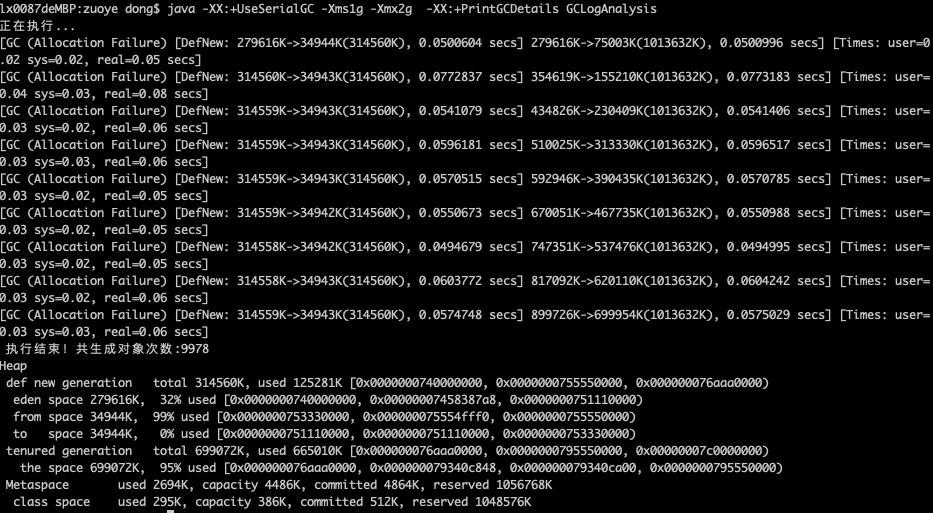
1. 使用GCLogAnlysis.java演示串行/并行/CMS/G1的案例 （机器：mac）

    1.1 串行GC：

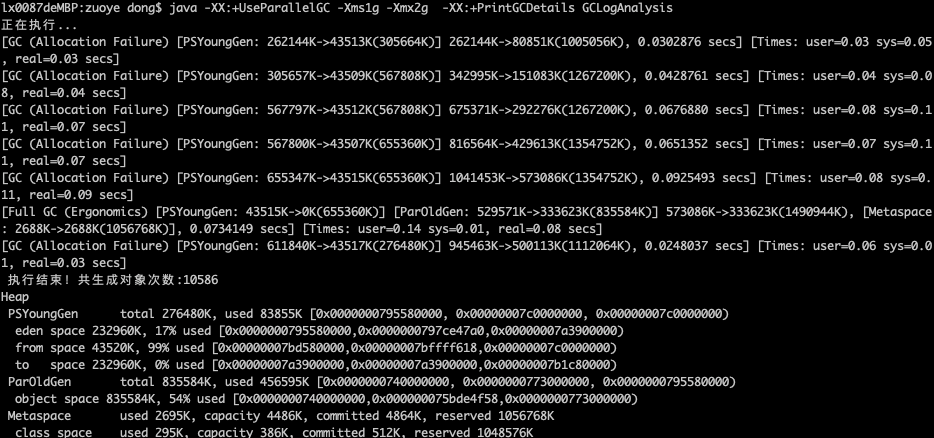
        执行命令：java -XX:+UseSerialGC -Xms1g -Xmx2g  -XX:+PrintGCDetails GCLogAnalysis



        共生成对象次数：9978

    1.2 并行GC:

        执行命令：java -XX:+UseParallelGC -Xms1g -Xmx2g  -XX:+PrintGCDetails GCLogAnalysis



        共生成对象次数：10586

    1.3 CMS:

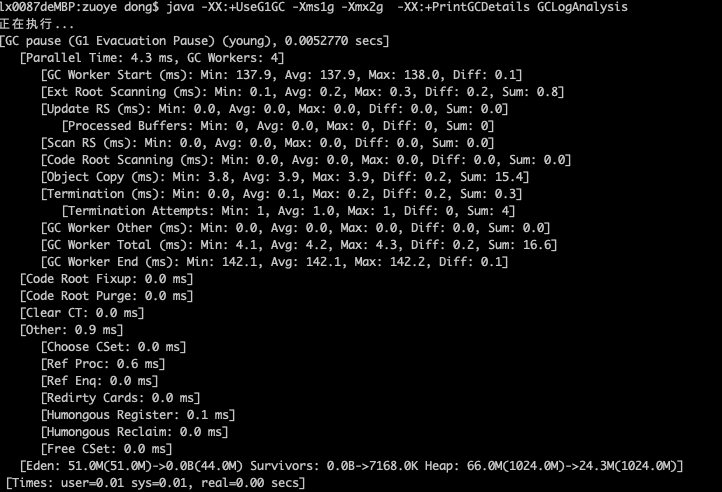
        执行命令：java -XX:+UseConcMarkSweepGC -Xms1g -Xmx2g  -XX:+PrintGCDetails GCLogAnalysis

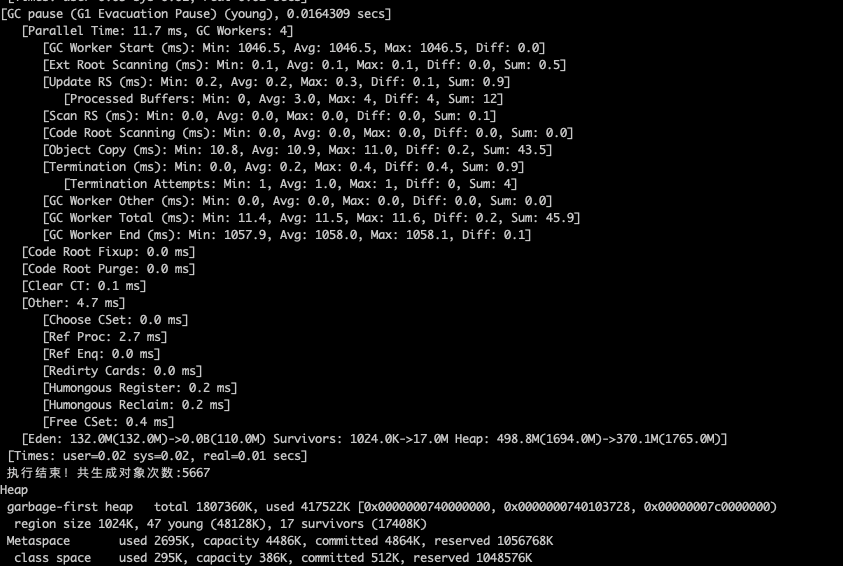


        共生成对象次数：9915

    1.4 G1:

        执行命令：java -XX:+UseG1GC -Xms1g -Xmx2g  -XX:+PrintGCDetails GCLogAnalysis

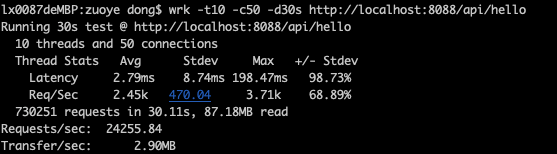




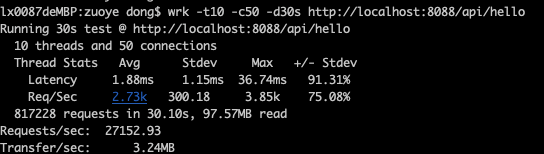
        共生成对象次数：5667

2. 使用压测工具演练gateway-server-0.0.1-SNAPSHOT.jar （wrk）

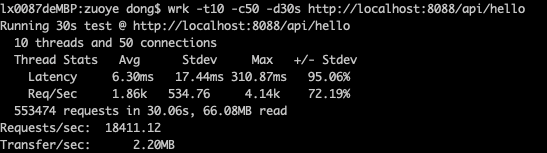
2.1 串行GC java -XX:+UseSerialGC -Xms1g -Xmx2g gateway-server-0.0.1-SNAPSHOT.jar



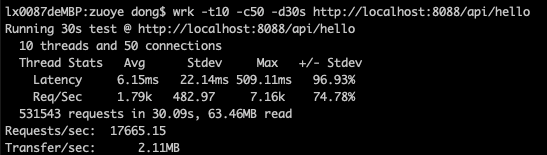
2.2 并行GC java -XX:+UseParallelGC -Xms1g -Xmx2g -jar gateway-server-0.0.1-SNAPSHOT.jar



2.3 CMS: java -XX:+UseConcMarkSweepGC -Xms1g -Xmx2g -jar gateway-server-0.0.1-SNAPSHOT.jar



2.4 G1： java -XX:+UseG1GC -Xms1g -Xmx2g -jar gateway-server-0.0.1-SNAPSHOT.jar



3. 总结：

3.1 串行GC

单个线程执行垃圾回收，此时用户线程处于等待状态，使用复制算法

适合小内存和逻辑简单的应用，当运行内存较大时，不建议使用，当并发高时，GC引起的延迟比较高

3.2 并行GC

多线程处理垃圾收集，此时用户线程仍处于等待状态

是jdk8默认垃圾回收器，适用于多核服务器，可以提高吞吐量，多个线程并行收集，GC延迟相对较小

3.3 CMS

一般用于老年代，和并行GC搭配使用

是一种以低延迟为目标的垃圾回收器，在回收过程冲，仅初始标记和最终标记两部分需要短暂暂停用户线程。

比并行GC占用更多的cpu资源，吞吐量会有所下降（15%左右）

3.4 G1

主要针对大容量内存机器（6G以上）

JDK11默认垃圾回收器，适用于新生代和老年代

将内存分为一个个的region，每次回收可以精确到region， 稳定延迟暂停小于0.5s