

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 5实验报告**

|  |  |
| --- | --- |
| 姓名 | 夏铭远 |
| 学号 | 1173710209 |
| 班号 | 1737102 |
| 电子邮件 | 462272701@qq.com |
| 手机号码 | 13029878656 |

**目录**

[1 实验目标概述 1](#_Toc3923017)

[2 实验环境配置 1](#_Toc3923018)

[3 实验过程 1](#_Toc3923019)

[3.1 Static Program Analysis 1](#_Toc3923020)

[3.1.1 人工代码走查（walkthrough） 1](#_Toc3923021)

[3.1.2 使用CheckStyle和SpotBugs进行静态代码分析 1](#_Toc3923022)

[3.2 Java I/O Optimization 1](#_Toc3923023)

[3.2.1 多种I/O实现方式 1](#_Toc3923024)

[3.2.2 多种I/O实现方式的效率对比分析 2](#_Toc3923025)

[3.3 Java Memory Management and Garbage Collection (GC) 3](#_Toc3923026)

[3.3.1 使用-verbose:gc参数 3](#_Toc3923027)

[3.3.2 用jstat命令行工具的-gc和-gcutil参数 3](#_Toc3923028)

[3.3.3 使用jmap -heap命令行工具 3](#_Toc3923029)

[3.3.4 使用jmap -clstats命令行工具 3](#_Toc3923030)

[3.3.5 使用jmap -permstat命令行工具 3](#_Toc3923031)

[3.3.6 使用JMC/JFR、jconsole或VisualVM工具 3](#_Toc3923032)

[3.3.7 分析垃圾回收过程 3](#_Toc3923033)

[3.3.8 配置JVM参数并发现优化的参数配置 3](#_Toc3923034)

[3.4 Dynamic Program Profiling 3](#_Toc3923035)

[3.4.1 使用JMC或VisualVM进行CPU Profiling 3](#_Toc3923036)

[3.4.2 使用VisualVM进行Memory profiling 3](#_Toc3923037)

[3.5 Memory Dump Analysis and Performance Optimization 3](#_Toc3923038)

[3.5.1 内存导出 3](#_Toc3923039)

[3.5.2 使用MAT分析内存导出文件 3](#_Toc3923040)

[3.5.3 发现热点/瓶颈并改进、改进前后的性能对比分析 3](#_Toc3923041)

[3.5.4 在MAT内使用OQL查询内存导出 4](#_Toc3923042)

[3.5.5 观察jstack/jcmd导出程序运行时的调用栈 4](#_Toc3923043)

[3.5.6 使用设计模式进行代码性能优化 4](#_Toc3923044)

[4 实验进度记录 4](#_Toc3923045)

[5 实验过程中遇到的困难与解决途径 4](#_Toc3923046)

[6 实验过程中收获的经验、教训、感想 5](#_Toc3923047)

[6.1 实验过程中收获的经验和教训 5](#_Toc3923048)

[6.2 针对以下方面的感受 5](#_Toc3923049)

# 实验目标概述

根据实验手册简要撰写。

# 实验环境配置

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

在这里给出你的GitHub Lab5仓库的URL地址（Lab5-学号）。

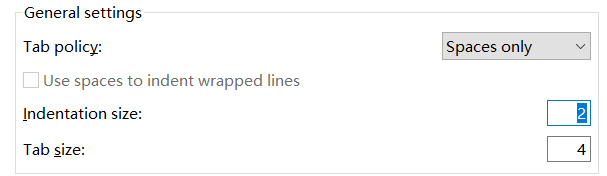
# 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## Static Program Analysis

### 人工代码走查（walkthrough）

1. 每一行的缩进由tab改为空格



在Eclipse中修改设置之后ctrl+shift+f即可

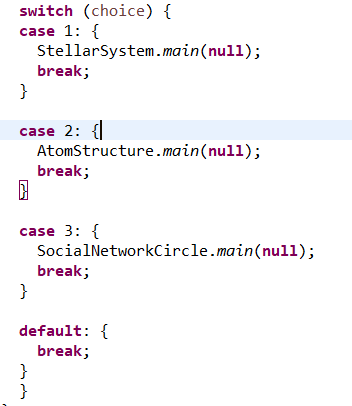
1. 命名规则有误

变量名改为驼峰式

 →

1. Switch块未定义default

加上default即可

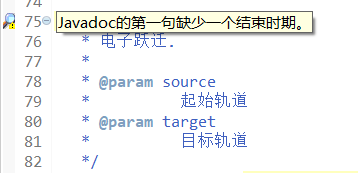


### 使用CheckStyle和SpotBugs进行静态代码分析

列出你所发现的问题和所做的修改。每种类型的问题只需列出一个示例即可。

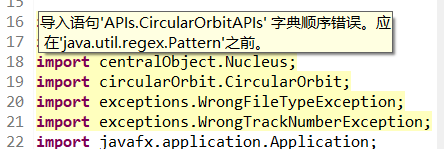
对比分析两种工具发现问题的能力和发现问题的类型上有何差异。

1. javadoc有误



第一行加上“.”即可

1. 导入package顺序错误



按提示修改即可

Findbugs主要检查bytecode中的bug patterns

Checkstyle则检查是否符合代码规范

## Java I/O Optimization

### 多种I/O实现方式

实现了哪些I/O方式来读写文件，具体如何实现的。

如何用strategy设计模式实现在多种I/O策略之间的切换。

### 多种I/O实现方式的效率对比分析

如何收集你的程序I/O语法文件的时间。



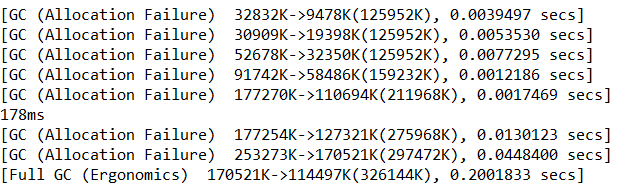
分别在读之前和之后记录系统时间，然后做差。

表格方式对比不同I/O的性能。

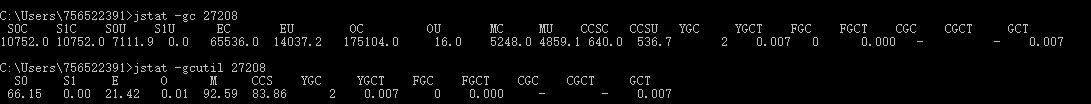
|  |  |  |  |
| --- | --- | --- | --- |
|  |  | StellarSystem.txt | SocialNetwork.txt |
| BufferedReader/BufferedWriter | 读文件 | 629ms | 342ms |
| 写文件 | 650ms | 400ms |
| FileReader/FileWriter | 读文件 | 548ms | 1387ms |
| 写文件 | 500ms | 1500ms |
| InputStream/OutputStream | 读文件 | 138ms | 162ms |
| 写文件 | 140ms | 170ms |

## Java Memory Management and Garbage Collection (GC)

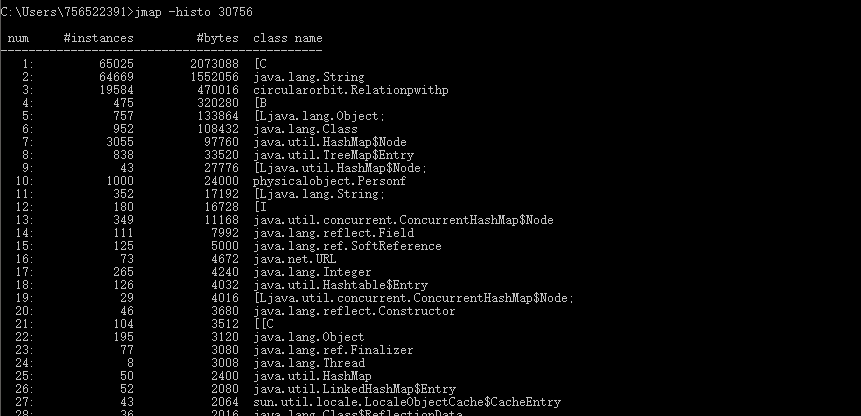
### 使用-verbose:gc参数

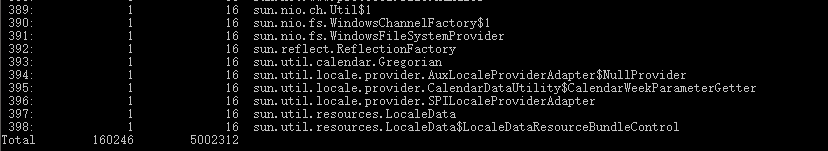
先增大后减小

### 用jstat命令行工具的-gc和-gcutil参数

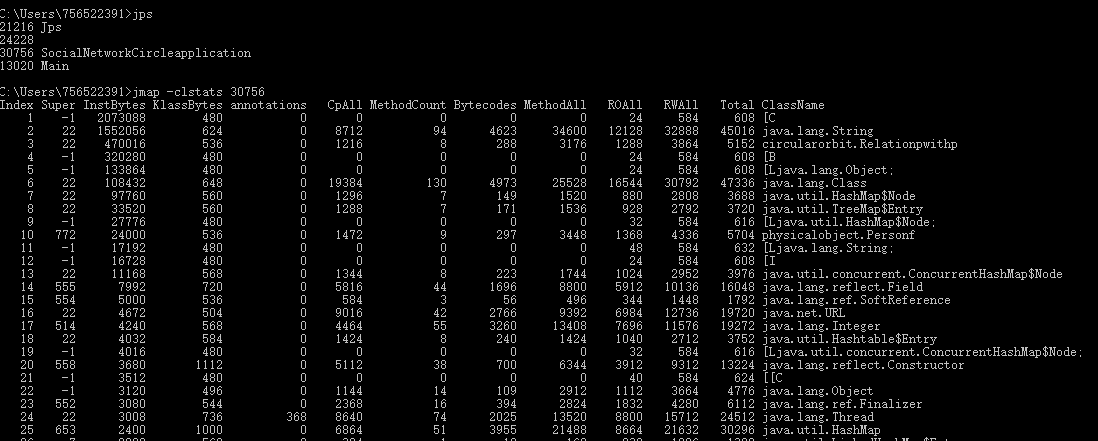


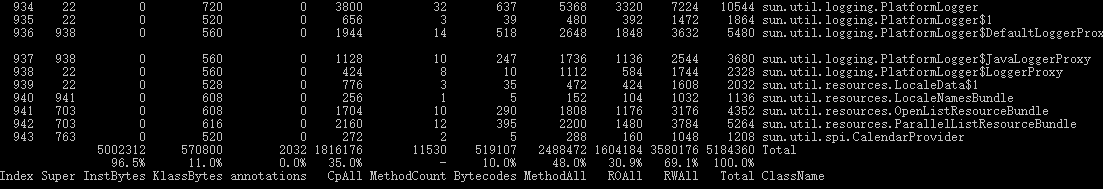
### 使用jmap -heap命令行工具



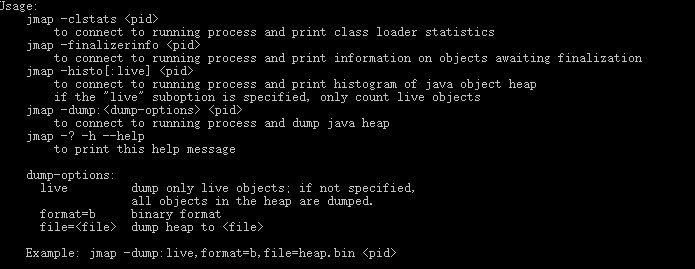


### 使用jmap -clstats命令行工具





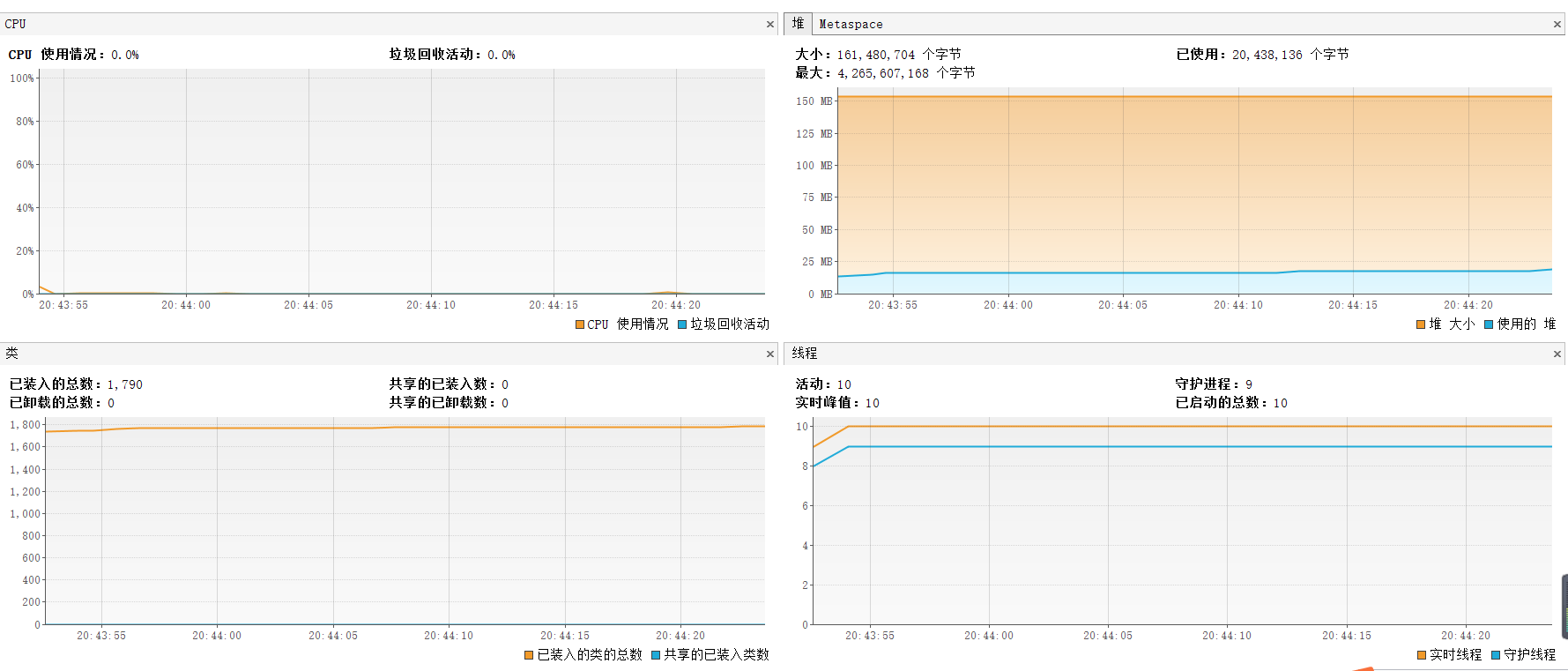
### 使用jmap -permstat命令行工具

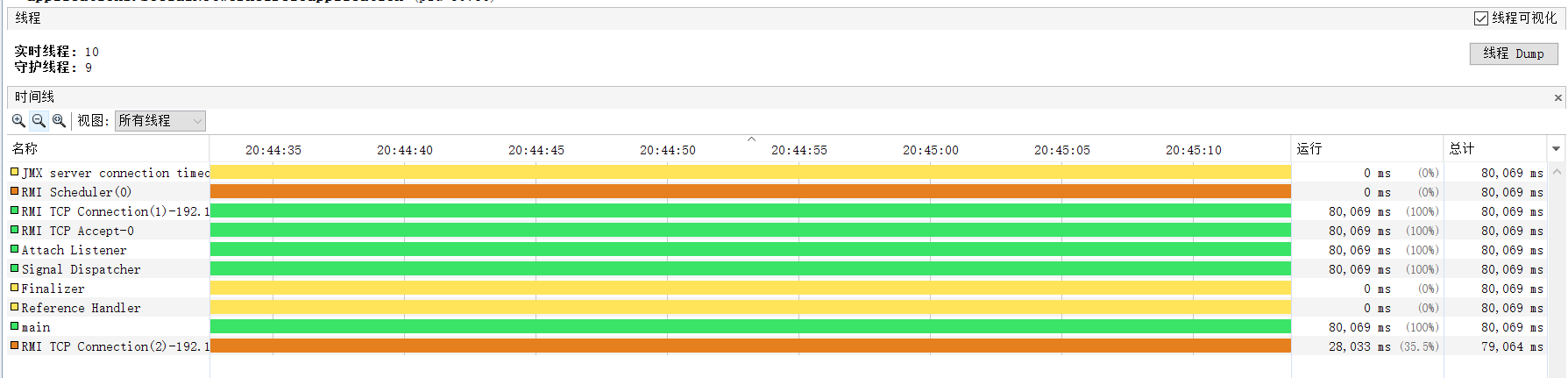


没有

### 使用JMC/JFR、jconsole或VisualVM工具

VisualVM





### 分析垃圾回收过程

1、所有new出来的对象都会最先分配到新生代区域中，两个survivor（S0和S1）区域初始化是为空的；

2、当伊甸园区域满了之后，就会引发一次minor garbage collection（小型垃圾回收）；

3、当在minor garbage collection（小型垃圾回收），存活下来的对象就会被移到S0区域；

4、当伊甸园区域再次填满时，又会发生下一次垃圾回收，存活下来的对象会被移到survivor区域，而未存活的对象则被直接删除，但是，不同的是，在这次垃圾回收中，存活对象和之前在S0区域中的对象都会移到S1区域中。一旦所有对象都被移到S1区域中，那么S0中的对象就会被清除；

5、下一次的垃圾回收的时候，又会重复上次的步骤，清除需要回收的对象，并且切换一次survivor区域，所有存活的对象都被移到S0，伊甸园区域和S1区域被清除；

6、重复以上步骤，并记录对象的年龄（记录垃圾回收的次数），当有对象的年龄达到一定的阈值时，就将新生代中的对象移到到老年代。比如下图，这个阈值为8；

7、接下来垃圾回收就会重复以上步骤，不断的进行对象的清除和年代的移动；

8、观察上述步骤发现，大部分的垃圾回收过程都是在新生代进行的，直到老年代的内存不够用了才会发起一次minor GC ，会进行标记和整理压缩。

### 配置JVM参数并发现优化的参数配置



1. 堆大小设置  
   JVM 中最大堆大小有三方面限制：相关操作系统的数据模型（32-bt还是64-bit）限制；系统的可用虚拟内存限制；系统的可用物理内存限制。

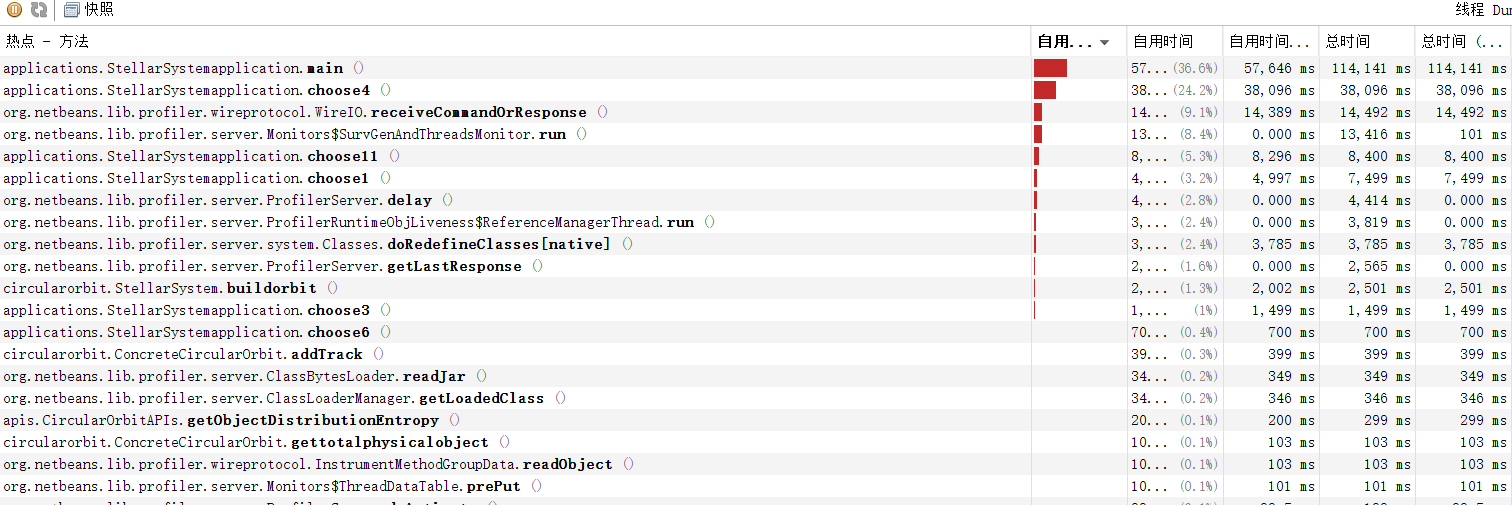
java -Xmx512m -Xms512m -Xss128k -XX:NewRatio=4 -XX:SurvivorRatio=4 -XX:MaxMetaspaceSize=16m  
-XX:SurvivorRatio=8：设置年轻代中Eden区与Survivor区的大小比值。设置为8，则两个Survivor区与一个Eden区的比值为2:8，一个Survivor区占整个年轻代的1/10  
-XX: MaxMetaspaceSize=16m:设置持久代大小为16m。

1. 回收器选择  
   JVM给了三种选择：串行收集器、并行收集器、并发收集器，但是串行收集器只适用于小数据量的情况，所以这里的选择主要针对并行收集器和并发收集器。默认情况下，JDK5.0以前都是使用串行收集器，如果想使用其他收集器需要在启动时加入相应参数。JDK5.0以后，JVM会根据当前[系统配置](http://java.sun.com/j2se/1.5.0/docs/guide/vm/server-class.html" \t "_blank)进行判断。

java -Xmx512m -Xms512m -Xss128k -XX:NewRatio=4 -XX:SurvivorRatio=4 -XX:+UseConMarkSweepGC;

## Dynamic Program Profiling

### 使用JMC或VisualVM进行CPU Profiling



方法时间运行较为准确，合理

### 使用VisualVM进行Memory profiling



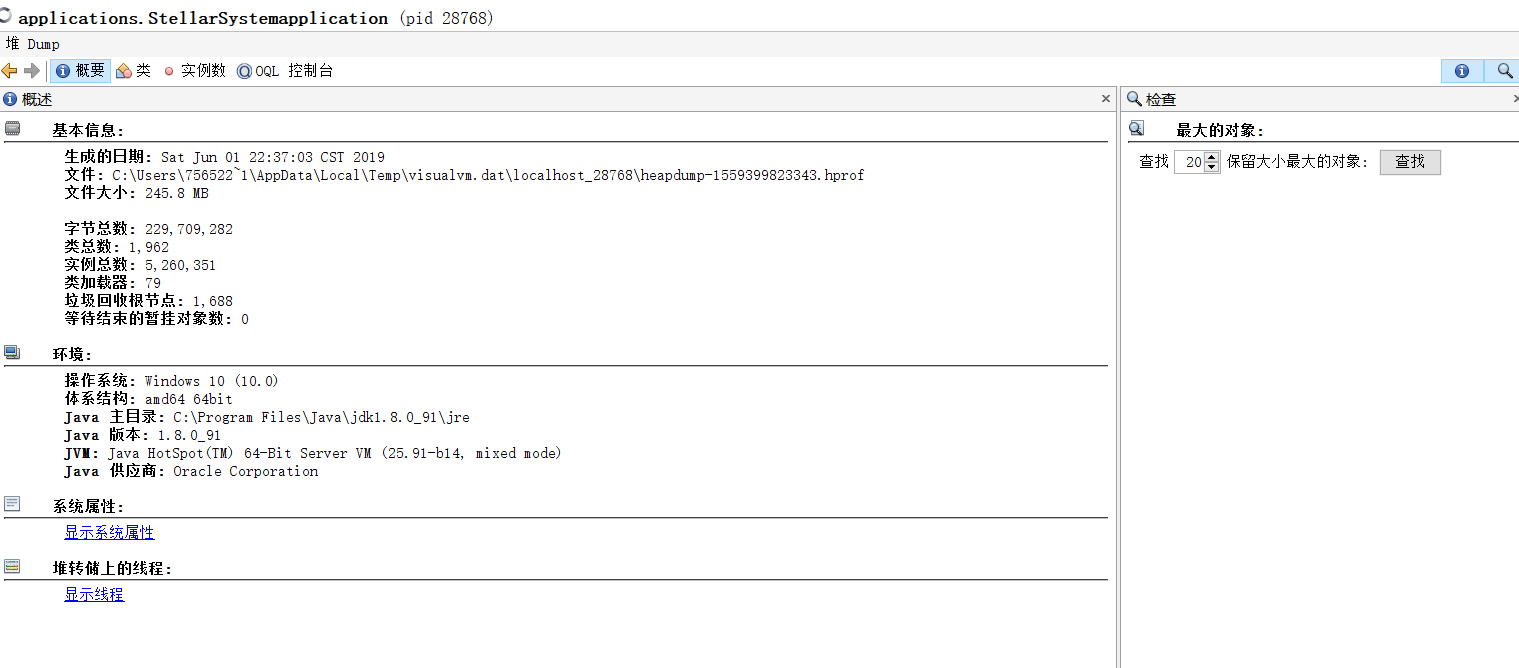
Pattern与matcher对象个数不合理

程序设计不好，不应该每次检查时都创建一个pattern对象

其他的对象数都比较合理

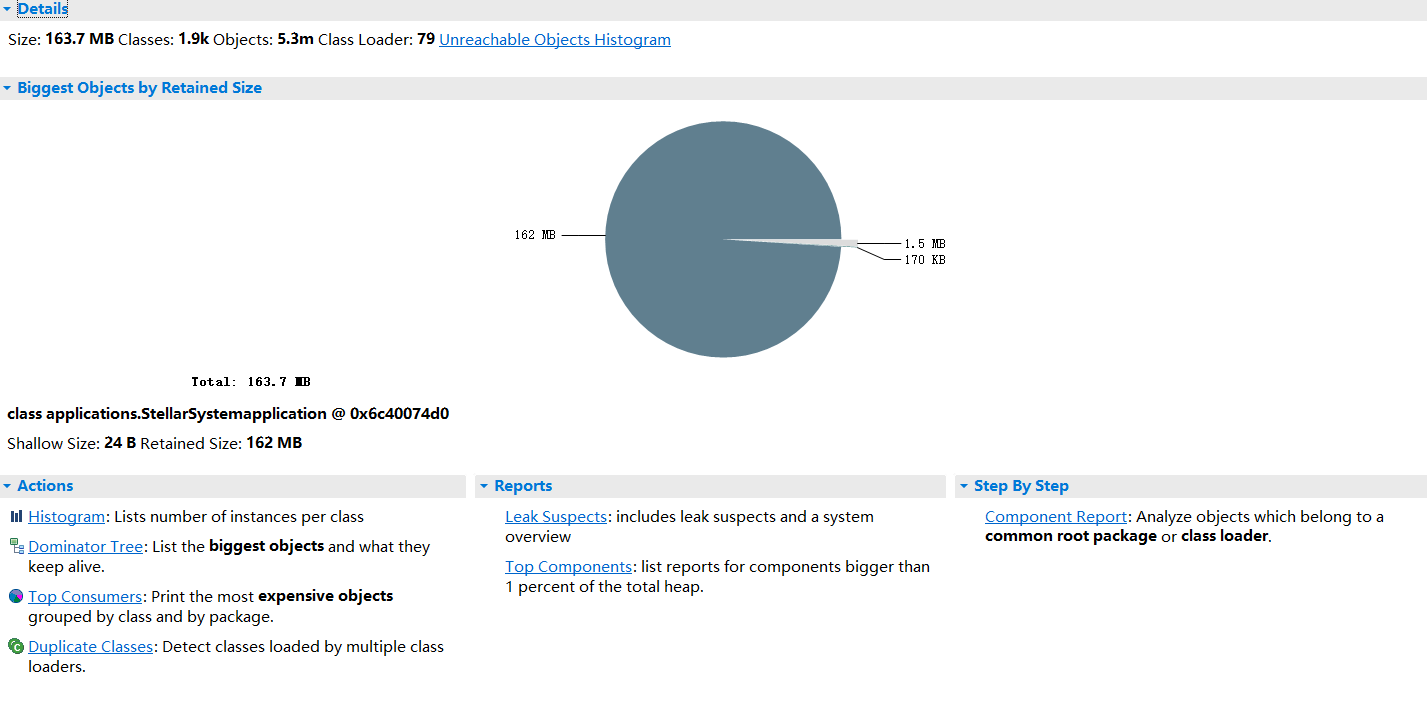
## Memory Dump Analysis and Performance Optimization

### 内存导出

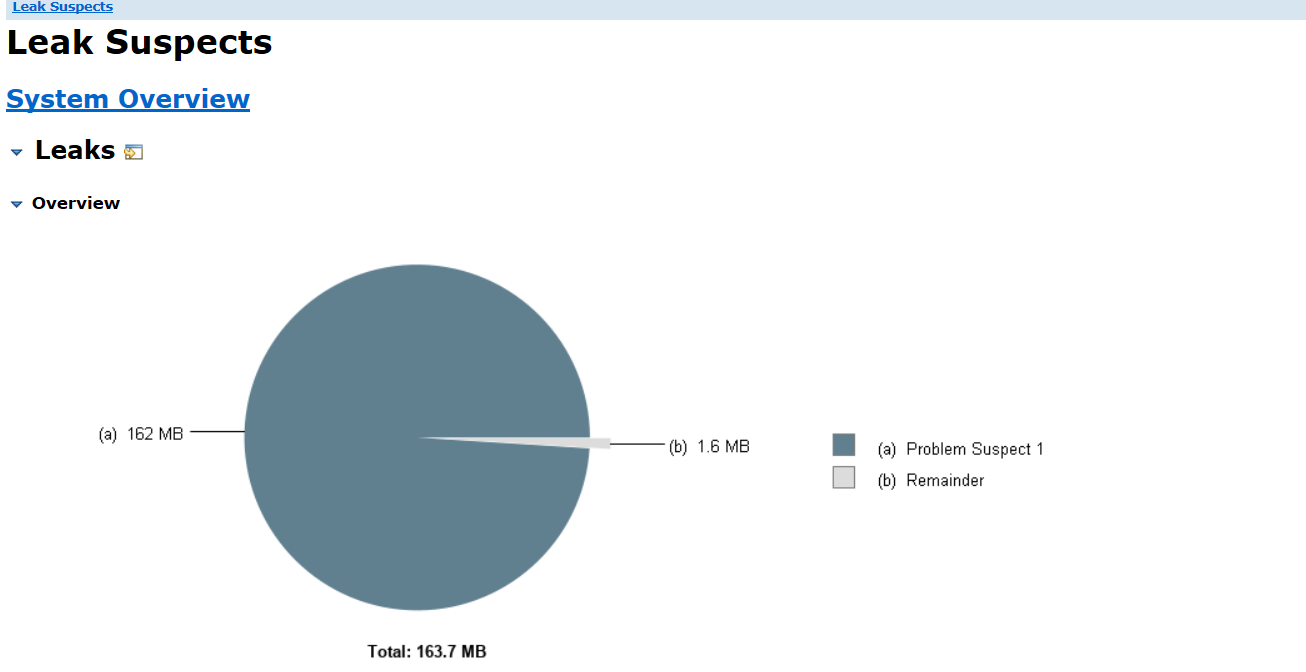


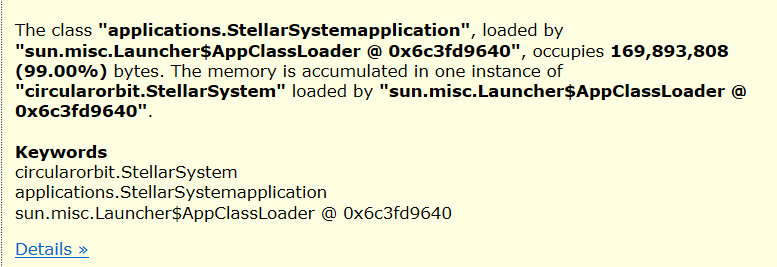
### 使用MAT分析内存导出文件

1.overview



2.leaksuspect





### 发现热点/瓶颈并改进、改进前后的性能对比分析

热点：对于每次添加的数据进行合法性检查，太浪费时间，需要的时间复杂度是O（n2）.

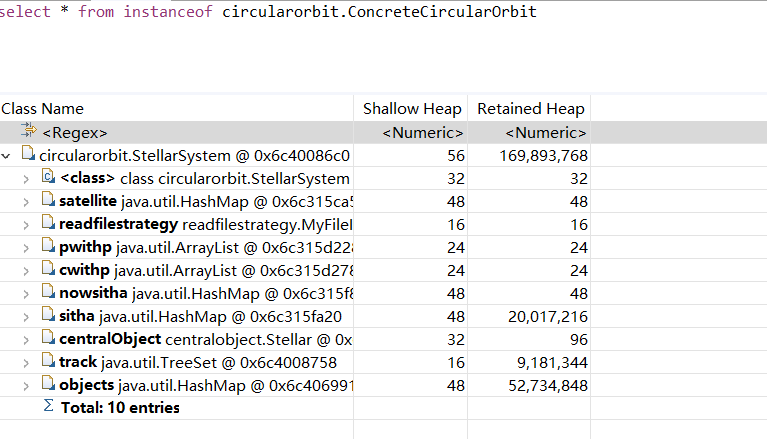
对合法性检查进行修改，发现速度加快了一大截。

我们对是否有重复名字的对象，不是遍历整个已经添加的对象，而是我们重新建立一个set集合，判断添加物体的名字是否在已经添加物体的名字的集合中。

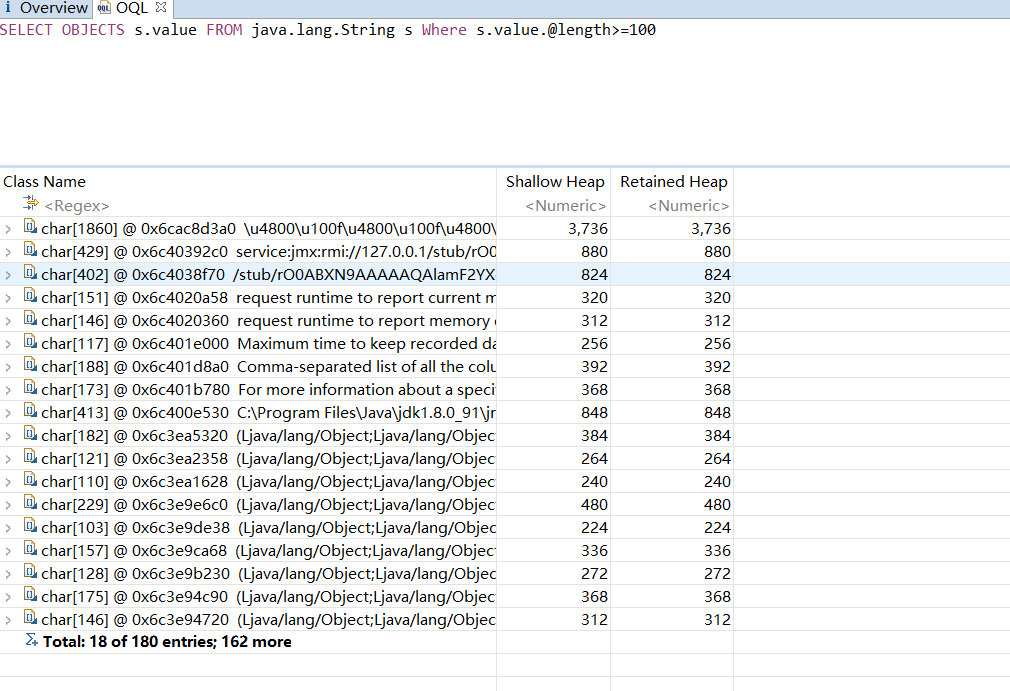
通过对集合的判断，节省了很多时间。

### 在MAT内使用OQL查询内存导出

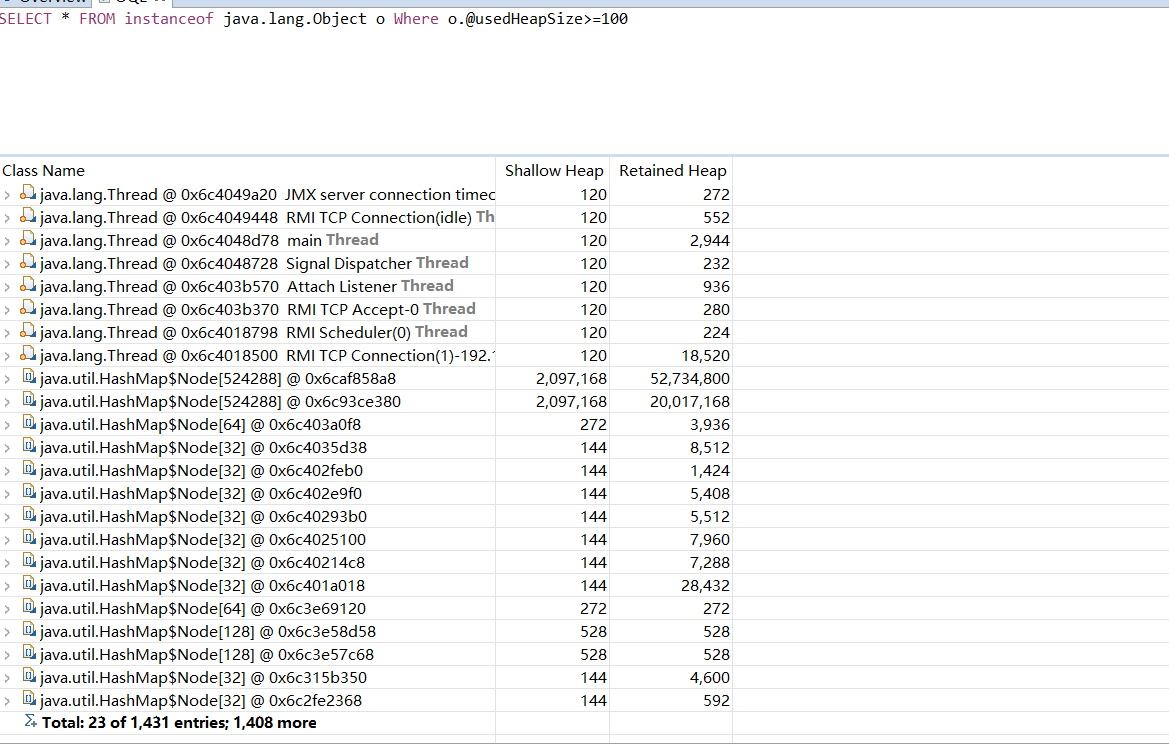
1/CircularOrbit类的实例



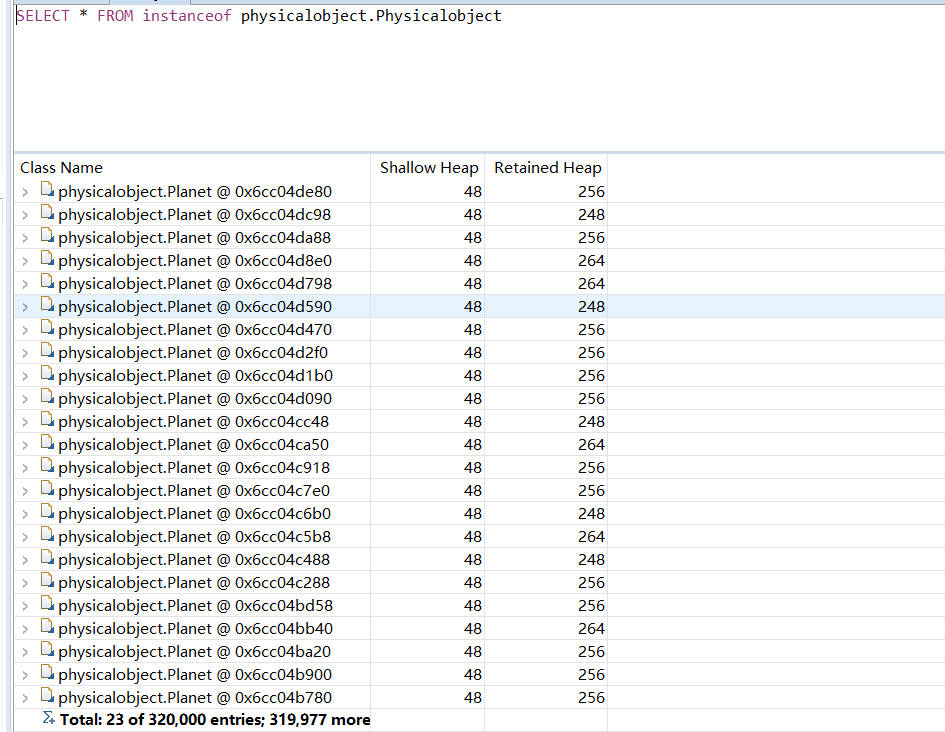
2. 特定长度n的字符串对象‘



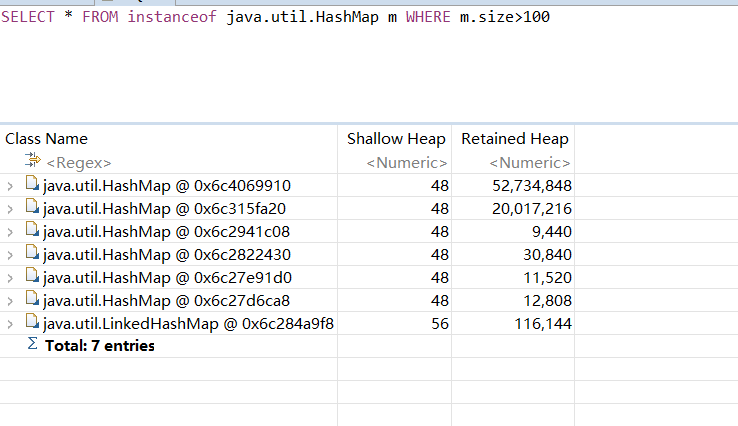
3大于特定大小的任意对象实例



4PhysicalObject（及其子类）的对象实例的数量和总占用内存大小；

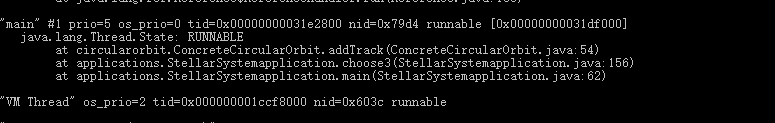


5. 所有包含元素数量大于 100 的 Collections 实例；

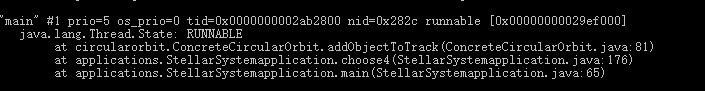


### 观察jstack/jcmd导出程序运行时的调用栈

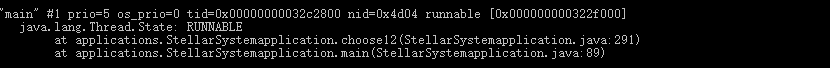
**1.增加轨道、删除轨道**



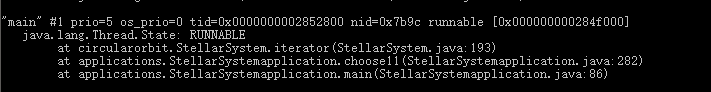
2. 向特定轨道上增加物体、从特定轨道上删除物体



3. 判断多轨道系统的合法性



4. 从里到外，从小角度到大角度输出所有行星

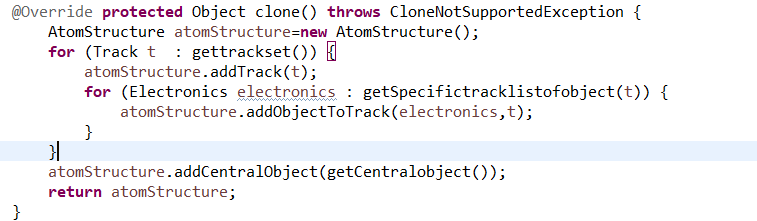


### 使用设计模式进行代码性能优化

1）使用FlyWeight设计模式，基本思想就是对于每一个轨道只有一个Electron实例。因为以前的程序是依据引用（内存地址）判断相同的，在添加物体时的普遍规则是如果轨道系统中有相同的对象则报错，此处需要在子类中覆盖该方法，不进行对象相同检查。

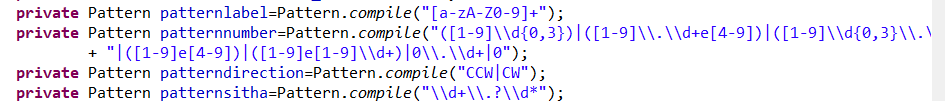


2) 基于 Prototype 设计模式的思想改造程序，支持 clone；



3）对于Track类，程序中采用值相等（轨道半径相等）的方法判断对象相同，因此可以使用FlyWeight设计模式进行优化。对于一个轨道半径的轨道只建立一个实例。

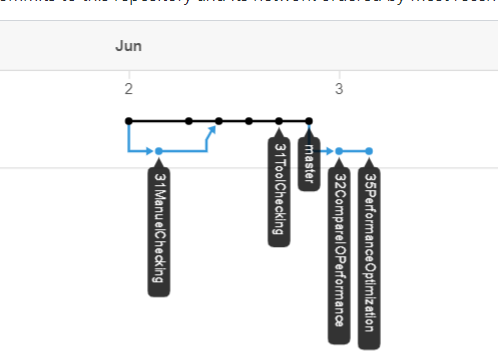
4）使用ObjectPool思想修改



对于每次判断时，不新创建pattern比对对象，直接设置几个永存的全局变量

5）使用 ”” 而不是用new String方法声明字符串变量

## Git仓库结构



# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 6.1 | 8:00-19:00 | 写完IO代码并进行部分优化 | 完成 |
| 6.2 | 10:00-19:00 | 优化其他部分+各种工具的使用 | 完成 |
| 6.3 | 18:00-20:00 | 修改一些错误+报告 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 命令行运行java程序 | 百度解决 |
|  |  |
|  |  |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

算法真的太重要了

## 针对以下方面的感受

1. 代码“看起来很美”和“运行起来很美”，二者之间有何必然的联系或冲突？哪个比另一个更重要些吗？在有限的编程时间里，你更倾向于把精力放在哪个上？

运行起来很美更重要，毕竟用户体验优先

1. 诸如SpotBugs和CheckStyle这样的代码静态分析工具，会提示你的代码里有无数不符合规范或有潜在bug的地方，结合你在本次实验中的体会，你认为它们是否会真的帮助你改善代码质量？

会

1. 为什么Java提供了这么多种I/O的实现方式？从Java自身的发展路线上看，这其实也体现了JDK自身代码的逐渐优化过程。你是否能够梳理清楚Java I/O的逐步优化和扩展的过程，并能够搞清楚每种I/O技术最适合的应用场景
2. JVM的内存管理机制，与你在《计算机系统》课程里所学的内存管理基本原理相比，有何差异？有何新意？你认为它是否足够好？
3. JVM自动进行垃圾回收，从而避免了程序员手工进行垃圾回收的麻烦（例如在C++中）。你怎么看待这两种垃圾回收机制？你认为JVM目前所采用的这些垃圾回收机制还有改进的空间吗？
4. 基于你在实验中的体会，你认为“通过配置JVM内存分配和GC参数来提高程序运行性能”是否有足够的回报？
5. 通过Memory Dump进行程序性能的分析，JMC/JFR、VisualVM和MAT这几个工具提供了很强大的分析功能。你是否已经体验到了使用它们发现程序热点以进行程序性能优化的好处？
6. 使用各种代码调优技术进行性能优化，考验的是程序员的细心，依赖的是程序员日积月累的编程中养成的“对性能的敏感程度”。你是否有足够的耐心，从每一条语句、每一个类做起，“积跬步，以至千里”，一点一点累积出整体性能的较大提升？
7. 关于本实验的工作量、难度、deadline。
8. 到目前为止，你对《软件构造》课程的意见与建议。