串行GC

```
java -XX:+UseSerialGC -Xms512m -Xmx512m -Xloggc:gc.demo.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps GCLogAnalysis
```

特点

最基本GC，单线程操作，简单，但是 Stop the world 明显，不适合服务器使用

测试结果

> 生成对象次数：7873次
>
> GC次数：15次
>
> 最后几次GC没有任何作用，并没有内存减少
>
> 时间: 0.03 secs

并行GC

```
java -XX:+UseParallelGC -Xms512m -Xmx512m -Xloggc:gc.demo.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps GCLogAnalysis
```

特点

多GC线程并行工作，关注系统吞吐量，没有GC时不会消耗系统资源

测试结果

> 生成对象次数：7600次
>
> GC 次数：32次，其中 Full GC 次数：6次
>
> Full GC 的时候， YoungGen 减少到0，OldGen 减少的幅度不断缩小
>
> 时间: 0.05 secs

CMS GC

Concurrent Mark Sweep，并发标记清除。

```
java -XX:+UseConcMarkSweepGC -Xms512m -Xmx512m -Xloggc:gc.demo.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps GCLogAnalysis
```

- **特点**

也是多线程的，尽可能地缩短 GC 时用户线程的停顿时间

- **测试结果**

> 生成对象次数：9013次
>
> GC次数：23次，其中CMS GC次数：6次
>
> 最后几次GC没有任何作用，并没有内存减少
>
> 时间: 0.03 secs

G1 GC

```
java -XX:+UseG1GC -Xms512m -Xmx512m -Xloggc:gc.demo.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps GCLogAnalysis
```

- **特点**

能独立管理整个GC堆（新生代和老年代），将整个堆划分为多个大小相等的独立区域（Region），能充分利用多CPU、多核环境下的硬件优势； 可以并行来缩短"Stop The World"停顿时间；也可以并发让垃圾收集与用户程序同时进行；

在下面的情况时，使用 G1 可能比 CMS 好：

1. 超过 50% 的 Java 堆被活动数据占用；
2. 对象分配频率或年代提升频率变化很大；
3. GC 停顿时间过长（长于0.5至1秒）。

- **测试结果**

> 生成对象次数：8371次
>
> GC次数：23次，其中CMS GC次数：6次
>
> 最后几次GC没有任何作用，并没有内存减少
>
> 时间: 0.03 secs