测试机器配置：4核16g内存 jdk版本8

一、先分析四种gc，生成大量对象的情境下的情况：

1、串行GC:

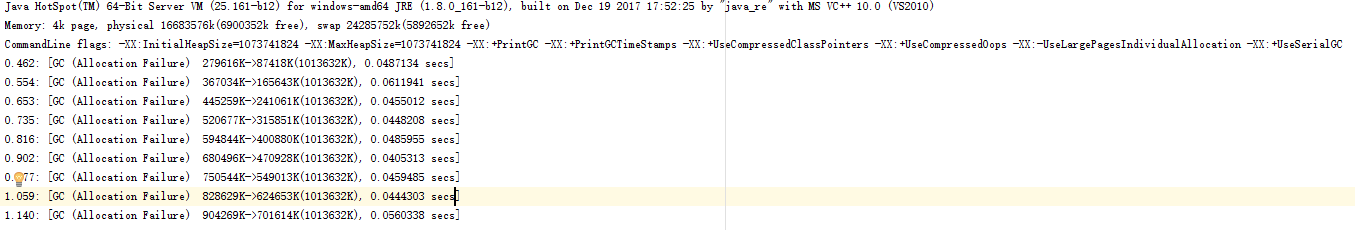
堆初始空间和最大空间都设置1g如下：

-XX:+UseSerialGC

-Xms1g

-Xmx1g

-Xloggc:gc.Serial.log



执行结束!共生成对象次数:9565

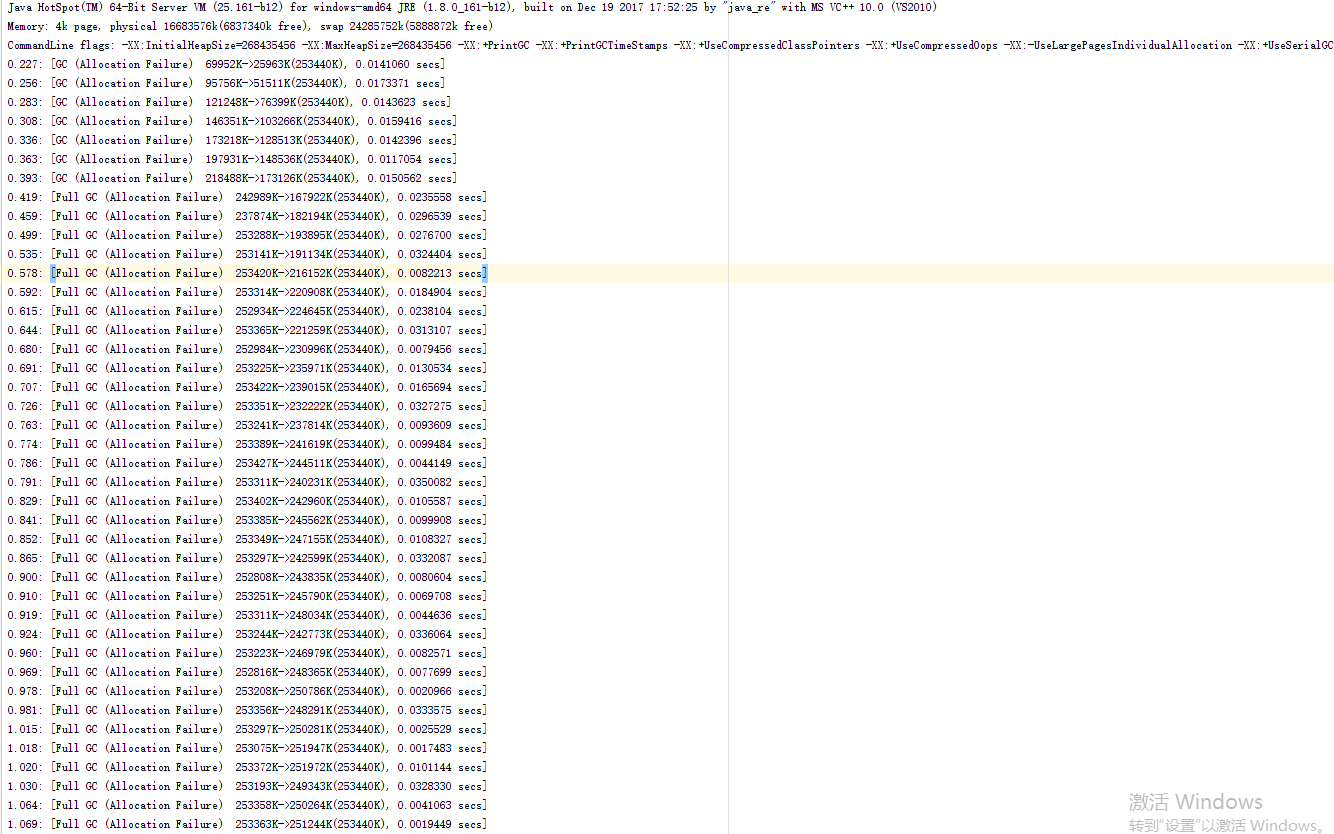
堆初始空间和最大空间都设置256m如下：

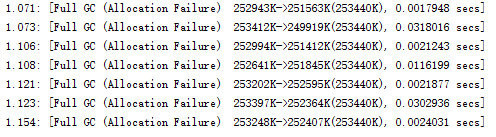
-XX:+UseSerialGC

-Xms256m

-Xmx256m

-Xloggc:gc.Serial.log





执行结束!共生成对象次数:4488

可以看到堆空间会直接影响串行gc，堆设置的大，对象生成约多，gc次数也越少；

堆设置的小，对象生成也变少了，gc次数也增加很多，并且几乎后面全是full gc，概率出现oom；

1g堆空间的gc之后剩余的数据量相比上次gc的会增长一些，可能是将年轻代的数据移动到老年代；256m堆空间后面几乎全是full gc，gc之后剩余数据量比较恒定，可见是一直清理老年代；256m堆空间相比1g堆空间情况，执行时间更长一些；

2、并行gc：

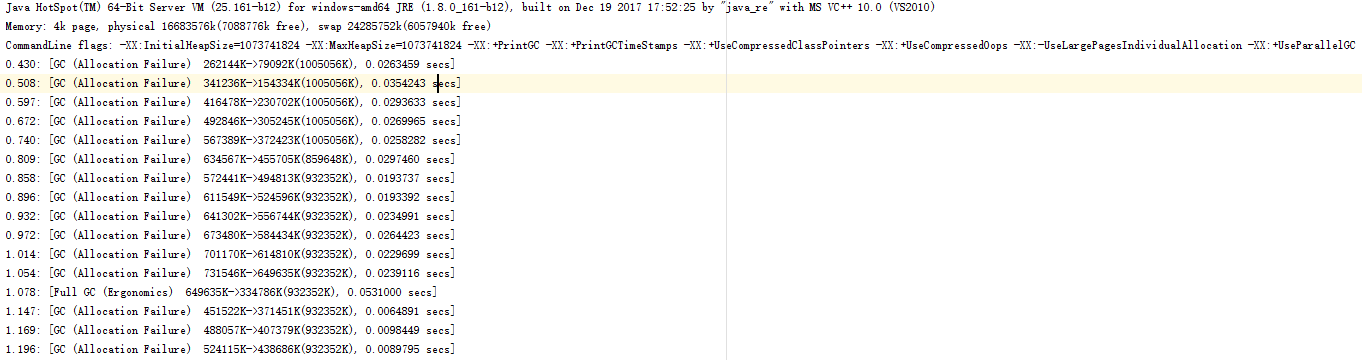
堆初始空间和最大空间都设置1g如下：

-XX:+UseParallelGC

-Xms1g

-Xmx1g

-Xloggc:gc.Parallel.log



执行结束!共生成对象次数:9886

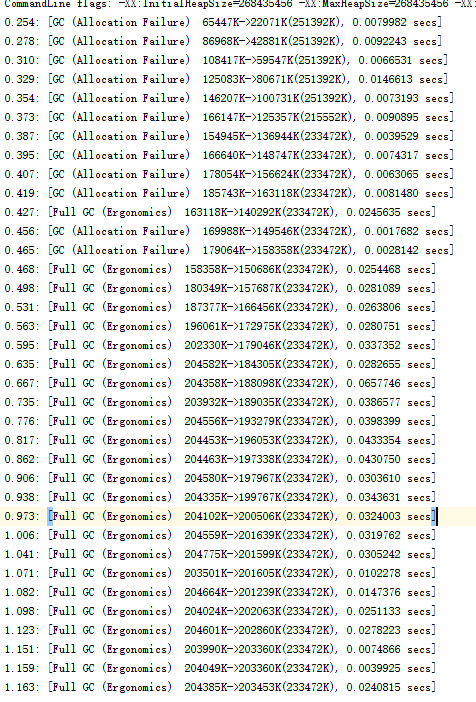
堆初始空间和最大空间都设置256m如下：

-XX:+UseParallelGC

-Xms256m

-Xmx256m

-Xloggc:gc.Parallel.log



执行结束!共生成对象次数:2988

可以看到同样设置堆大小时，并行gc和串行gc很相似，和串行gc相比，并行gc堆比较小的情境下，生成的对象更少；

1. CMS gc：

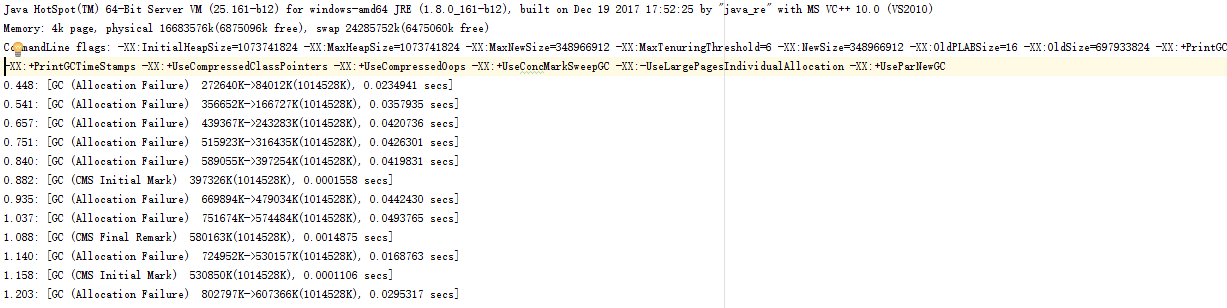
堆初始空间和最大空间都设置1g如下：

-XX:+UseConcMarkSweepGC

-Xms1g

-Xmx1g

-Xloggc:gc.CMS.log



执行结束!共生成对象次数:12448

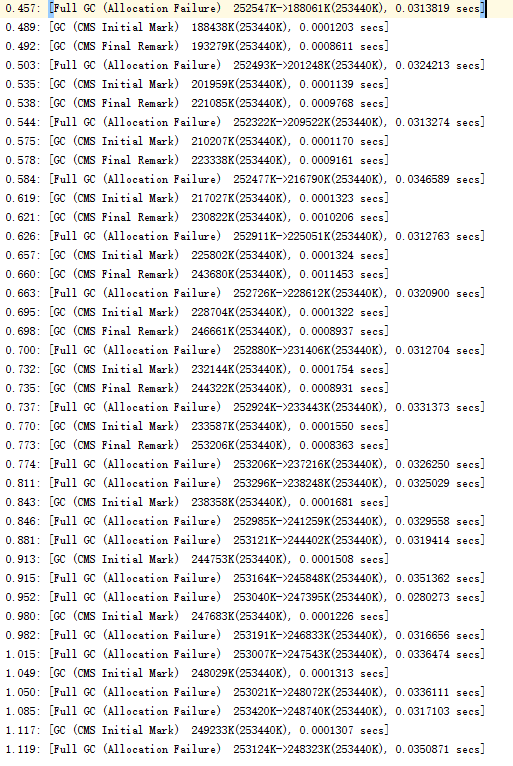
堆初始空间和最大空间都设置256m如下：

-XX:+UseConcMarkSweepGC

-Xms256m

-Xmx256m

-Xloggc:gc.CMS.log



执行结束!共生成对象次数:4644

可以看到，堆空间小的时候，相比大的堆空间，生成对象更少，gc次数更多，cms gc次数也更多，随着对象生成，后面几乎很多次full gc；堆空间比较大的时 gc 次数也少；

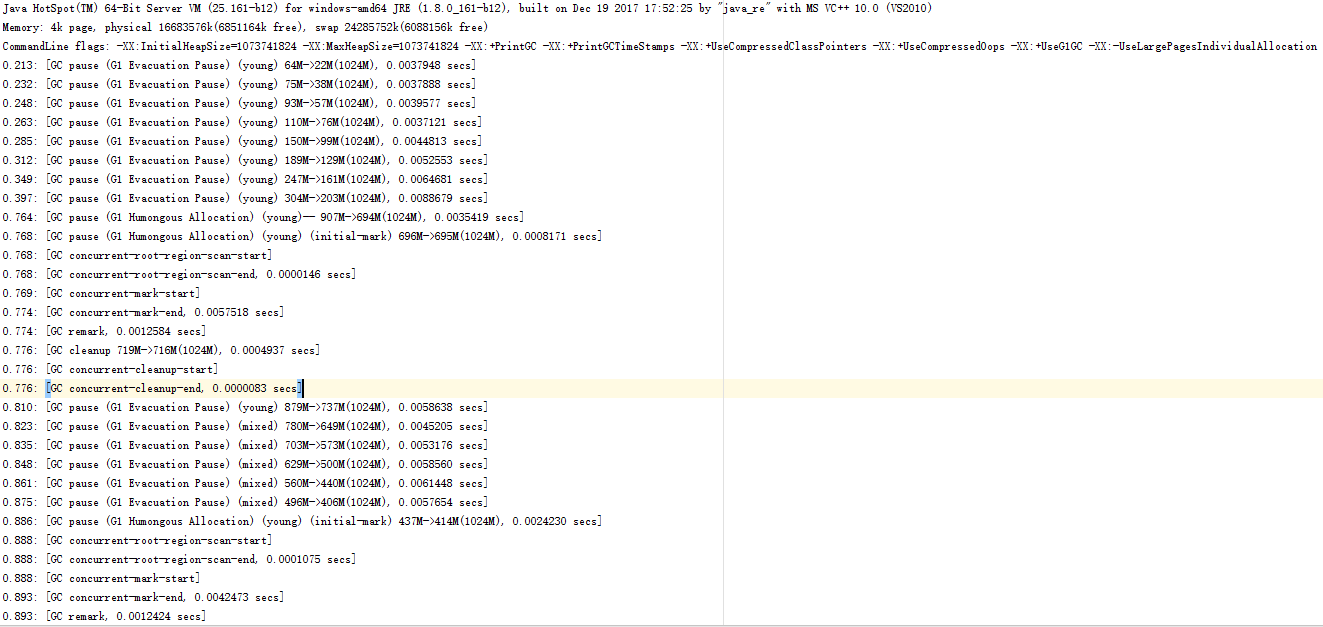
1. G1 gc:

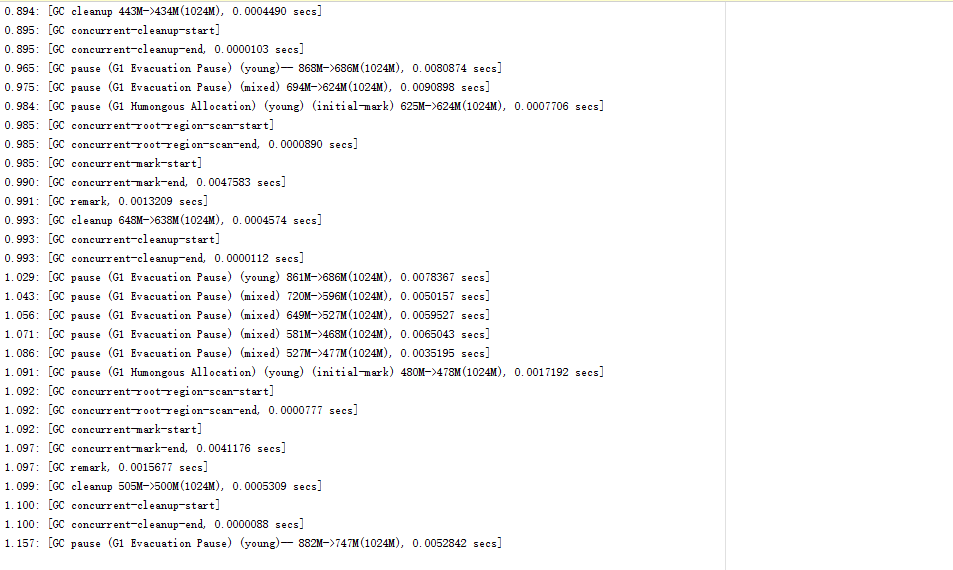
堆初始空间和最大空间都设置1g如下：

-XX:+UseG1GC

-Xms1g

-Xmx1g

-Xloggc:gc.G1.log



执行结束!共生成对象次数:11678

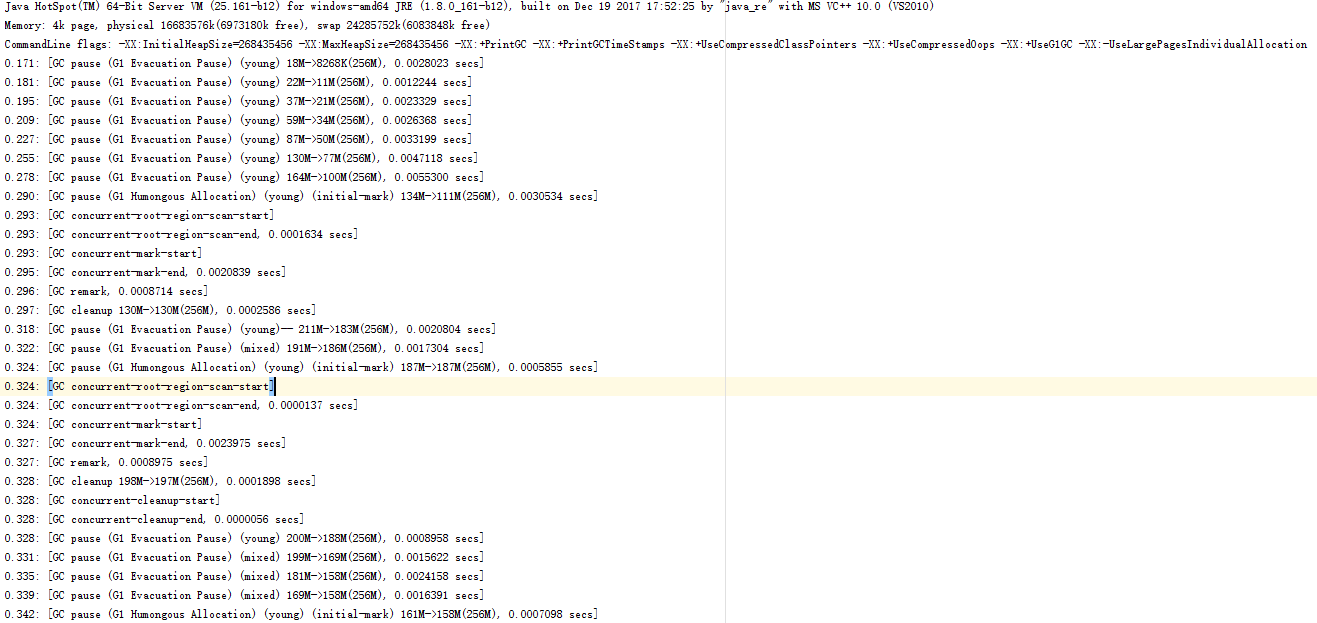
堆初始空间和最大空间都设置256m如下：

-XX:+UseG1GC

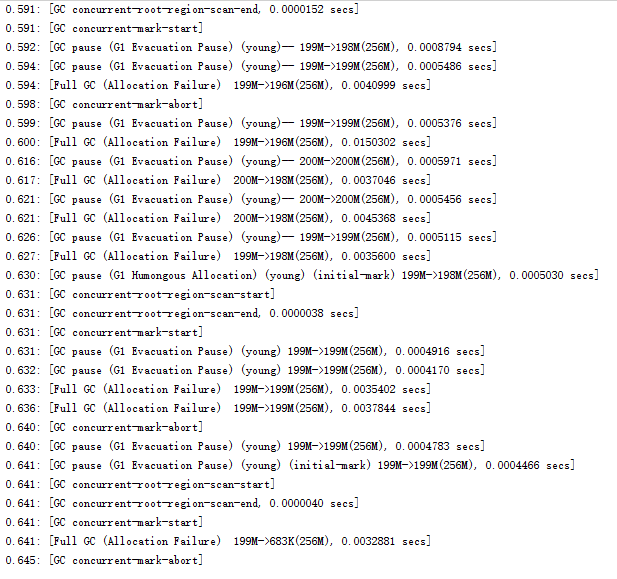
-Xms1g

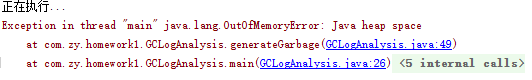
-Xmx1g

-Xloggc:gc.G1.log



...





可以看到设置256m堆大小时，gc次数更多，也包括full gc，执行了10次，10次最后全部直接oom了，而设置1g的堆则正常；

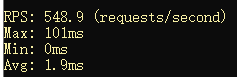
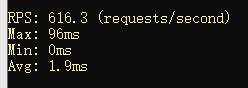
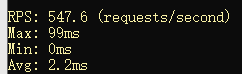
内存越大，G1 gc越有优势；在内存比较小的情况下，可以看到四种gc生成的对象相比大的堆空间更少，并且并行gc的平均耗时更短；

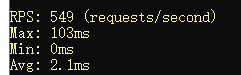
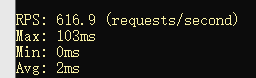
1. 再分析四种gc，大量请求的情境下的情况：

以下取部分测试结果，请求数1000，并发数20

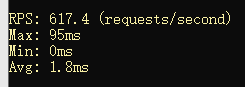
1. 串行gc：

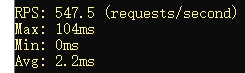


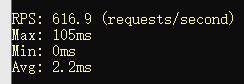
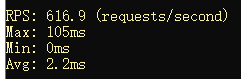








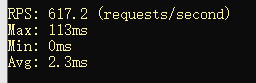
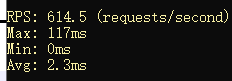


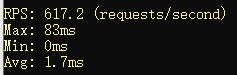
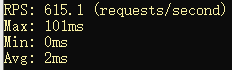


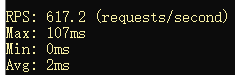
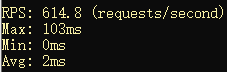
4核16g内存 jdk版本8下，在串行gc情况下，256m和1g堆空间，rps几乎无差别，几乎rps都是在540~620之间浮动；

2、并行gc：



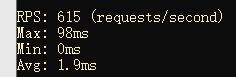
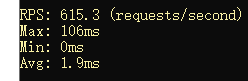


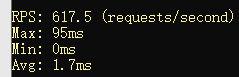
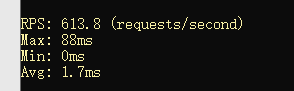


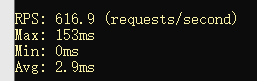
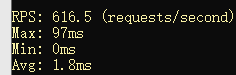


1g堆空间时最大时间131ms，rps268.7







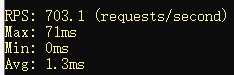
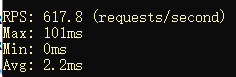


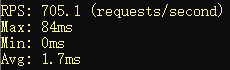
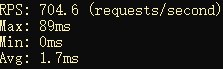
256m堆空间时最大时间90ms，rps245

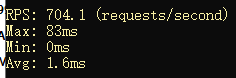
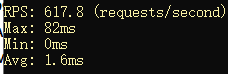
4核16g内存 jdk版本8下，并行gc 1g堆空间和256m堆空间rps几乎一致

3、CMS gc：

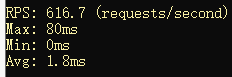
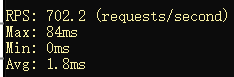


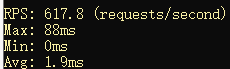
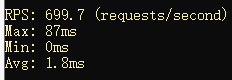


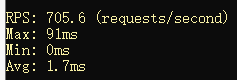
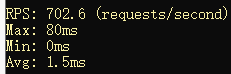








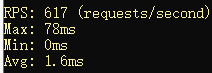
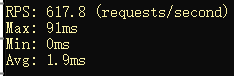


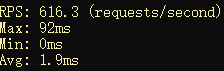
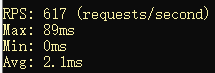


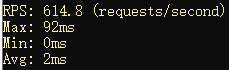
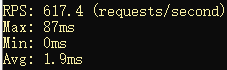
可以看到CMS gc相比串行和并行的平均rps更高一些；

4、G1 gc:

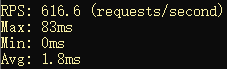
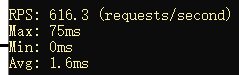


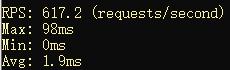


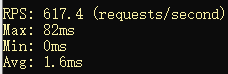
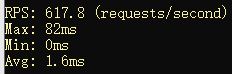












可以看到G1 gc 处理请求能力不如CMS gc；

整体上看，内存不确定，对于处理请求性能要求不高的场景，优先选择并行gc；

如果以处理并发请求优先，不考虑内存大小的场景，则选择CMS gc；