Young区对象无法进入old区,导致很快Young
区也会满
(5)最终由于内存不足导致OOM
原因分析:
(1)对象创建速率大于回收速率,需要增加堆内存大小,尽量在年轻代能回收掉的,不进入老年
代
疑问:
(1)中间是否会存在这样问题,当eden过度到survivor时候,发现survivor区内存空间已
经不足,这时候是否会出现survivor区部分对象提前进行老年代?
(2)为什么每次发生YoungGC后Old区都会有增加,一般不是经过预定次数的YoungGC后对象
才会进入Old区?
```

串行GC

1. 参数配置

```
java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+UseSerialGC  -Xmx512m -Xms512m  GCLogAnalysis
```

2．中间过程

```
(1)YoungGC经过几次YoungGC后old区很快变满了,这时候触发GC时候,由于单线程处理，gc只会回收old区,young区内存会处于没有被回收状态
(2)当再次发生gc时候,young部分对象会晋升old区,young区占用会减少但是不会出现被回收情况，没有线程处理
(3)经过几次gc后,old区减少后,再gc时会进行Young的回收
(4)经过几次gc后,老年代不下降反而上升触发fullgc
```

3．暂停时间

```
younggc:20-30ms
gullgc:30-40ms
```

4．吞吐量

```
(1)GC吞吐量:7K/sec
```

5．分析

```
(1)暂停时间比较长
(2)吞吐量比较低
(3)gc只能处理young区或者old区,不能兼顾,old区满时没办法对young区进行同时回收,导致young区回收不及时,没有足够空间分配给新对象,导致连续引发fullgc
(5)younggc次数：20，fullgc次数:2
(6)对象数:12208
```

6．Xmx1024m情况

```
(1)没有出现fullgc
(2)由于内存变大,大部分对象在新生代被回收掉,没有进入老年代，younggc暂停时间有适
```

当增加
(3)暂停时间40-50ms

(4)吞吐里没有太大变化
(5)younggc次数：12
(6)对象数:13950

7. 2048m情况

(1)没有出现fullgc,younggc次数也减少
(3)暂停时间70-80ms
(4)吞吐里没有太大变化
(5)younggc次数：6
(6)对象数:12936

8. 4096m情况

(1)没有出现fullgc,younggc次数也减少
(3)暂停时间120-130ms
(4)吞吐量没有太大变化
(5)younggc次数：3
(6)对象数:12610

并行GC

1. 512m情况

(1)younggc次数：35,fullgc次数:14
(2)younggc暂停时间10-20ms,fullgc暂停时间:20-30ms
(3)对象数:10725
(4)最后当老年代快满时,由于大部分对象处于存活状态回收不掉,会连续出现fullgc情况
(5)SerialGC触发频率低于并行GC

2. 1024m

(1)younggc次数：26,fullgc次数:2
(2)younggc暂停时间7-10ms个别长一些,fullgc暂停时间:30ms

(3)对象数:14789
(4)增大堆内存后大部分对象年轻代就会被回收,所以fullgc次数减少,减轻了old区gc压力

3. 2048m

(1)younggc次数: 10
(2)younggc暂停时间30-50ms个别短一些
(3)对象数:15432

4. 4096m

(1)younggc次数: 3
(2)younggc暂停时间70-100ms个别短一些
(3)对象数:13356,13099,12731

CMS

1. 512m

(1) YoungGC发生次数:20,CMSGC发生次数:8
(2) 生成对象数:12615,12190,12065,11039,12275
(3) YoungGC暂停时间:10-40ms,FullGC暂停时间:初始化标记+最终标记耗时非常短
(4) CMSGC在内存相对小的情况下,old区gc暂停时间会比较短,但是吞吐量会相对低一些,在执行cmsgc同时因为是并发执行的,中间可能会伴随younggc
(5)分配速率比较快,内存比较小情况下,可能会出现对象提前晋升或者直接进入old区,造成old区回收困难,从而连续引发fullgc,造成长时间暂停。
(6)cms在内存相对小情况下,gc暂停时间是比较小,对吞吐量影响体现在线程资源争抢和younggc暂停时间

2. 1024m

(1) YoungGC发生次数:15,CMSGC发生次数:3
(2) 生成对象数:13935,13660,15171,14750,14784

(3) YoungGC暂停时间:10-50ms,FullGC暂停时间:初始化标记+最终标记耗时非常短
(4) 适当增加内存后,cmsgc和younggc触发频率降低,younggc单次执行时间适当有所增长,cmsgc时间还是比较短

3. 2048m

```
(1)YoungGC发生次数:7,CMSGC发生次数:1
(2)生成对象数量:14405,13830,14397,14415,14572,13793
(3)YoungGC暂停时间:50-80ms
```

4. 4096m

```
(1)YoungGC发生次数:6
(2)生成对象数量:14530,13800,14207,13611
(3)YoungGC暂停时间:60-80ms
```

G1

1. 512m

```
(1)full-Young GC发生次数:37,mix gc次数:41
(2)生成对象数量:12138,11040,12328,11040,10834
(3)full-youngGC暂停时间:5-14ms,mixGC暂停时间:3-4ms
(5)并发标记过程中出现的停顿:初始化标记,再次标记,和清除时间很短。
(6)gc频率很高每次gc时间相比与之前gc有所下降
```

2. 1024m

```
(1)full-Young GC发生次数:14,mix gc次数:8
(2)生成对象数量:12956,13123,12436,13599,13120
(3)full-youngGC暂停时间:6-27ms,mixGC暂停时间:3-7ms
```

3. 2048m

```
(1)full-Young GC发生次数:12
(2)生成对象数量:11994,11171,11116
```

```
(3)full-youngGC暂停时间:8-25ms
```

4. 4096

```
(1)full-Young GC发生次数:12
(2)生成对象数量:12948,13432,11983
(3)full-youngGC暂停时间:20-47ms
```

总结

- 通过实验发现并行GC的吞吐量相对比较高
- 在适当增大内存时可以发现稳定性也是最好的GC是并行GC
- CMS GC的gc暂停时间相对比较短,回收率相对低,会和应用增抢线程
- G1GC内存比较小触发频率很高每次时间很短,增大内存会适当降低触发频率,整体吞吐量一般。
- 适当增加堆内存可以降低gc频率,让对象可以尽量在young区被回收，避免让对象过早进行old区,降低old区gc压力,从而减少Fullgc带来的长暂停的危害。
- 连续引发Fullgc情况:

```
(1)内存过小造成大量对象晋升,老年代经过fullgc后回收不掉,经过gc后内存不下降。
(2)对象生成速率明显大于回收速率也会造成大量对象提前晋升或者直接进入老年代
```

压力测试分析

1. CMS

```
内存配置:Xms512m Xmx512m
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:37885.87/sec
GC次数:406
平均耗时:1ms,最大耗时:3ms
```

```
内存配置:Xms1g Xmx1g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
```

```
吞吐量:37583.01/sec
GC次数:213

平均耗时:3ms,最大耗时:20ms
```

```
内存配置:Xms2g Xmx2g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:38221.95/sec
GC次数:105
平均耗时:5ms,最大耗时:41ms
```

```
内存配置:Xms4g Xmx4g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:36265.81/sec
GC次数:105
平均耗时:9ms,最大耗时:45ms
```

2. 并行GC

```
内存配置:Xms512m Xmx512m
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:43707.70/sec
GC次数:319
平均耗时:1ms,最大耗时:6ms
```

```
内存配置:Xms1g Xmx1g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:44450.22/sec
GC次数:196
平均耗时:1ms,最大耗时:2ms
```

```
内存配置:Xms2g Xmx2g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:42850.75/sec
```

```
吞吐量:43858.75/sec
GC次数:83

平均耗时:1ms,最大耗时:6ms
```

```
内存配置:Xms4g Xmx4g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:34480.78/sec
GC次数:40
平均耗时:1ms,最大耗时:15ms
```

3. G1

```
内存配置:Xms512m Xmx512m
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:40476.27/sec
GC次数:197
平均耗时:1ms,最大耗时:3ms
```

```
内存配置:Xms1g Xmx1g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:37094.99/sec
GC次数:93
平均耗时:1ms,最大耗时:3ms
```

```
内存配置:Xms2g Xmx2g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:39133.57/sec
GC次数:48
平均耗时:1ms,最大耗时:4ms
```

```
内存配置:Xms4g Xmx4g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:40418.84/sec
```

```
吞吐量:40418.84/sec
GC次数:24

平均耗时:2ms,最大耗时:3ms
```

4. 串行GC

```
内存配置:Xms512m Xmx512m
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:35356.07/sec
GC次数:397
平均耗时:2ms,最大耗时:4ms
```

```
内存配置:Xms1g Xmx1g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:35356.07/sec
GC次数:397
平均耗时:2ms,最大耗时:4ms
```

```
内存配置:Xms1g Xmx1g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:35588.52/sec
GC次数:201
平均耗时:2ms,最大耗时:11ms
```

```
内存配置:Xms2g Xmx2g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
吞吐量:38545.56/sec
GC次数:106
平均耗时:3ms,最大耗时:16ms
```

```
内存配置:Xms4g Xmx4g
模拟参数:wrk -t8 -d60s -c20 http://localhost:8088/api/hello
```

吞吐量:35893.11/sec
GC次数:51

平均耗时:4ms,最大耗时:12ms

5. 总结

(1)吞吐量最高和稳定性最好的并行GC,相对比较平稳。
(2)CMSGC在内存比较大时,暂停时间可能增大,CMSGC不是很合适配置很大堆内存,在并发标记过程中有初始化标记和最终标记是需要暂停的,当对象生成速率比较快的时会造成堆内存很快会满,内存比较大的话可能造成标记量比较大暂停时间会比较长。
(3)串行GC整体吞吐量相对比较一般,暂停时间比较长,不是适合多核时代
(4)G1GC的暂停时间和吞吐量相对比较平均,更加可控