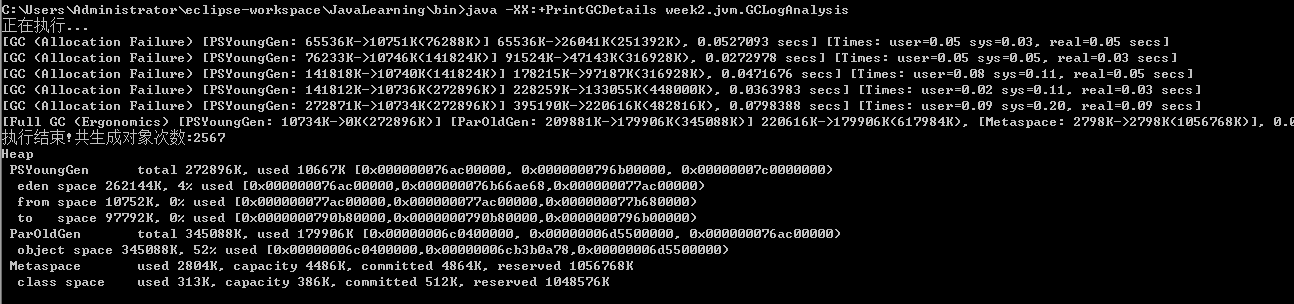
1. **使用 GCLogAnalysis.java 自己演练一遍串行/并行/CMS/G1的案例。**

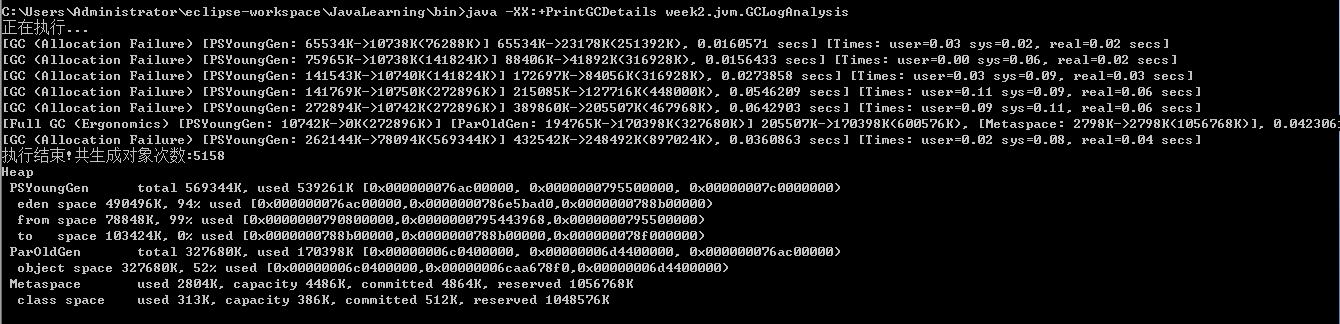
默认GC是ParallelGC还有就是默认的对大小配置Max heap size 4G, Initial heap size 268M





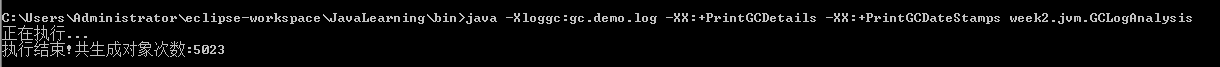
默认GC(ParallelGC)





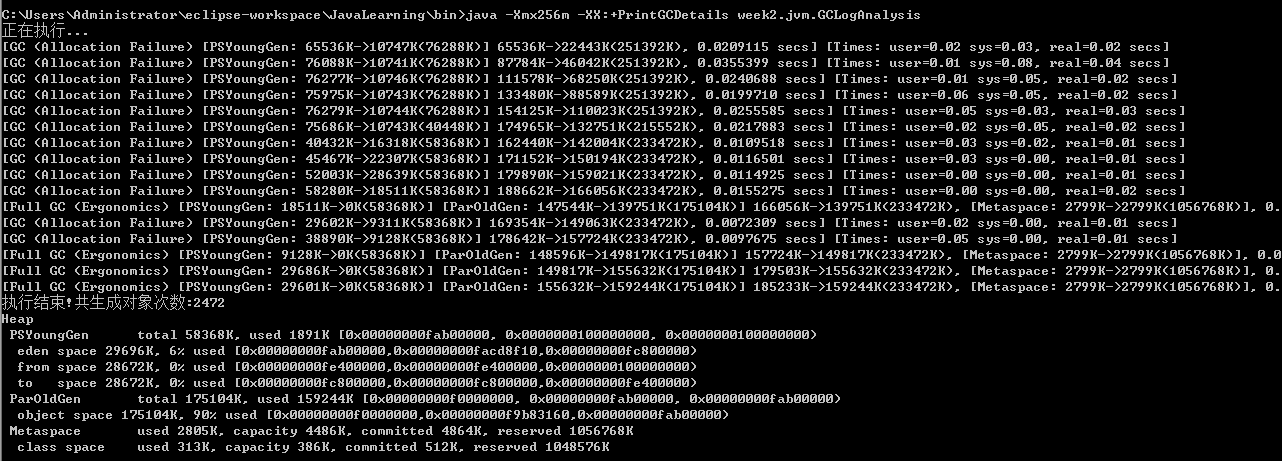
默认GC(ParallelGC), 打印详细信息

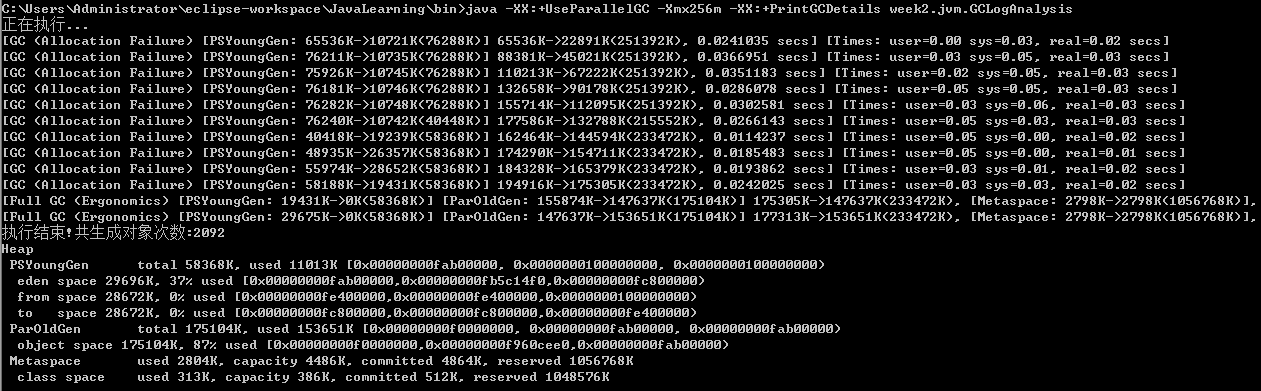
java -Xloggc:gc.demo.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps week2.jvm.GCLogAnalysis

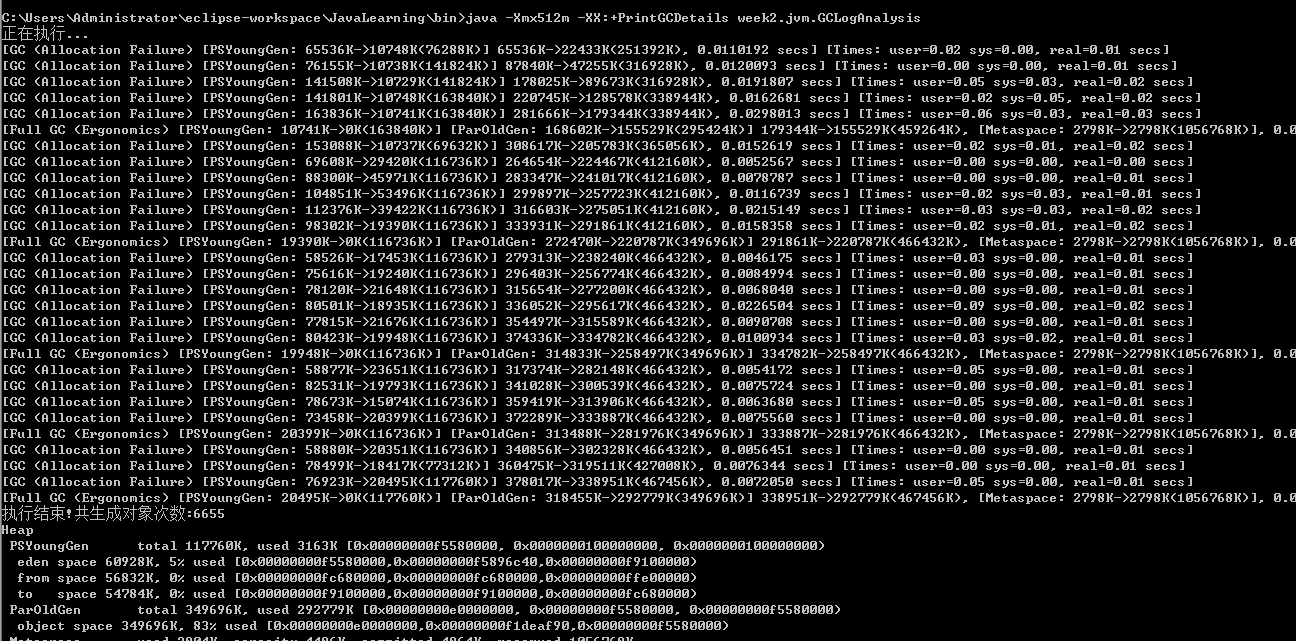


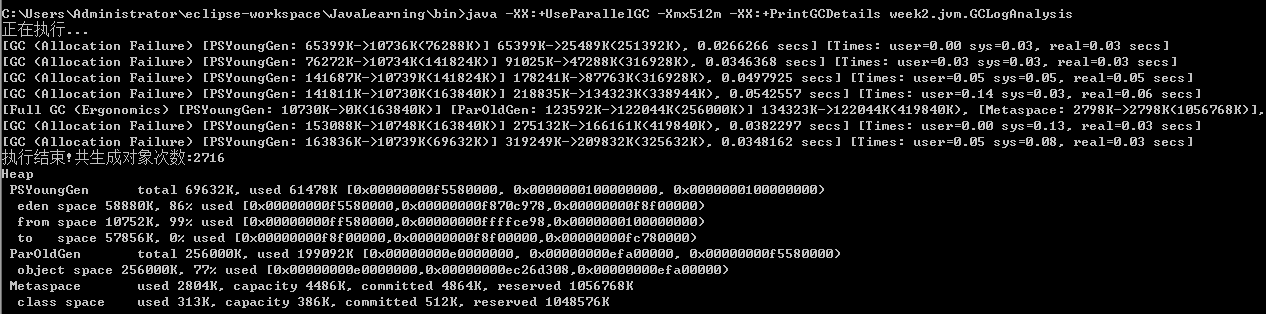
**ParallelGC**

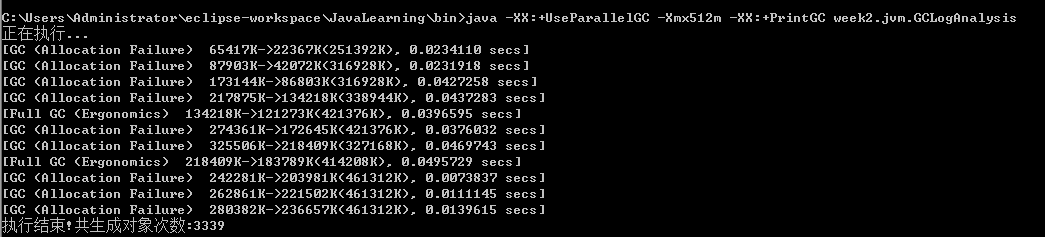
**256M**



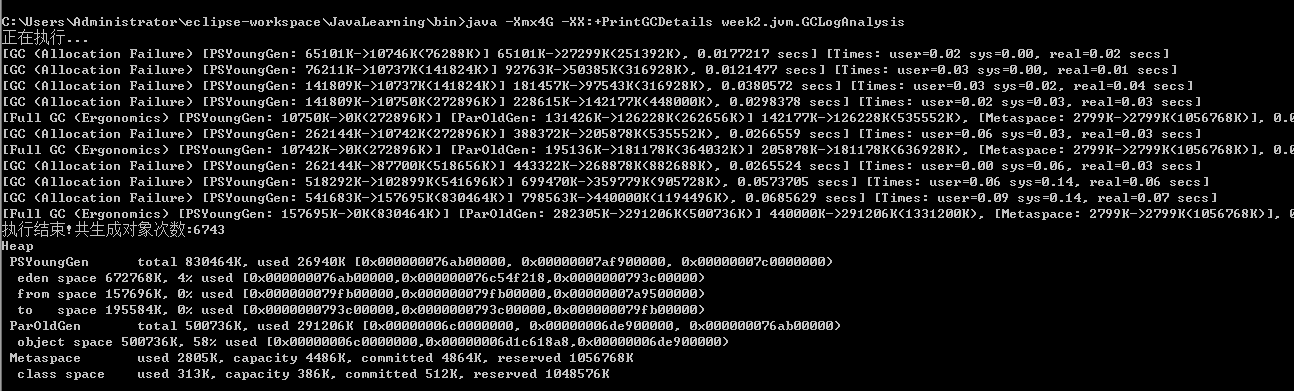


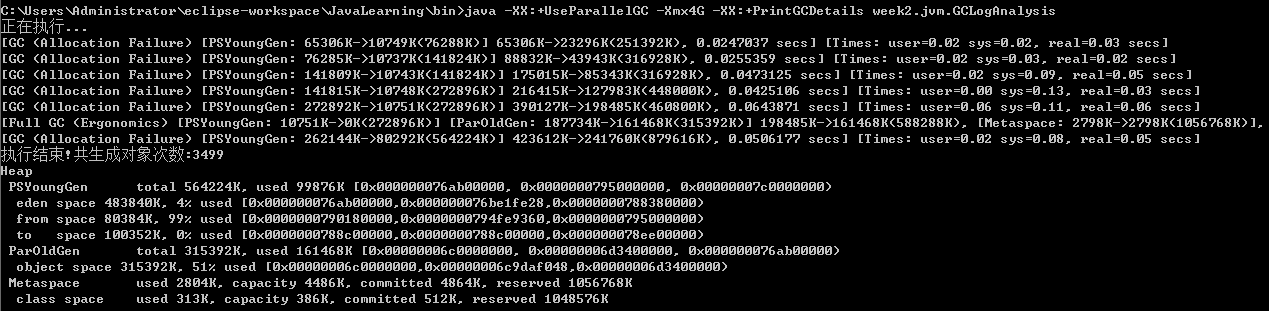
**512M** 

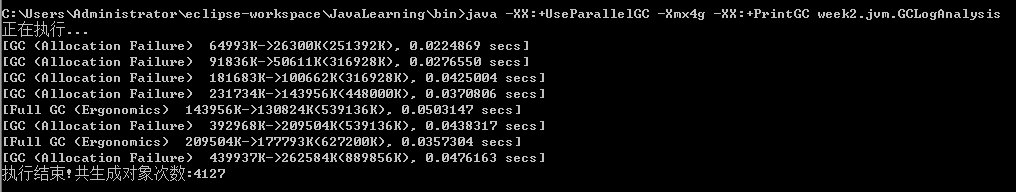




**4G**

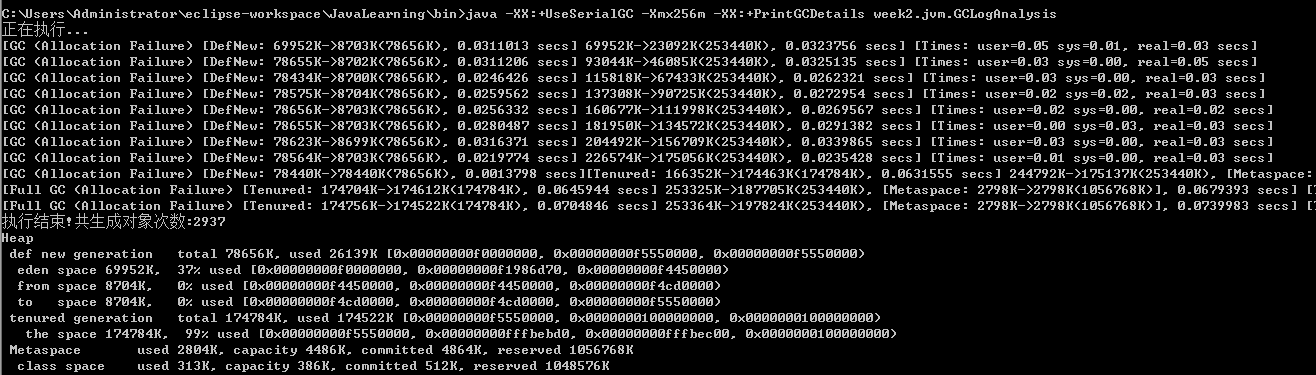


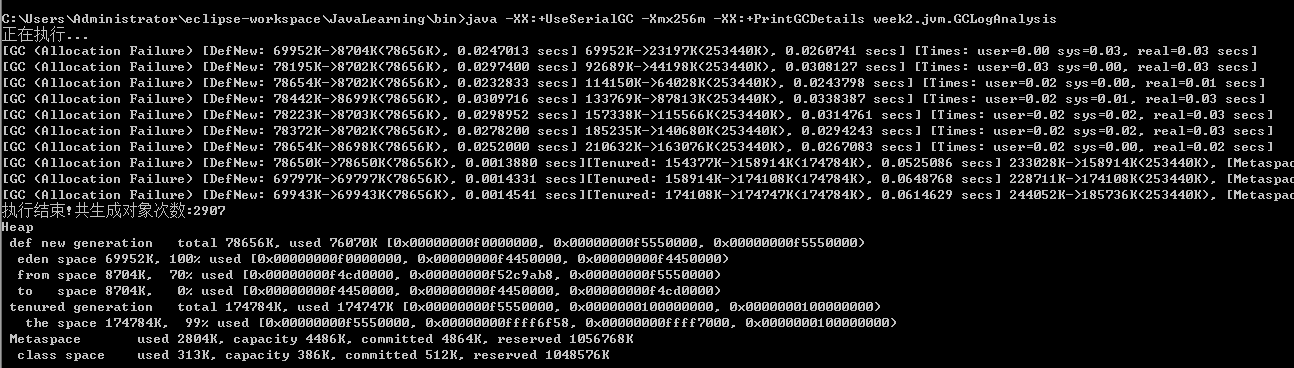




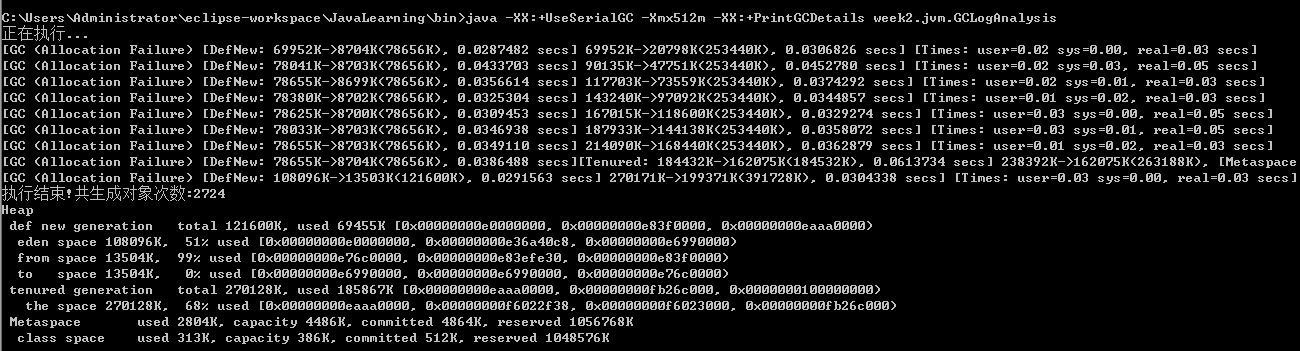
**SerialGC:**

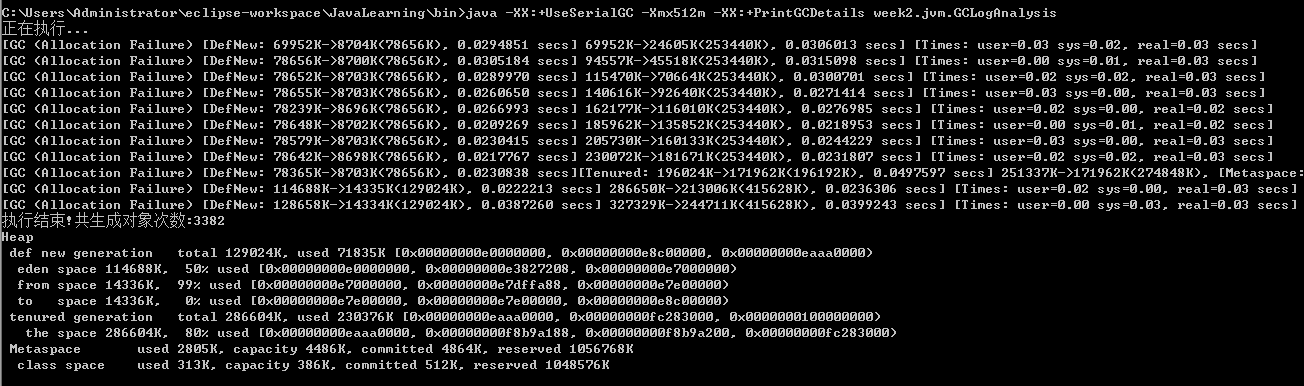
**256M**



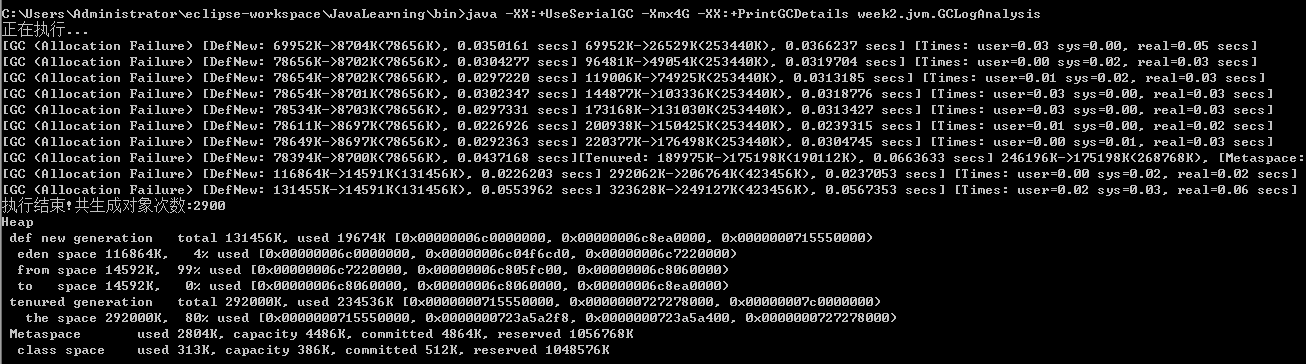


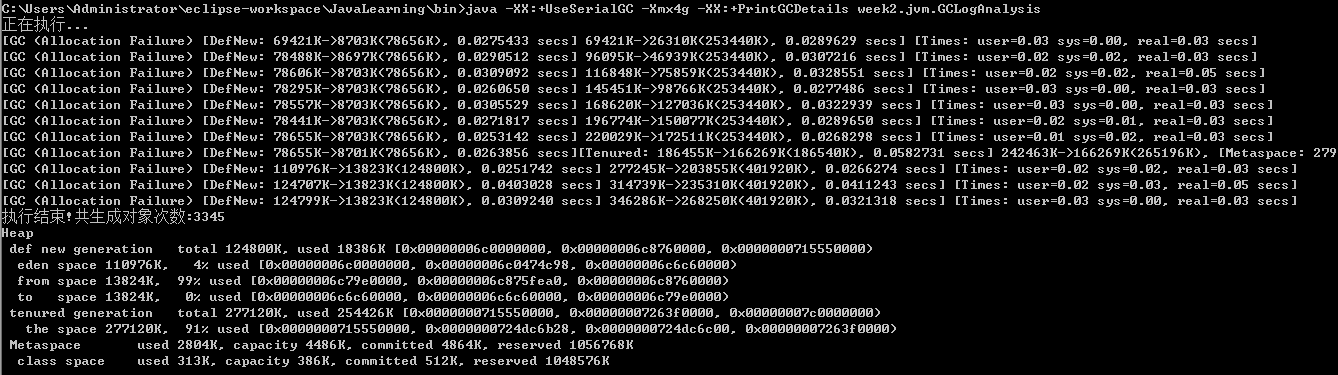
**512M**





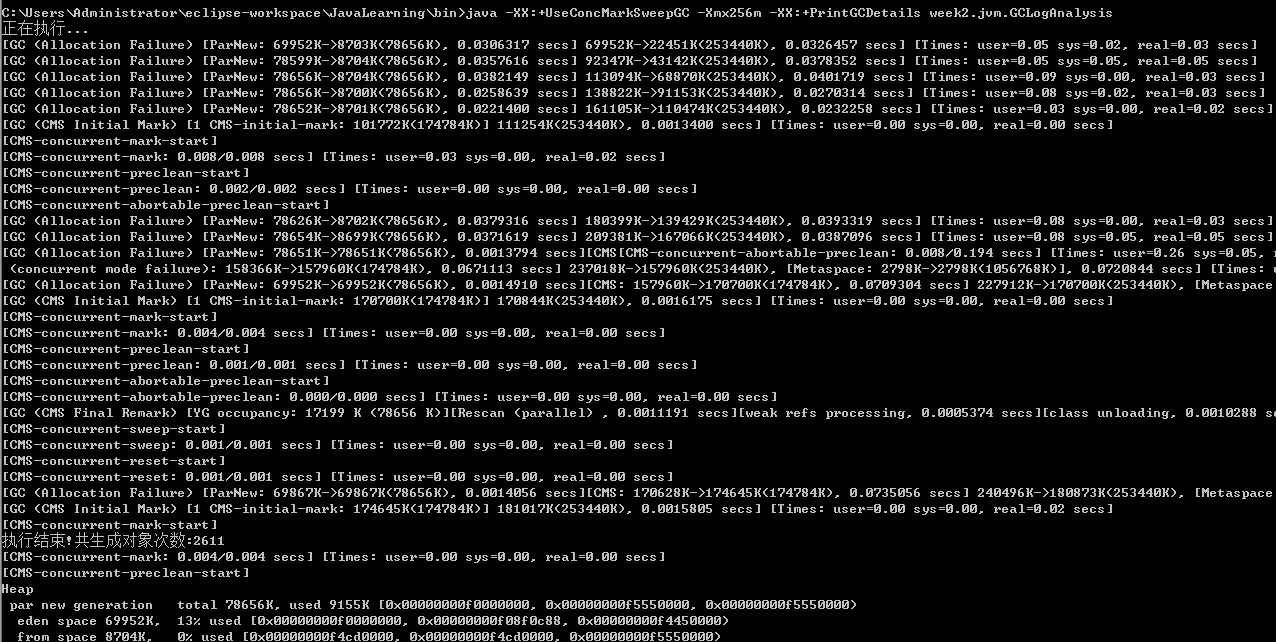
**4G**

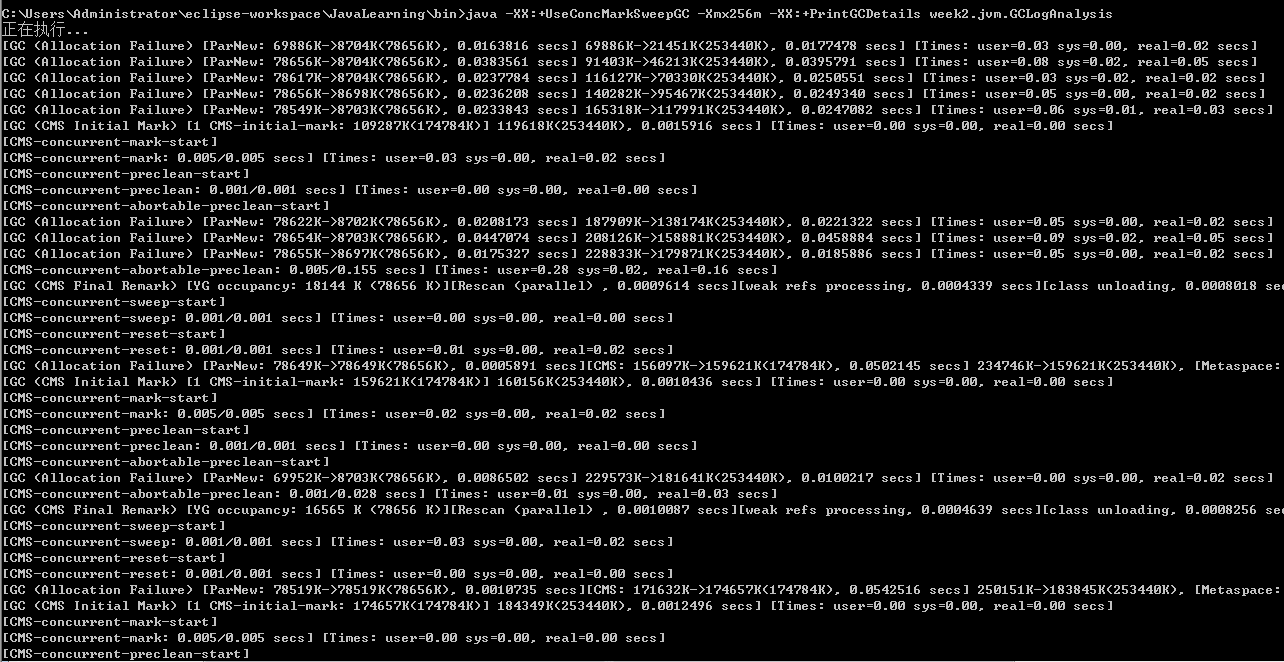


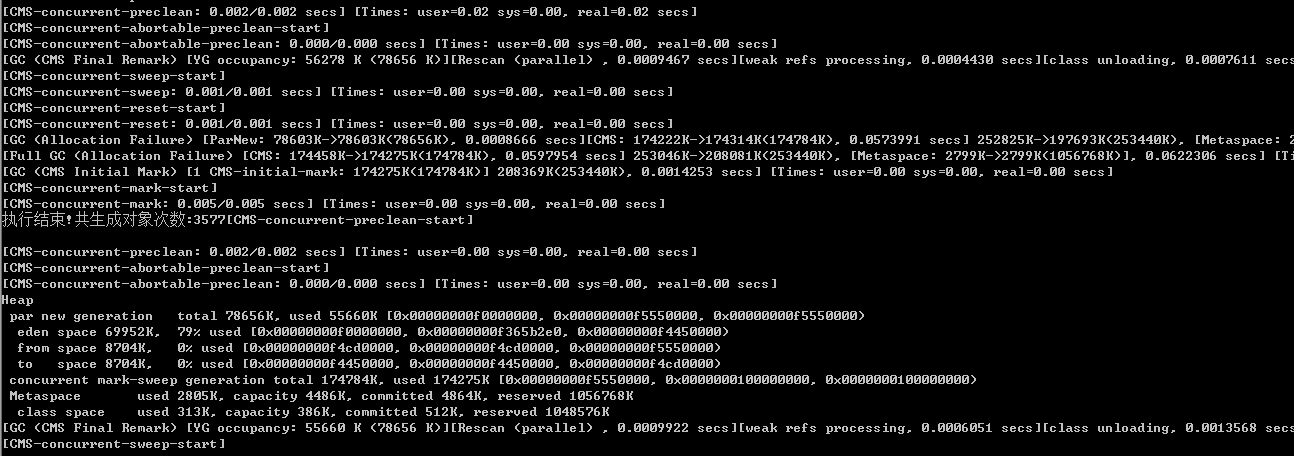


**CMS GC**

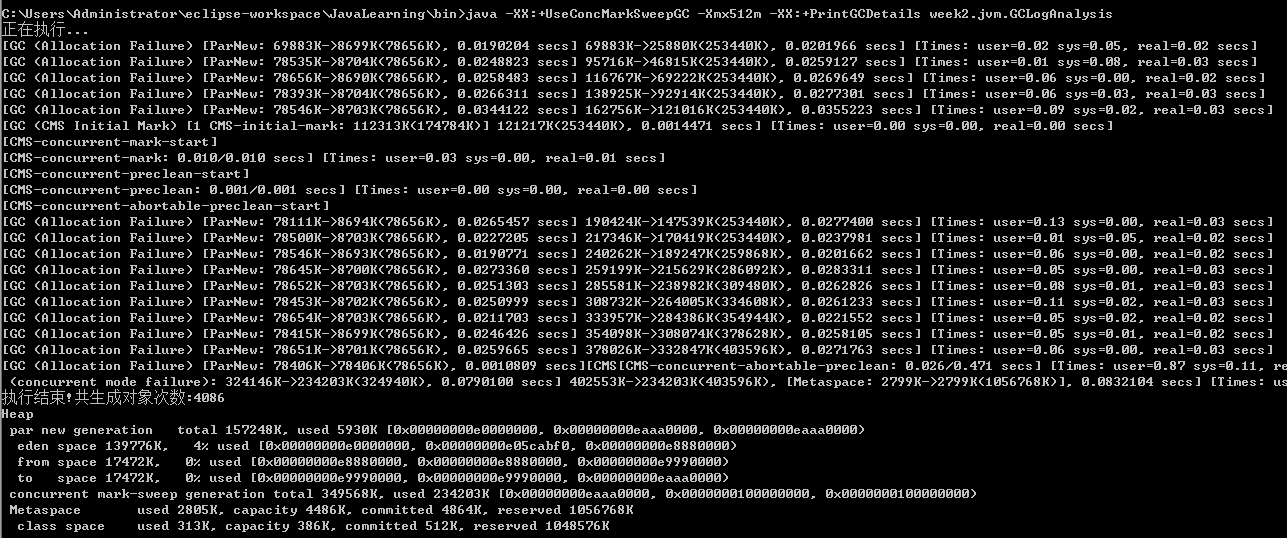
**256M**

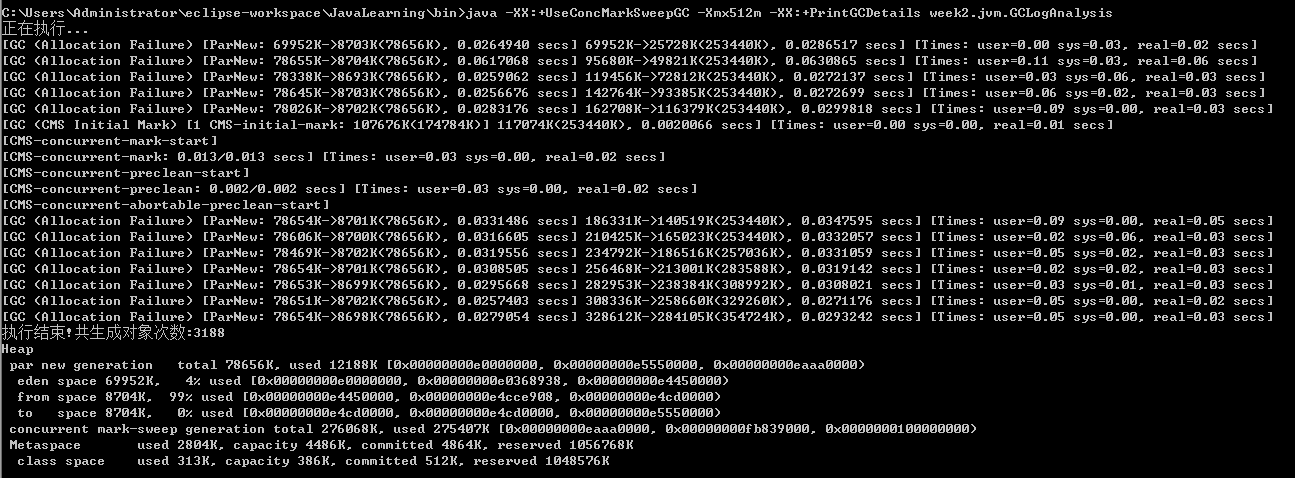




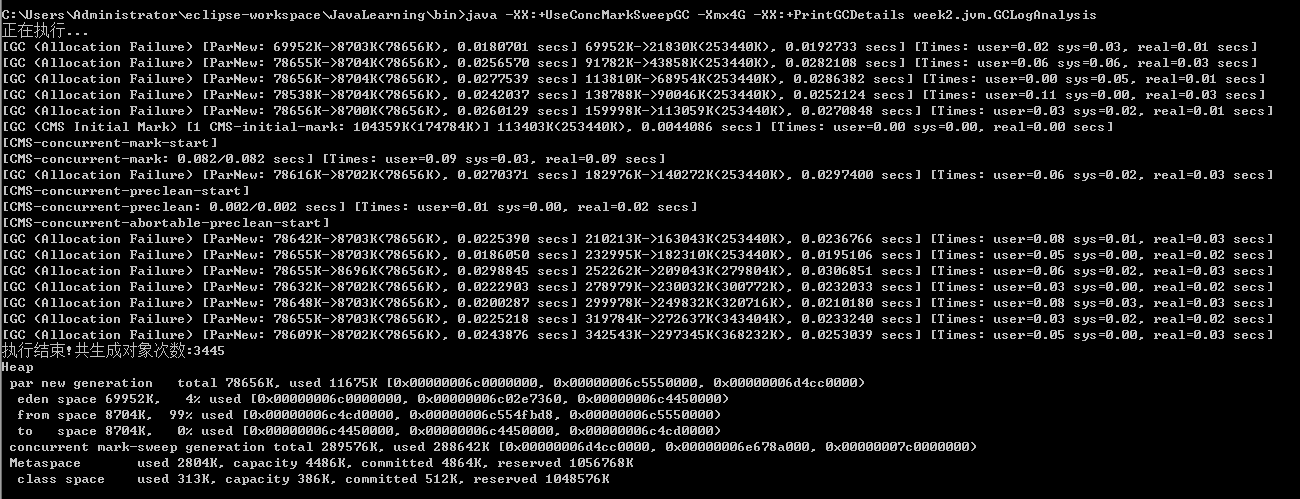


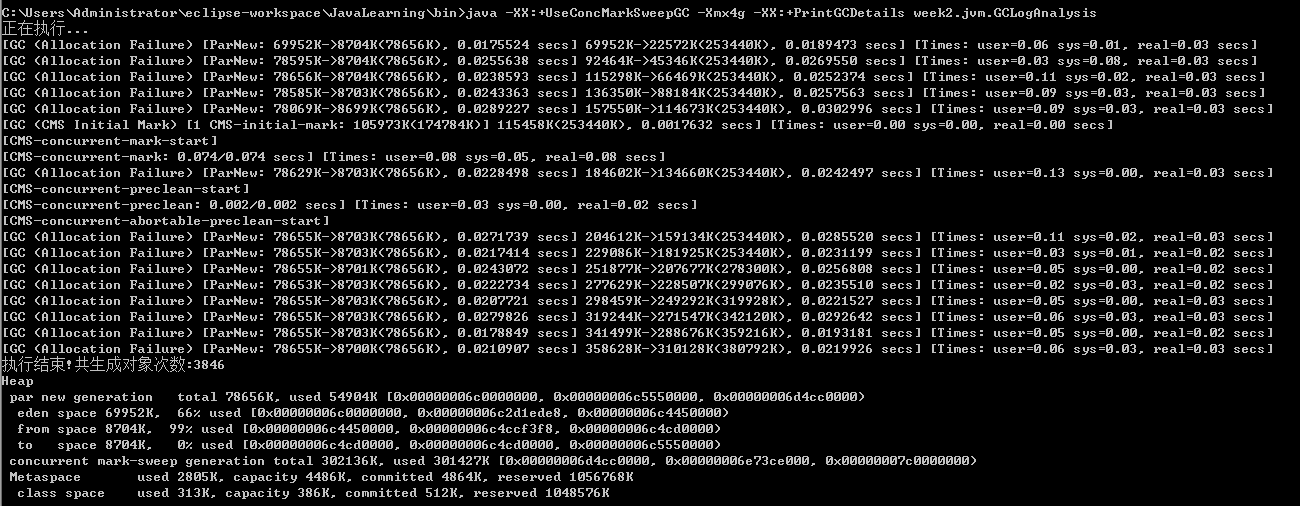
**512M**





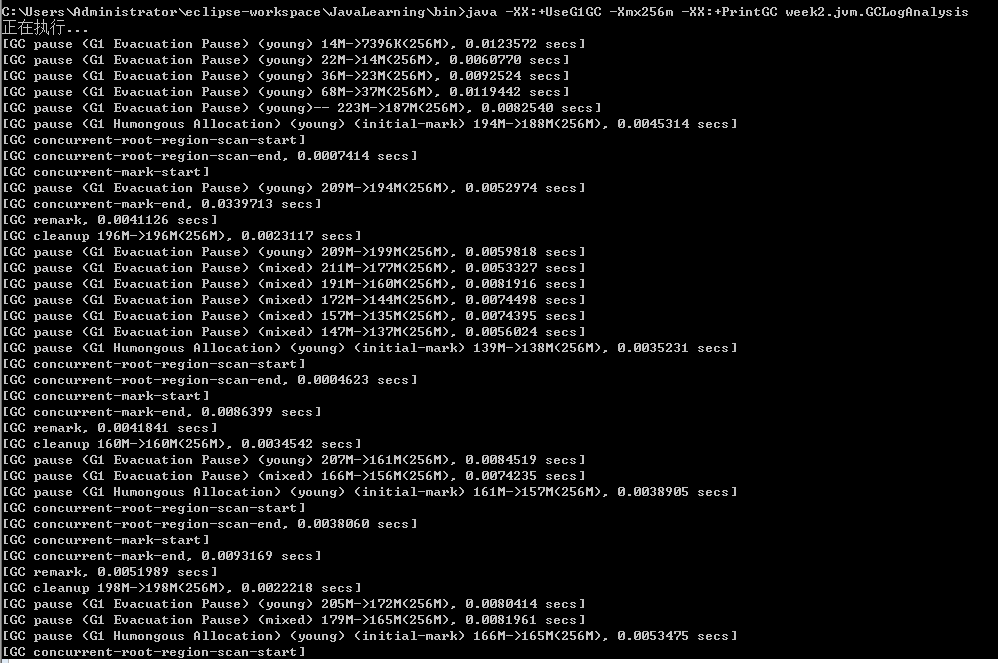
**4G**

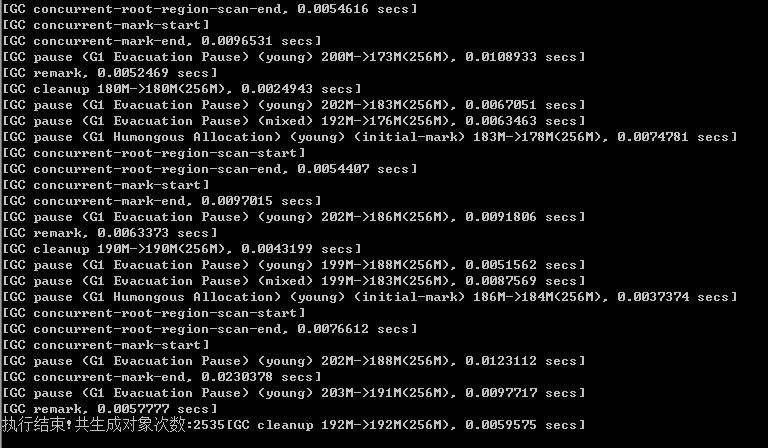


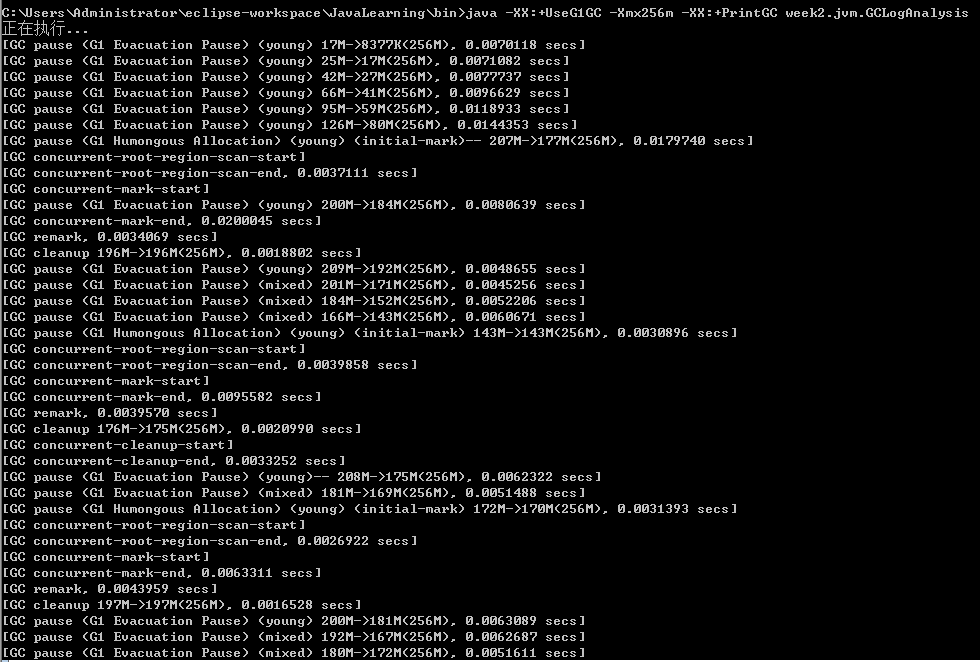


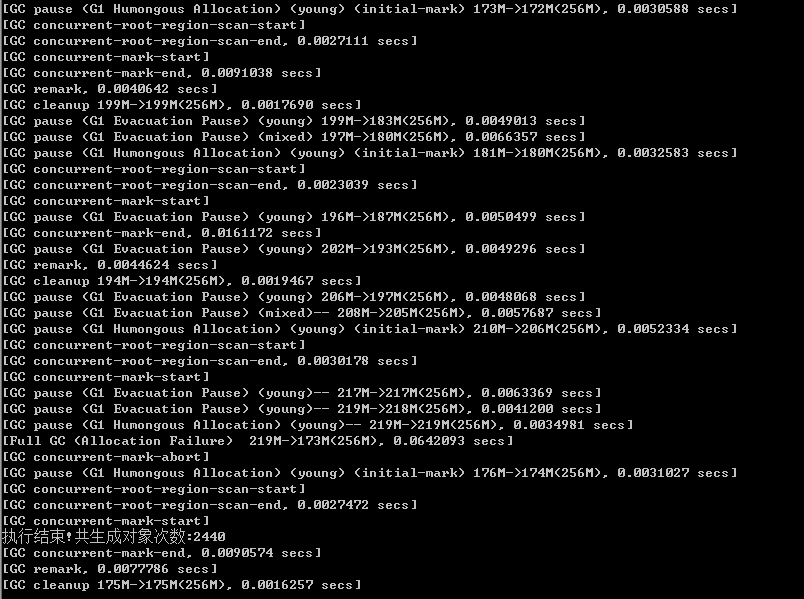
**G1 GC**

**256M**

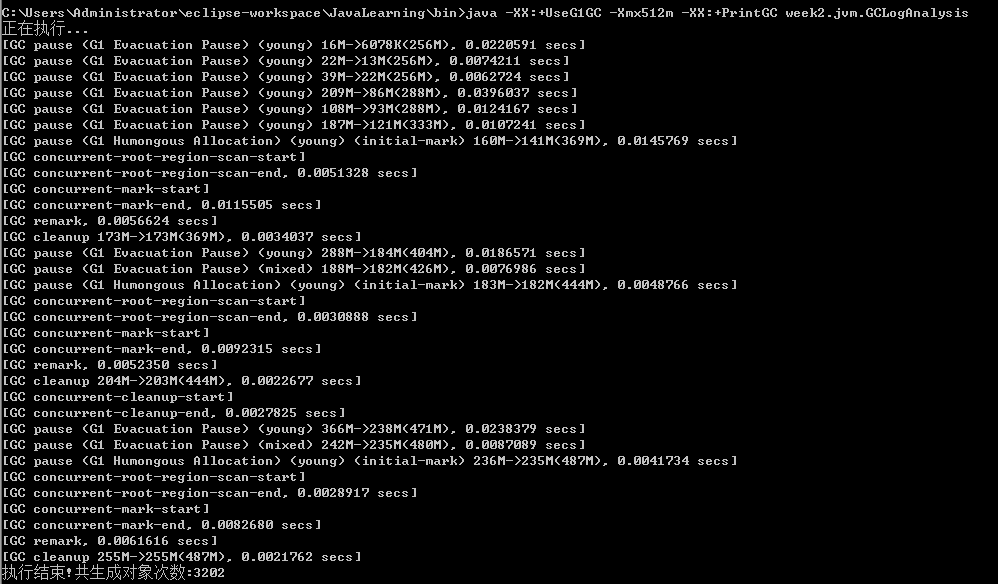


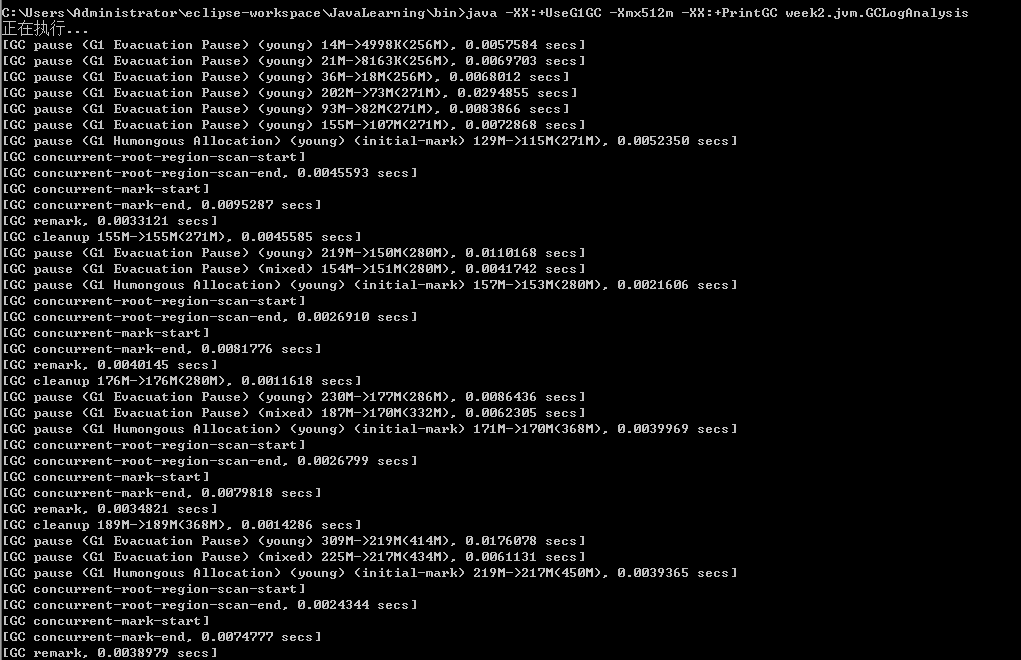






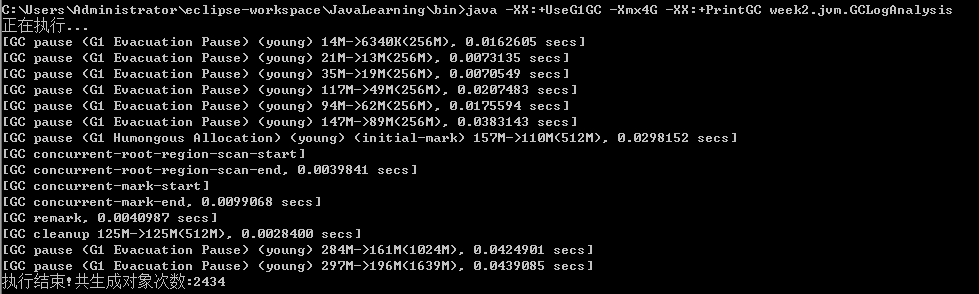
**512M**

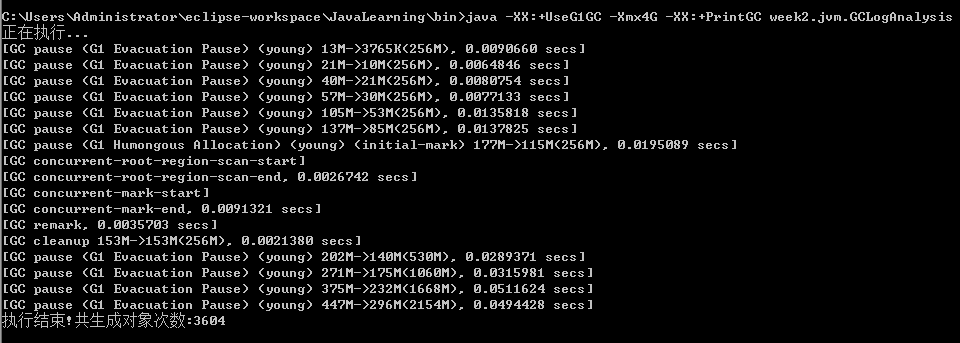






**4G**





**四种GC方式在256M, 512M, 4G max heap 下的表现情况如下**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **GC类型** | **最大堆内存分配** | **新建对象数** | **Young GC数** | **Full GC数** | **有无内存溢出**  **或一直GC情况** |
| 并行 GC | 256M | 2472;2092 | 10;10 | 4;2 | Y |
| 串行 GC | 256M | 2937;2907 | 9;10 | 2;0 | N |
| CMS GC | 256M | 2611;3577 | 9;11 | 2;3 | N |
| G1 GC | 256M | 2535;2440 |  |  | N |
| 并行 GC | 512M | 2716;3339 | 6;8 | 1;2 | N |
| 串行 GC | 512M | 2724;3382 | 9;11 | 0;0 | N |
| CMS GC | 512M | 4086;3188 | 15;12 | 1;1 | N |
| G1 GC | 512M | 3202;3199 |  |  | N |
| 并行 GC | 4G | 3499;4127 | 6;6 | 1;2 | N |
| 串行 GC | 4G | 2900;3345 | 10;11 | 0;0 | N |
| CMS GC | 4G | 3445;3846 | 12;13 | 1;1 | N |
| G1 GC | 4G | 2434;3604 |  |  | N |

**总结（机器 4核， 16GB）**

串行GC: 由于只有单线程进行垃圾回收，随着堆内存的增大，新建对象数小幅度增加， Young GC次数减少, Full GC次数也减少

并行GC：多线程，所有线程参加垃圾回收，随着堆内存的增大，对象增加个数明显，Young GC次数减少, Full GC次数也减少

CMS GC：多线程，部分线程参加垃圾回收，随着堆内存的增大，对象增加个数比较明显，Young GC次数先增后减, Full GC次数减少，但是和并行GC生成的对象个数差不太多； 和并行GC相比Young GC 次数显著增加

G1 GC： 生成的对象个数略好于串行GC，但和预期的不一样，特别是堆内存设置为4G时，生成对象数小于512M的堆内存时，这一点和预期相差的比较大。也是有所不理解的事情 ，可能是因为自己CPU计算能力的问题

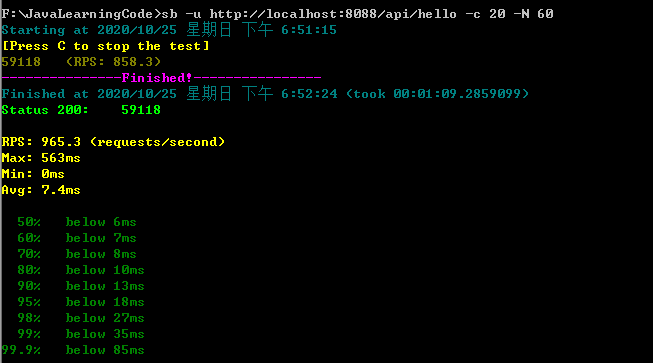
**2、使用压测工具（wrk或sb），演练gateway-server-0.0.1-SNAPSHOT.jar 示例。**

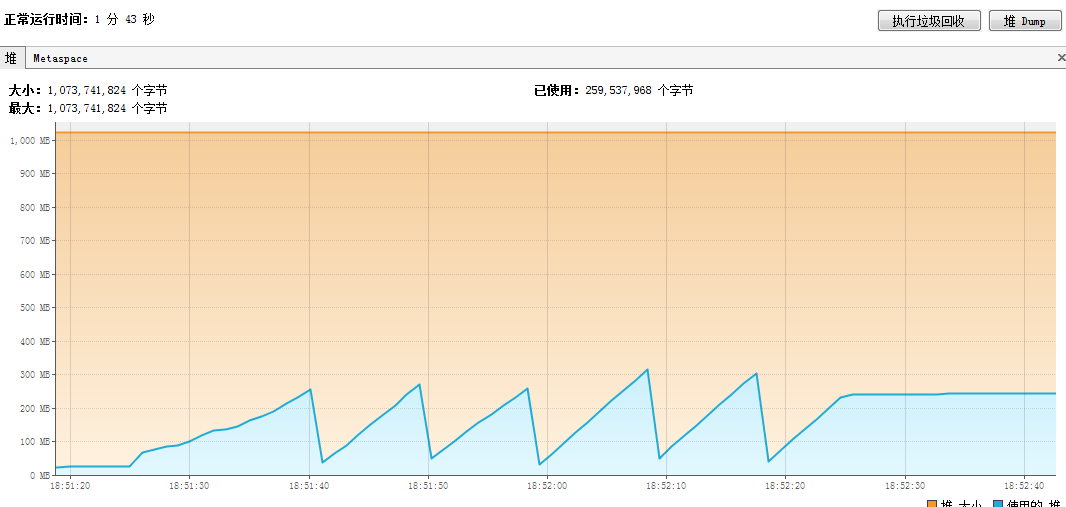
**并行GC启动， 1G堆**



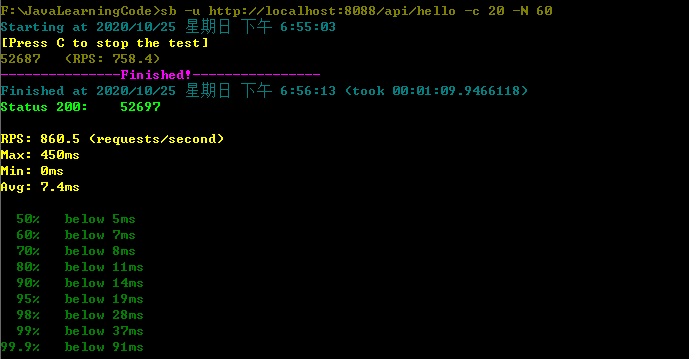
**压测**

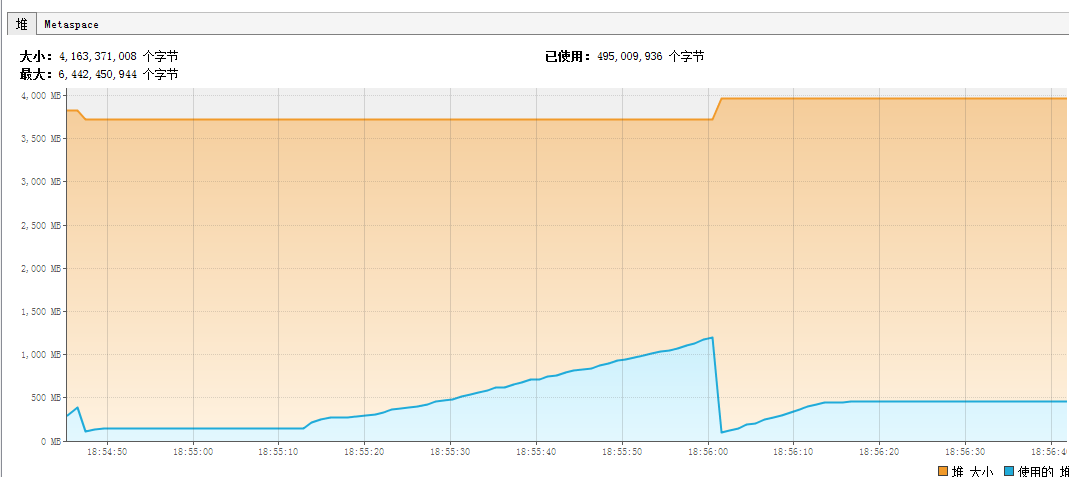
sb -u <http://localhost:8088/api/hello> -c 20 -N 60



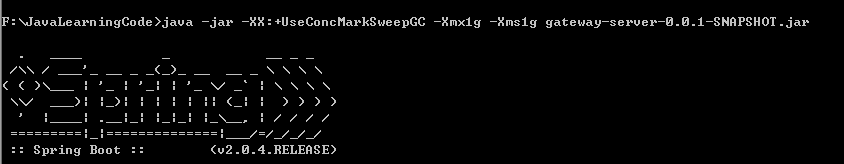


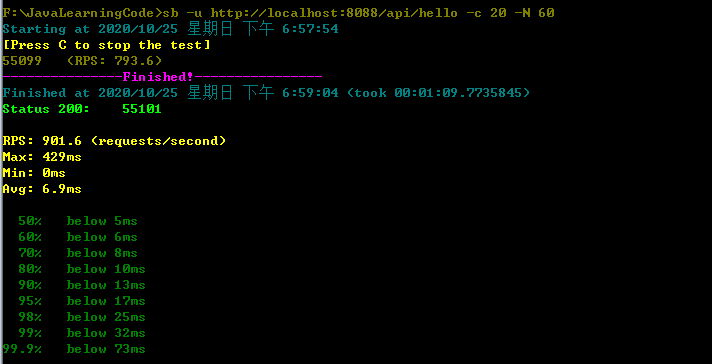
并行6G堆

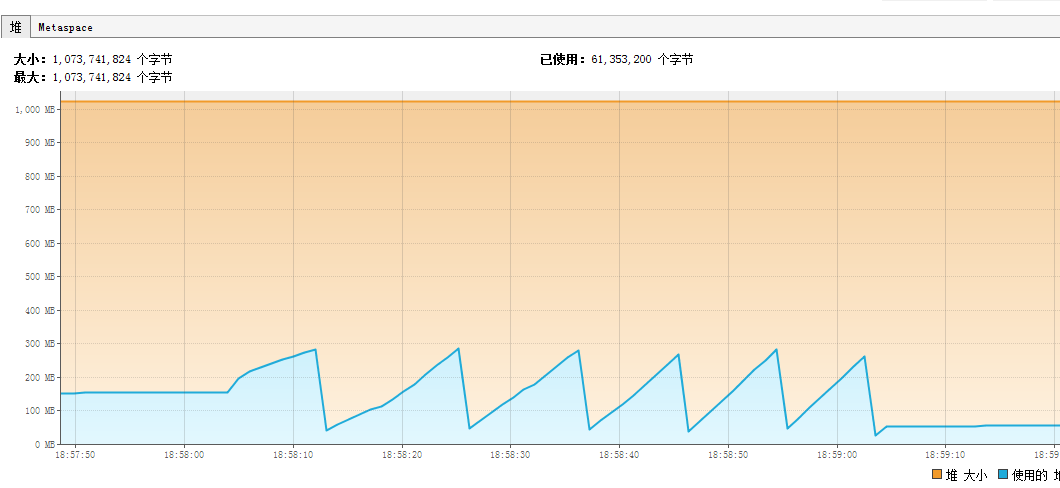




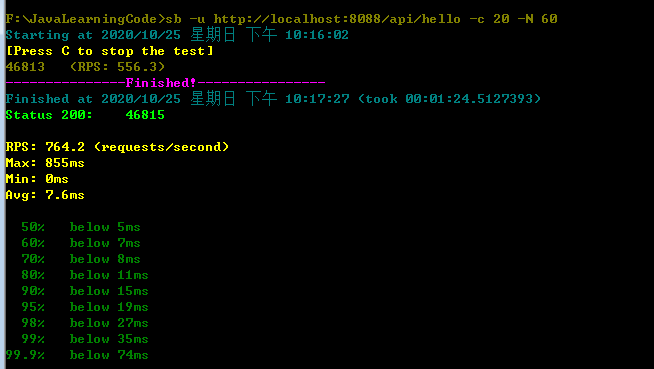
CMS 1G堆

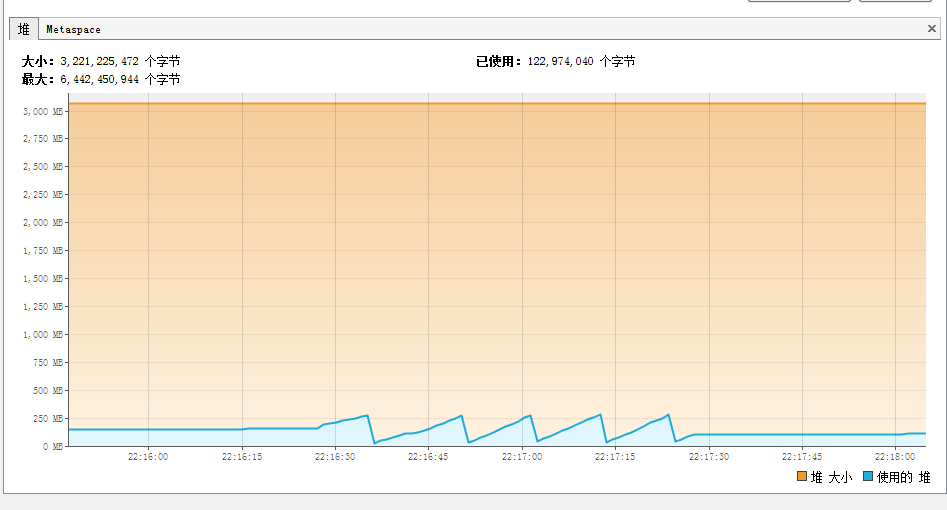




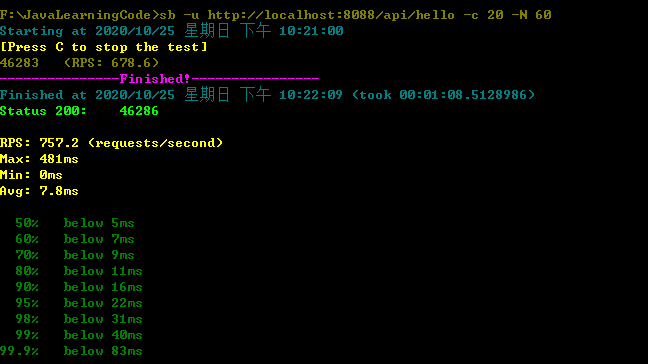


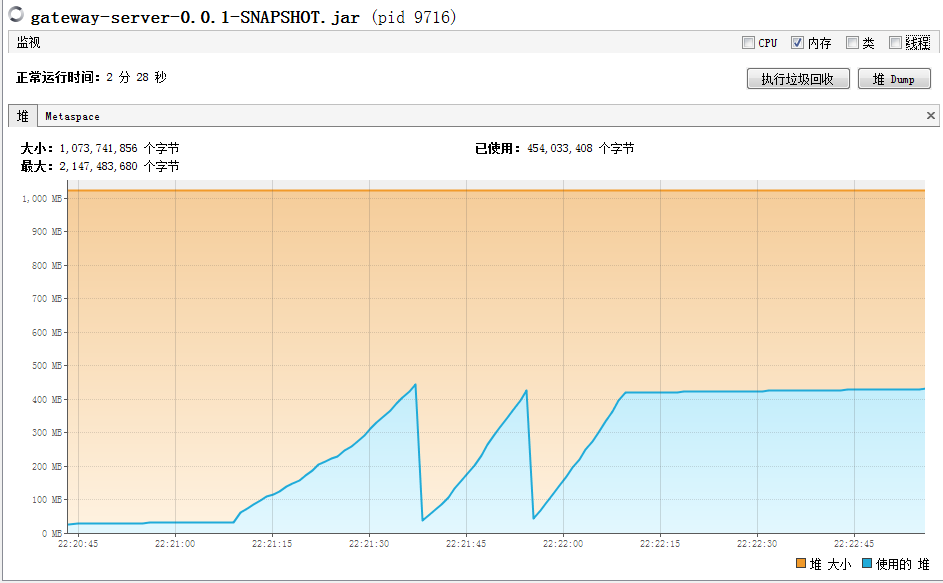
CMS 6G堆



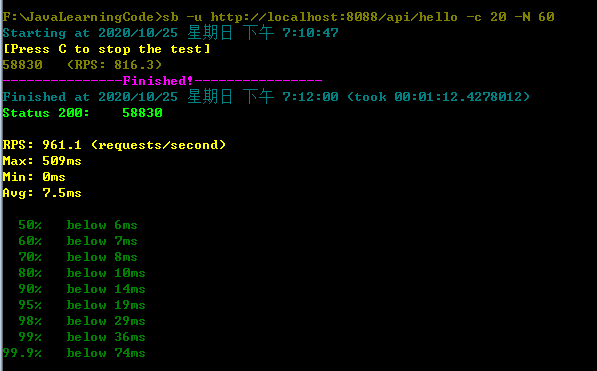


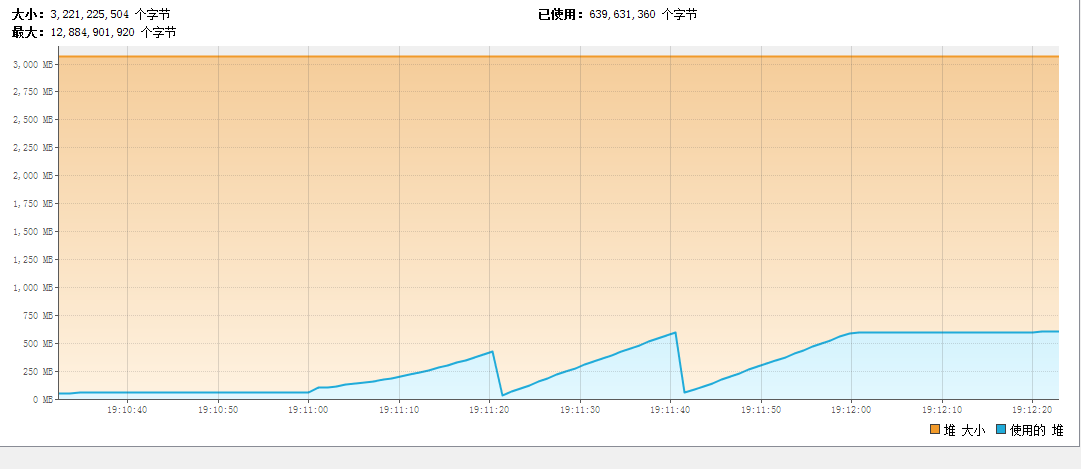
G1 1G堆





G1 6G堆





**结果统计**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **GC类型** | **最大堆**  **内存分配** | **RPS** | **最大响应时间(ms)** | **实际最大堆内存使用约(M)** | **Full GC数** |
| 并行GC | 1G | 965 | 563 | 260 | 5 |
| 并行GC | 6G | 860 | 450 | 1200 | 1 |
| CMS GC | 1G | 901 | 429 | 290 | 6 |
| CMS GC | 6G | 764 | 855 | 260 | 5 |
| G1 GC | 1G | 757 | 481 | 450 | 2 |
| G1 GC | 6G | 961 | 509 | 650 | 2 |

**结果总结：** 可能由于CPU（测试时100%使用率）太弱的问题，内存的增大并没有显著的提高RPS,有的还有所下降，并行GC的次数显著减少， 最大内存使用数也有所提升。 而CMS和G1的内存使用比较平稳，说明没有很大的GC发生. G1的GC时的内存量有上升趋势。

**3、(选做)如果自己本地有可以运行的项目，可以按照2的方式进行演练。**

**根据上述自己对于1和2的演示，写一段对于不同 GC 的总结，提交到 Github。**

**第二次课作业**

**1、（可选）运行课上的例子，以及 Netty 的例子，分析相关现象。**

**2、写一段代码，使用 HttpClient 或 OkHttp 访问 http://localhost:8801，代码提交到Github。**

**package** io.github.czh.http.client;

**import** java.io.IOException;

**import** org.apache.http.HttpEntity;

**import** org.apache.http.ParseException;

**import** org.apache.http.client.ClientProtocolException;

**import** org.apache.http.client.config.RequestConfig;

**import** org.apache.http.client.methods.CloseableHttpResponse;

**import** org.apache.http.client.methods.HttpGet;

**import** org.apache.http.impl.client.CloseableHttpClient;

**import** org.apache.http.impl.client.HttpClientBuilder;

**public** **class** HttpClient {

**public** **static** **void** main(String[] args) {

CloseableHttpClient httpClient = HttpClientBuilder.*create*().build();

HttpGet httpGet = **new** HttpGet("http://127.0.0.1:8808");

// 响应模型

CloseableHttpResponse response = **null**;

**try** {

// 配置信息

RequestConfig requestConfig = RequestConfig.*custom*()

// 设置连接超时时间(单位毫秒)

.setConnectTimeout(5000)

// 设置请求超时时间(单位毫秒)

.setConnectionRequestTimeout(5000)

// socket读写超时时间(单位毫秒)

.setSocketTimeout(5000)

// 设置是否允许重定向(默认为true)

.setRedirectsEnabled(**true**).build();

// 将上面的配置信息 运用到这个Get请求里

httpGet.setConfig(requestConfig);

// 由客户端执行(发送)Get请求

response = httpClient.execute(httpGet);

// 从响应模型中获取响应实体

HttpEntity responseEntity = response.getEntity();

System.***out***.println("响应状态为:" + response.getStatusLine());

**if** (responseEntity != **null**) {

System.***out***.println("响应内容长度为:" + responseEntity.getContentLength());

}

} **catch** (ClientProtocolException e) {

e.printStackTrace();

} **catch** (ParseException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

// 释放资源

**if** (httpClient != **null**) {

httpClient.close();

}

**if** (response != **null**) {

response.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

运行结果总是报connection refused, 还没找到具体原因

