

01-测试目的

通过模拟高并发的场景观察 jvm 内部内存、gc 的情况，通过不同的 jvm 参数配置，观察不同配置下应用的性能。

02-测试工具

Grafana+Jmeter5.4.1+Prometheus2.15.1+Micrometer

03-测试环境

机器	配置
hero 应用服务器	4C8G
mysql 数据库服务器	4C8G
Grafana、Jmeter、Prometheus 部署服务器	4C8G

04-测试场景

04-1 默认参数启动



启动参数仅输出 GC 日志：

```
JAVA_OPT="{JAVA_OPT} -Xms1096m -Xmx1096m -Xmn408m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m"
JAVA_OPT="{JAVA_OPT} -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
XX:+PrintHeapAtGC -Xloggc:{BASE_DIR}/logs/gc-best-heap-metaspace.log"
```

jvm 内存、吞吐量与延时、fullgc 次数如图：

JVM memory size

(To learn about JVM Memory, [click here](#))



Generation	Allocated 	Peak 
Young Generation	612 mb	466 mb
Old Generation	79 mb	18.73 mb
Meta Space	1.03 gb	32.6 mb
Young + Old + Meta space	2.87 gb	517.33 mb

Key Performance Indicators

(Important section of the report. To learn more about KPIs, [click here](#))

1 Throughput  : 95.261%

2 Latency:

Avg Pause GC Time 	15.0 ms
Max Pause GC Time 	70.0 ms

GCPauseDuration Time Range 

Duration (ms)		No. of GCs	Percentage
<u>10</u>	ms 		
			
	0 - 10	8	66.67%
	10 - 20	3	25.0%
	70 - 80	1	8.33%

70 - 80ms

10 - 20ms

0 - 10ms

Total GC stats	Minor GC stats	Full GC stats	GC Pause Statistics
Total GC count ?	Minor GC count	Full GC Count	Pause Count
Total reclaimed bytes ?	Minor GC reclaimed ?	Full GC reclaimed ?	Pause total time
Total GC time ?	Minor GC total time	Full GC total time	Pause avg time ?
Avg GC time ?	Minor GC avg time ?	Full GC avg time ?	Pause avg time std dev
GC avg time std dev	Minor GC avg time std dev	Full GC avg time std dev	Pause min/max time
GC min/max time	Minor GC min/max time	Full GC min/max time	
GC Interval avg time ?	Minor GC Interval avg ?	Full GC Interval avg ?	


通过 gc 日志分析，项目启动时，就分配了 1.03gb 的元空间，并且进行了两次 fullGc,这是不合理的。

04-2 选用合适的堆内存参数配置

启动参数如下：

```
JAVA_OPT="${JAVA_OPT}" -Xms1096m -Xmx1096m -Xmn408m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m"
JAVA_OPT="${JAVA_OPT}" -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -
XX:+PrintHeapAtGC -Xloggc:${BASE_DIR}/logs/gc-best-heap-metaspace.log"
```

jvm 内存、吞吐量与延时、fullgc 次数如图：



JVM memory size

(To learn about JVM Memory, [click here](#))

Generation	Allocated ?	Peak ?
Young Generation	408 mb	403.25 mb
Old Generation	688 mb	137.91 mb
Meta Space	128 mb	n/a
Young + Old + Meta space	1.2 gb	540.84 mb

🔑 Key Performance Indicators

(Important section of the report. To learn more about KPIs, [click here](#))

❶ Throughput🔗 : 99.972%

❷ Latency:

Avg Pause GC Time 🔗	9.87 ms
Max Pause GC Time 🔗	30.0 ms

GCPauseDuration Time Range 🔗:

Duration (ms)	No. of GCs	Percentage
10 ms ▾ Change		
0 - 10	137	90.73%
10 - 20	11	7.28%
20 - 30	3	1.99%

20 - 30r

10 - 20r

0 - 10r

Total GC stats

Total GC count 🔗	151
Total reclaimed bytes 🔗	n/a
Total GC time 🔗	1 sec 490 ms
Avg GC time 🔗	9.87 ms
GC avg time std dev	5.27 ms
GC min/max time	0 / 30.0 ms
GC Interval avg time 🔗	35 sec 371 ms

Minor GC stats

Minor GC count	151
Minor GC reclaimed 🔗	55.23 gb
Minor GC total time	1 sec 490 ms
Minor GC avg time 🔗	9.87 ms
Minor GC avg time std dev	5.27 ms
Minor GC min/max time	0 / 30.0 ms
Minor GC Interval avg 🔗	35 sec 371 ms

Full GC stats

Full GC Count	0
Full GC reclaimed 🔗	n/a
Full GC total time	n/a
Full GC avg time 🔗	n/a
Full GC avg time std dev	n/a
Full GC min/max time	n/a
Full GC Interval avg 🔗	n/a

GC Pause Statistics

Pause Count	151
Pause total time	1 sec 490 ms
Pause avg time 🔗	9.87 ms
Pause avg time std dev	0.0
Pause min/max time	0 / 30.0 ms

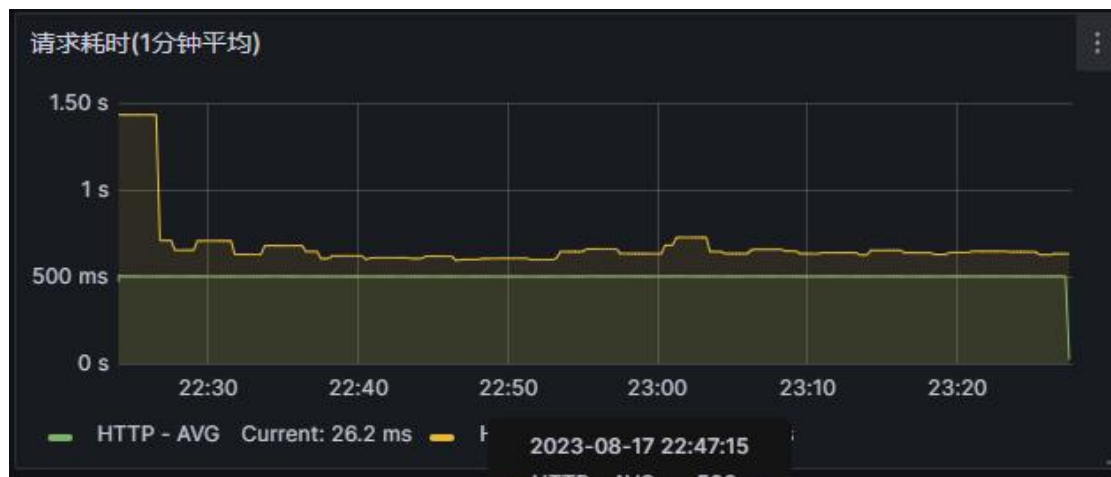
结论: 通过合适的堆内存、元空间参数的配置, 发现 gc 的吞吐量从 95.261%上升到 99.972%, 平均时延由 70ms 降低到 30ms。并且没有 fullGc。

04-3 配置 PS+PO,模拟高负载

启动参数配置:

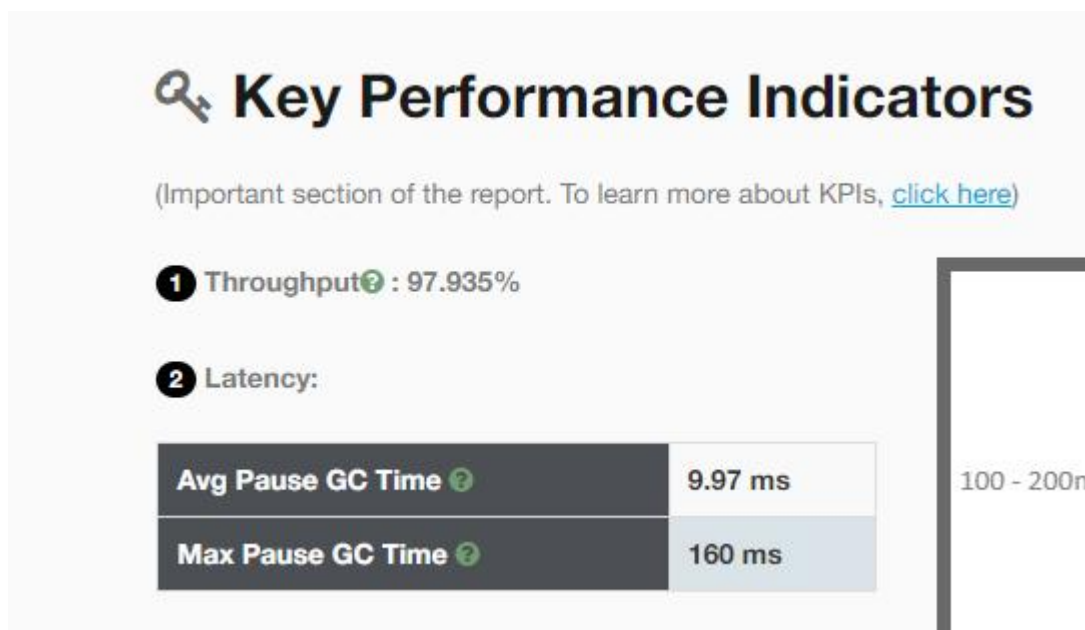
```
JAVA_OPTS="${JAVA_OPTS} -Xms256m -Xmx256m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k"
JAVA_OPTS="${JAVA_OPTS} -XX:+UseParallelGC -XX:+UseParallelOldGC "
JAVA_OPTS="${JAVA_OPTS} -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -
XX:+PrintHeapAtGC -Xloggc:${BASE_DIR}/logs/gc-ps-po.log"
```

通过 Grafana 观测 RT,TPS,JVM 内存如图:





通过 GcEasy 观测垃圾回收的吞吐量和延时



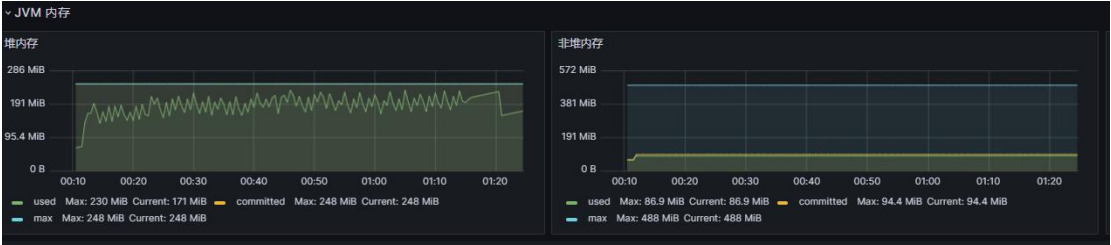
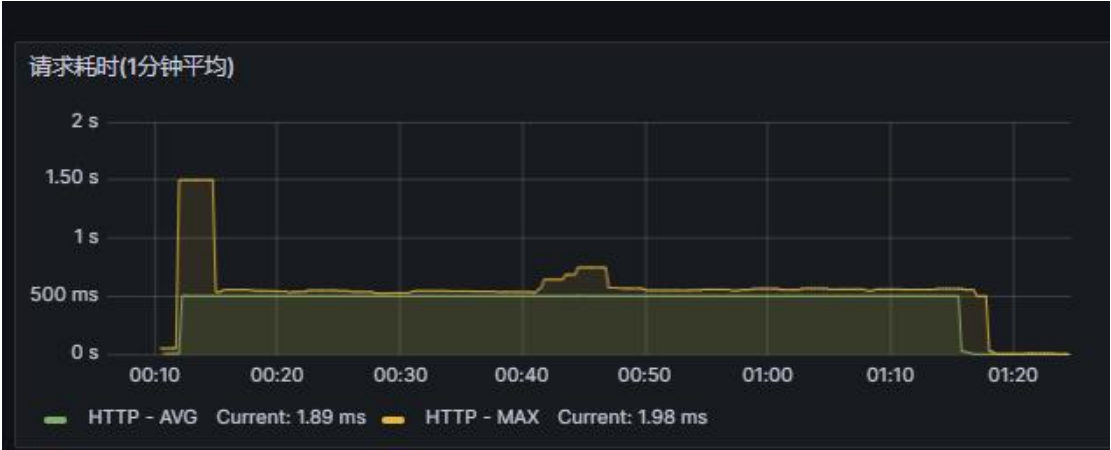
Total GC stats	Minor GC stats	Full GC stats	GC Pause Statistics
Total GC count ②	Minor GC count	Full GC Count	Pause Count
8071	7945	126	8071
Total reclaimed bytes ②	Minor GC reclaimed ②	Full GC reclaimed ②	Pause total time
592.62 gb	589.59 gb	3.03 gb	1 min 20 sec 449 ms
Total GC time ②	Minor GC total time	Full GC total time	Pause avg time ②
1 min 20 sec 449 ms	1 min 10 sec 227 ms	10 sec 220 ms	9.97 ms
Avg GC time ②	Minor GC avg time ②	Full GC avg time ②	Pause avg time std dev
9.97 ms	8.84 ms	81.1 ms	0.0
GC avg time std dev	Minor GC avg time std dev	Full GC avg time std dev	Pause min/max time
9.70 ms	3.32 ms	13.8 ms	0 / 160 ms
GC min/max time	Minor GC min/max time	Full GC min/max time	
0 / 160 ms	0 / 30.0 ms	70.0 ms / 160 ms	
GC interval avg time ②	Minor GC interval	Full GC interval avg ②	
482 ms	490 ms	29 sec 644 ms	

04-4 配置 parNew+CMS,模拟高负载

启动参数配置:

```
JAVA_OPT="${JAVA_OPT} -Xms256m -Xmx256m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k"
JAVA_OPT="${JAVA_OPT} -XX:+UseParNewGC -XX:+UseConcMarkSweepGC "
JAVA_OPT="${JAVA_OPT} -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -
XX:+PrintHeapAtGC -Xloggc:${BASE_DIR}/logs/gc-parnew-cms.log"
```

通过 Grafana 观测 RT,TPS,JVM 内存如图：



通过 GcEasy 观测垃圾回收的吞吐量和延时

🔑 Key Performance Indicators

(Important section of the report. To learn more about KPIs, [click here](#))

❶ Throughput📈 : 97.811%

❷ Latency:

Avg Pause GC Time 📈	10.0 ms
Max Pause GC Time 📈	190 ms

100 - 200m:

GCPauseDuration Time Range 📈:

Duration (ms)		No. of GCs	Percentage
100	ms ▾ Change		
0 - 100		9242	99.98%
100 - 200		2	0.02%

0 - 100m:

	Concurrent Abortable Preclean	Young GC 📈	Concurrent Mark	Final Remark 📈	Concurrent Sweep	Concurrent Preclean	Initial Mark 📈	Full GC 📈	Concurrent Reset
Total Time 📈	1 min 36 sec 720 ms	1 min 27 sec 90 ms	8 sec 930 ms	4 sec 970 ms	3 sec 20 ms	790 ms	620 ms	190 ms	40.0 ms
Avg Time 📈	537 ms	9.81 ms	49.3 ms	27.6 ms	16.8 ms	4.36 ms	3.43 ms	190 ms	0.222 ms
Std Dev Time	136 ms	3.86 ms	5.72 ms	4.98 ms	6.12 ms	4.96 ms	4.75 ms	0	1.47 ms
Min Time 📈	130 ms	0	40.0 ms	20.0 ms	10.0 ms	0	0	190 ms	0
Max Time 📈	780 ms	180 ms	60.0 ms	60.0 ms	40.0 ms	10.0 ms	10.0 ms	190 ms	10.0 ms
Interval Time 📈	21 sec 41 ms	477 ms	20 sec 928 ms	21 sec 39 ms	21 sec 39 ms	20 sec 928 ms	20 sec 928 ms	n/a	21 sec 39 ms
Count 📈	180	8882	181	180	180	181	181	1	180

04-5 配置 G1，模拟高负载

启动参数配置：

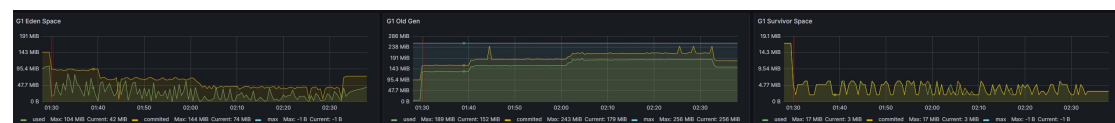
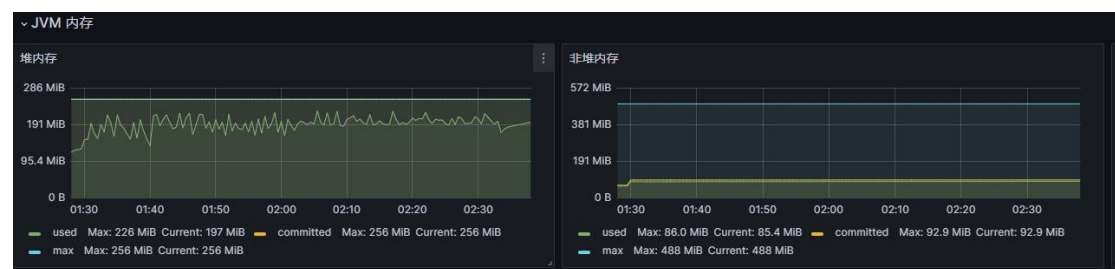
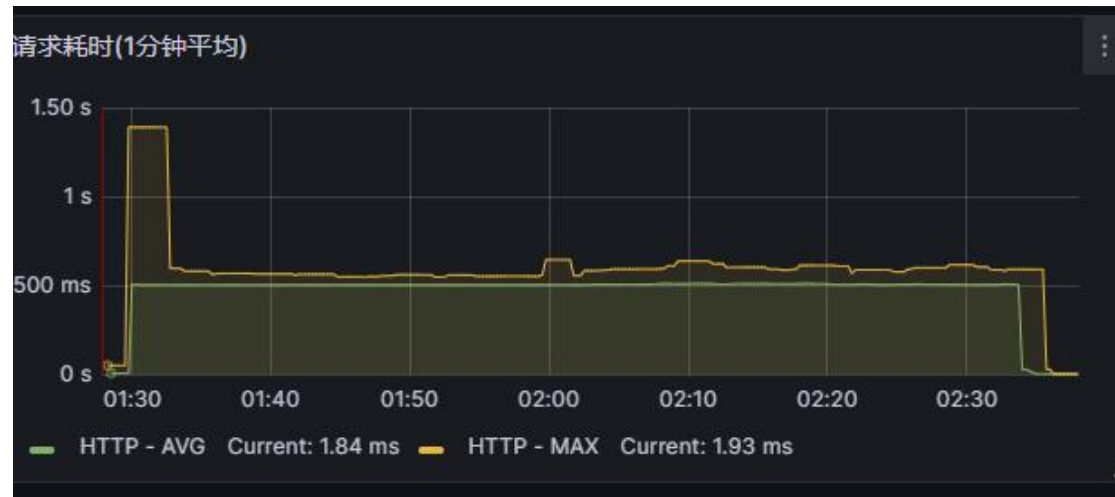
```
JAVA_OPT="${JAVA_OPT} -Xms256m -Xmx256m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -Xss512k"
```

```
JAVA_OPT="${JAVA_OPT} -XX:+UseG1GC -XX:MaxGCPauseMillis=100"
```



```
JAVA_OPTS="${JAVA_OPTS} -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:${BASE_DIR}/logs/gc-g-one.log"
```

通过 Grafana 观测 RT,TPS,JVM 内存如图:



通过 GcEasy 观测垃圾回收的吞吐量和延时

05 测试结果

通过 04-1 与 04-2 的测试对比，合适的堆内存、元空间配置能有效的减少 FullGc 的次数，提高垃圾回收的吞吐量，降低垃圾回收的时延。

通过 04-3、04-4、04-5 的测试对比，发现 jvm 的 gc 收集器性能都很强大，各个 gc 收集器表现都很好。Ps+po 的吞吐量稍高，parnew+cms 能将 RT 的最大值波动降下来，g1 在各个方面都表现很好。