

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 5实验报告**

|  |  |
| --- | --- |
| 姓名 | 郭潇 |
| 学号 | 1160300521 |
| 班号 | 03005 |
| 电子邮件 | Wqsnzszw@163.com |
| 手机号码 | 18523735573 |

**目录**

[1 实验目标概述 1](#_Toc513573923)

[2 实验环境配置 1](#_Toc513573924)

[3 实验过程 1](#_Toc513573925)

[3.1 Static Program Analysis 1](#_Toc513573926)

[3.1.1 人工代码走查（walkthrough） 1](#_Toc513573927)

[3.1.2 使用CheckStyle和FindBugs进行静态代码分析 1](#_Toc513573928)

[3.2 Java I/O 1](#_Toc513573929)

[3.2.1 多种I/O实现方式 1](#_Toc513573930)

[3.2.2 多种I/O实现方式的效率对比分析 1](#_Toc513573931)

[3.3 Java Memory Management and Garbage Collection (GC) 1](#_Toc513573932)

[3.3.1 使用-verbose:gc参数 1](#_Toc513573933)

[3.3.2 用jstat命令行工具的-gc和-gcutil参数 1](#_Toc513573934)

[3.3.3 使用jmap -heap命令行工具 1](#_Toc513573935)

[3.3.4 使用jmap -histo命令行工具（可选） 1](#_Toc513573936)

[3.3.5 使用jmap -permstat命令行工具（可选） 1](#_Toc513573937)

[3.3.6 使用jconsole或VisualVM工具 1](#_Toc513573938)

[3.3.7 分析垃圾回收过程是否正常、异常 1](#_Toc513573939)

[3.3.8 配置JVM参数并发现最优参数配置 1](#_Toc513573940)

[3.4 Dynamic Program Profiling 1](#_Toc513573941)

[3.4.1 使用Visual VM进行CPU Profiling 1](#_Toc513573942)

[3.4.2 使用Visual VM进行Memory profiling 1](#_Toc513573943)

[3.5 Memory Dump Analysis and Performance Optimization 1](#_Toc513573944)

[3.5.1 内存导出(memory dump) 1](#_Toc513573945)

[3.5.2 使用MAT分析内存导出文件 1](#_Toc513573946)

[3.5.3 发现热点/瓶颈并改进、改进前后的性能对比分析 1](#_Toc513573947)

[3.5.4 jhat和OQL查询内存导出（可选） 1](#_Toc513573948)

[3.5.5 jstack导出java程序运行时的调用栈（可选） 1](#_Toc513573949)

[4 实验进度记录 1](#_Toc513573950)

[5 实验过程中遇到的困难与解决途径 1](#_Toc513573951)

[6 实验过程中收获的经验、教训、感想 1](#_Toc513573952)

# 实验目标概述

本次实验通过对 Lab4 的代码进行静态和动态分析，发现代码中存在的不符 合代码规范的地方、具有潜在 bug 的地方、性能存在缺陷的地方(执行时间热点、 内存消耗大的语句、函数、类)，进而使用第 4、7、8 章所学的知识对这些问题 加以改进，掌握代码持续优化的方法，让代码既“看起来很美”，又“运行起来 很美”。

具体训练的技术包括:

1.静态代码分析(CheckStyle 和 FindBugs)

2.动态代码分析(Java 命令行工具 jstat、jmap、jConsole、VisualVM)

3.JVM内存管理与垃圾回收(GC)的优化配置

4. 运行时内存导出(memory dump)及其分析(Java 命令行工具 jhat、MAT)

5. 运行时调用栈及其分析(Java 命令行工具 jstack); 

6.高性能I/O;

7.基于设计模式的代码调优

8.代码重构

# 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）。

https://github.com/ComputerScienceHIT/Lab5-1160300521

# 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）

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。

1. 布局上先下载google的代码风格文件，设置为默认的code style，再利用eclipse的format功能变为规定格式。
2. 注释上，为vertex、edge和graph中的各个类添加java doc注释。

### 使用CheckStyle和FindBugs进行静态代码分析

使用checkstyle遇到的问题：

1. java doc注释格式不正确，根据checkstyle的提示，完善注释。
2. 多行代码的长度超过80，将长代码变为多行，在以后的编程中避免这种情况。
3. 多个方法需要在输入参数前用“final”标注，这样可避免输入参数被修改。
4. 对于为被继承的方法需添加“final”标注。
5. 函数输入的参数不能与类中定义的变量名称相同。
6. 删除不必要的空白字符。

使用findbugs遇到的问题：

对于每个vertex和 edge中各个类的equal方法未考虑比较对象为null的情况导致findbugs报错，于是加入对null情况的考虑。

Checkstyle和findbugs相比，checkstyle主要关注代码的格式问题，使得代码的可读性增强，更加美观，便于理解；findbugs主要关注运行中可能导致错误的地方，使得程序的正确性和健壮性增强。

## Java I/O

### 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。

实现了Stream、Reader/Writer、Buffer/Channel、Scanner（只有读文件）、java.nio.file.Files方式。

1. Stream，每次读入一个字节，读到换行符时则获取一行图的指令，利用该条指令对图进行操作，再将这条指令写入相应文件。
2. Reader/Writer，每次读入一行图的指令，利用该条指令对图进行操作，再将这条指令写入相应文件。
3. Buffer/Channel，每次读入一行图的指令，利用该条指令对图进行操作，再将这条指令写入相应文件。
4. Scanner，每次读入一个字节，读到换行符时则获取一行图的指令，利用该条指令对图进行操作。
5. java.nio.file.Files，每次读入一行图的指令，利用该条指令对图进行操作，再将这条指令写入相应文件。

如何用strategy设计模式实现在多种I/O策略之间的切换。

在各个图的factory的中对每种策略写一种build方法，根据用户选取的读写策略，调用相应的方法即可。

### 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。

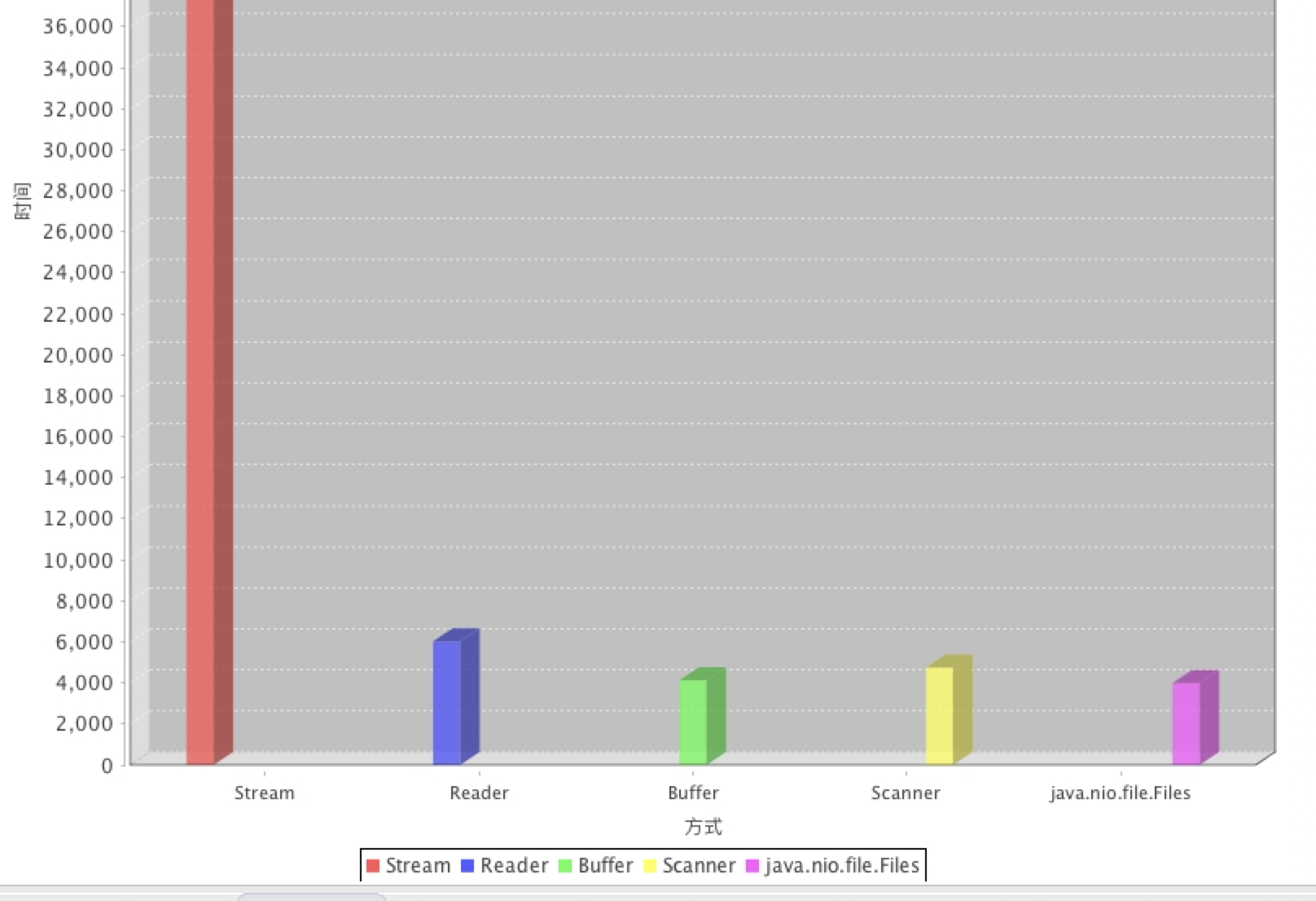
分别统计只读取文件和写入文件的时间，其中读入文件包含对图的解析，而写入文件只讲获取的信息直接写入。

表格方式对比不同I/O的性能。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Fiel1.txt | Fiel2.txt | Fiel3.txt | Fiel4.txt |
| Stream | 读文件(ms) | 51756 | 67950 | 54624 | 88790 |
| 写文件(ms) | 1732 | 7554 | 2679 | 3547 |
| Reader/Writer | 读文件(ms) | 6025 | 7522 | 6873 | 32434 |
| 写文件(ms) | 955 | 1027 | 358 | 4393 |
| Buffer/Channel | 读文件(ms) | 4127 | 4994 | 4232 | 30840 |
| 写文件(ms) | 552 | 587 | 246 | 2305 |
| Scanner | 读文件(ms) | 4751 | 5729 | 4904 | 31134 |
| java.nio.file.Files | 读文件(ms) | 3994 | 4918 | 4190 | 29494 |
| 写文件(ms) | 183 | 189 | 245 | 4658 |

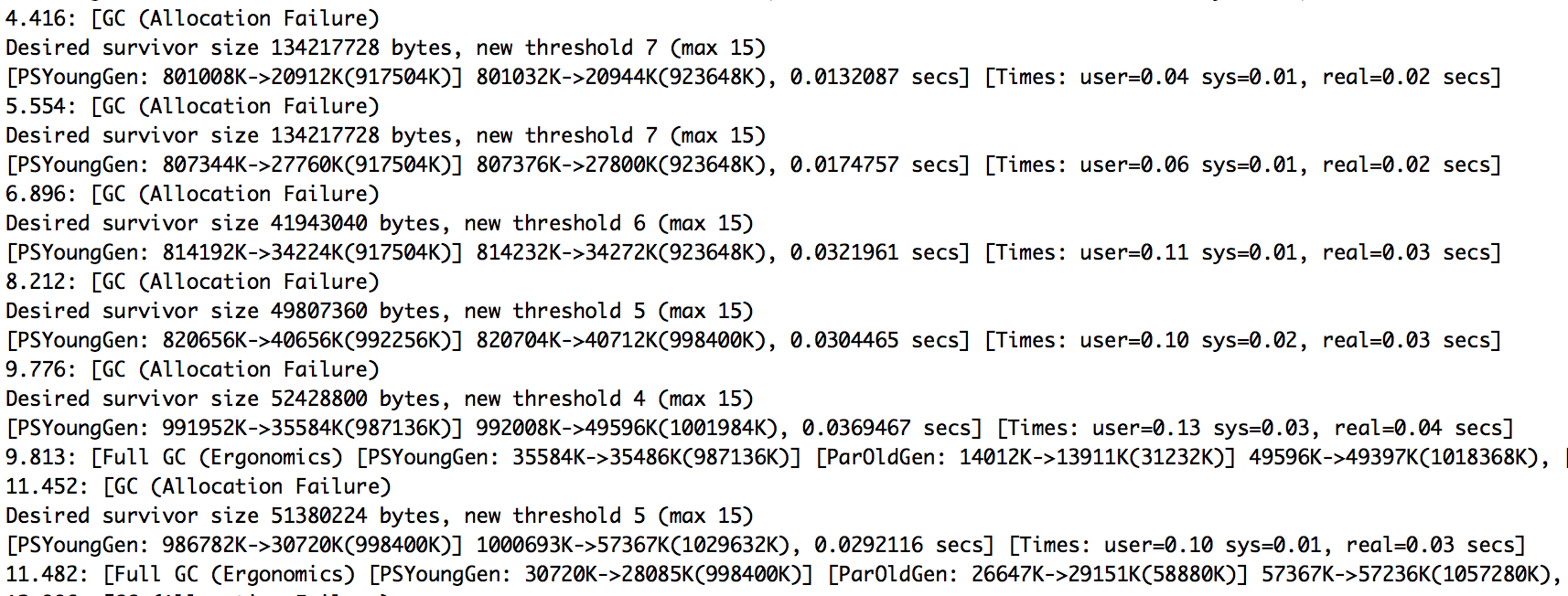
图形对比不同I/O的性能（可选）。

编写方法绘制柱状图用来比较读入时间的长短，先获取读取时间长度的数组，在利用数组数据绘图。绘制的图如下：



## Java Memory Management and Garbage Collection (GC)

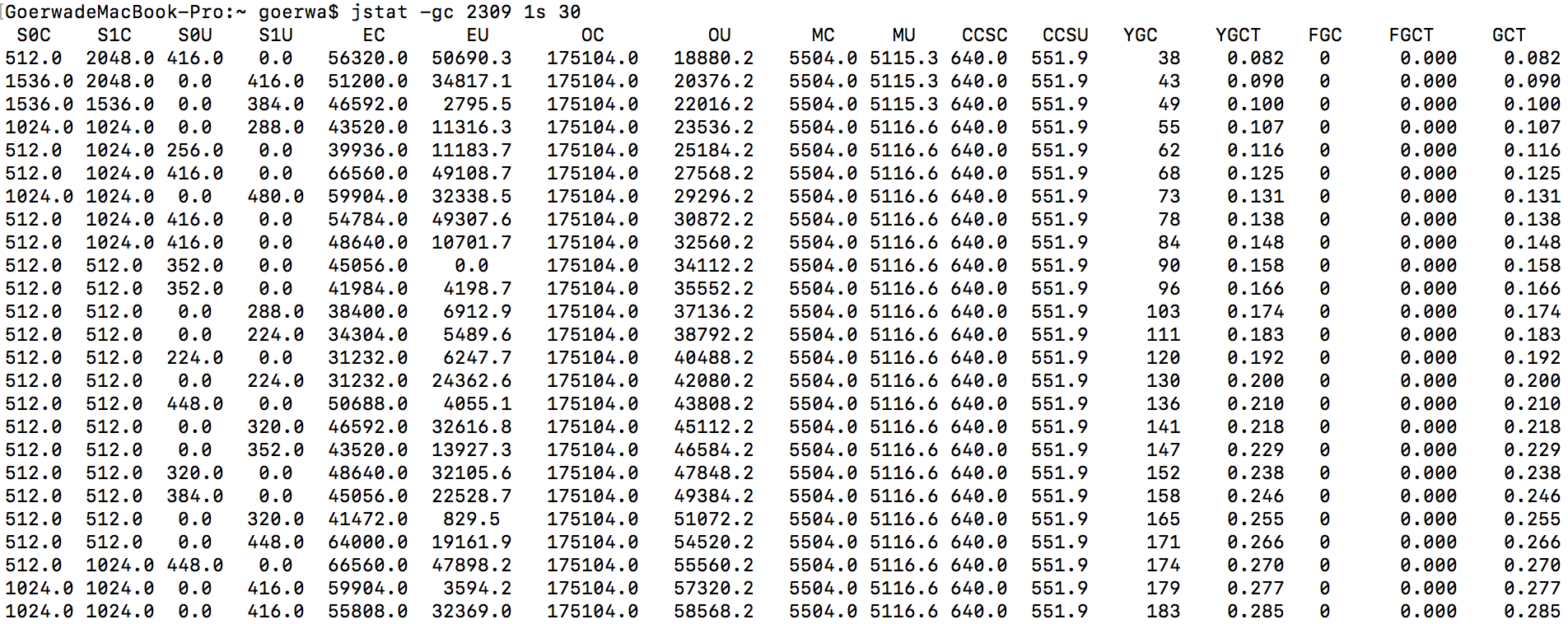
### 使用-verbose:gc参数

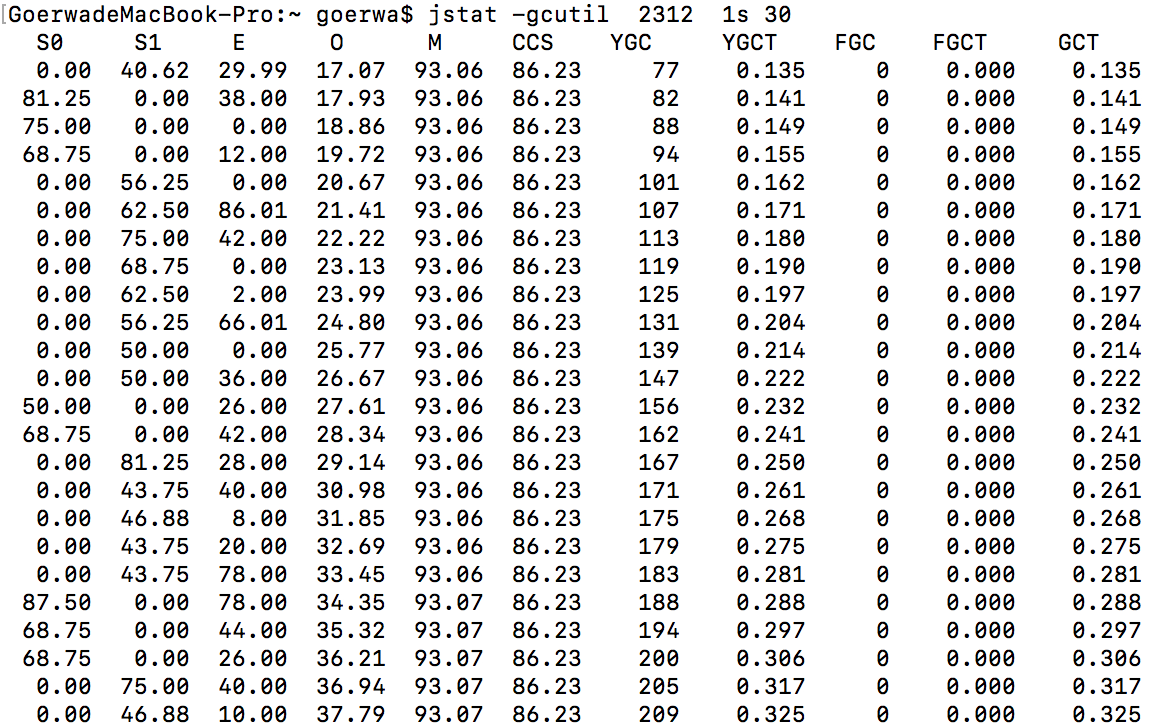


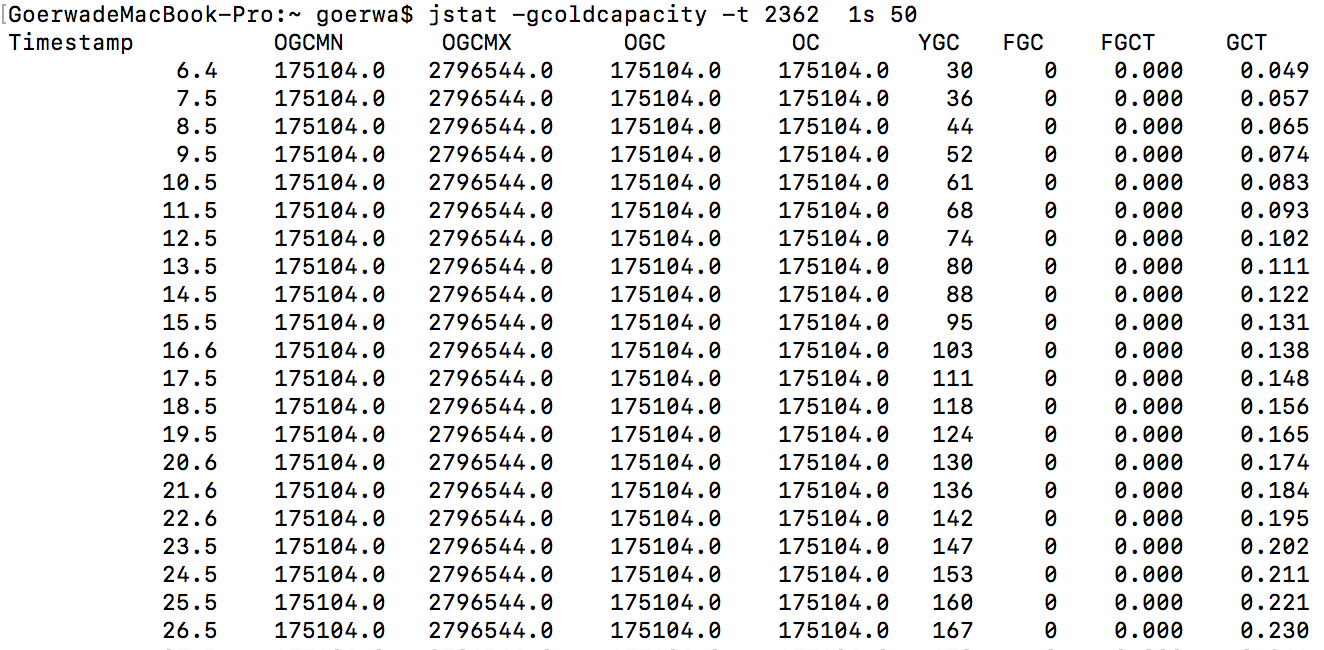
1. Minor GC 与 Full GC 相比发生的频率更频繁。
2. Minor GC 单次时间约为0.02秒； Full GC单次时间约为0.25秒
3. 每次Minor GC前后，新生代减少；每次Full GC前后，新生代减少，老生代增加。

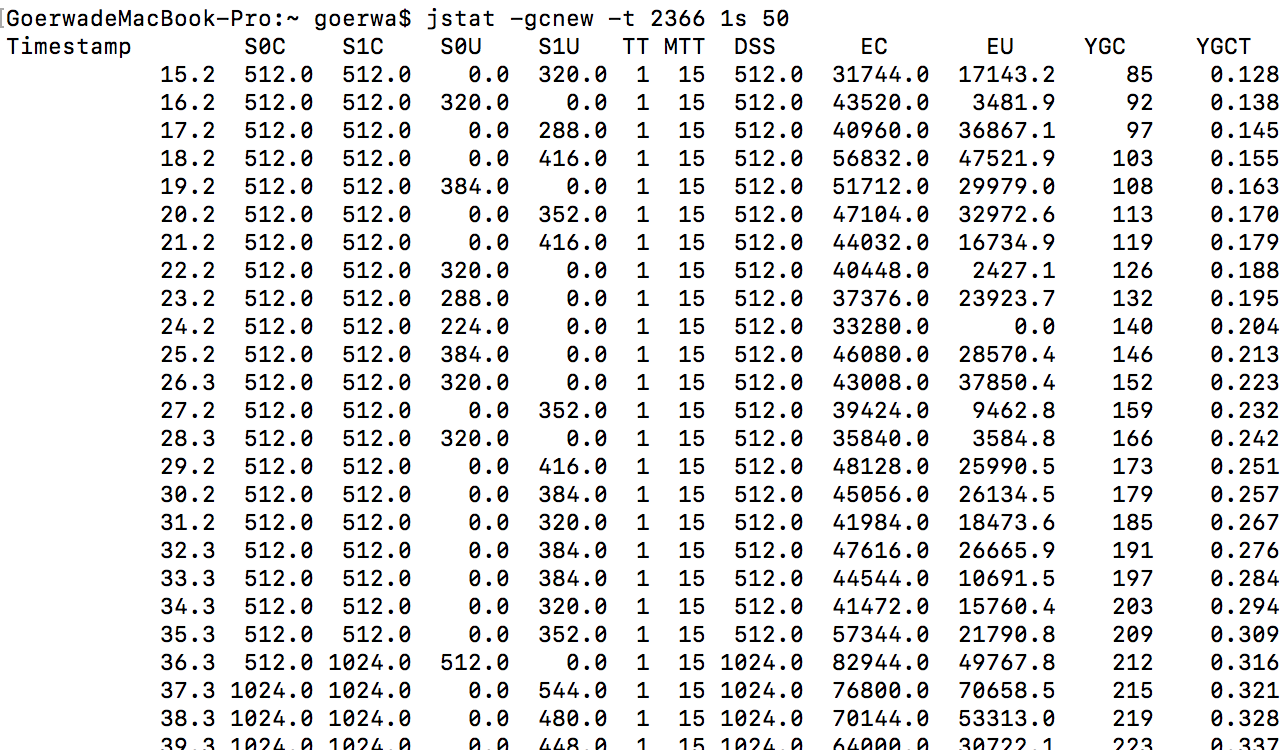
基于观察，我认为由于频繁的创建新的对象，Eden区满了触发Minor GC，把Eden区存活的对象复制到Survivor区，所以Survivor区不断增大。

### 用jstat命令行工具的-gc和-gcutil参数



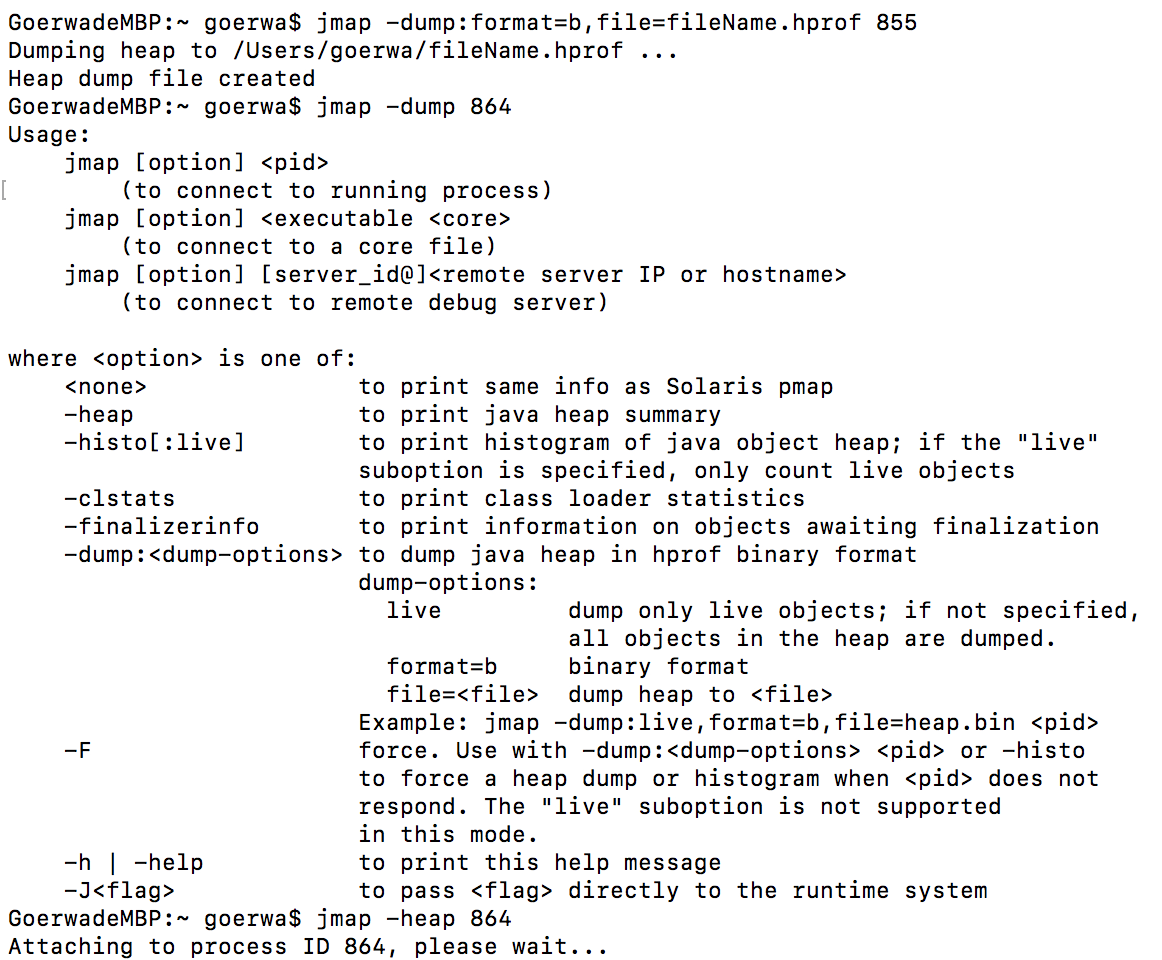




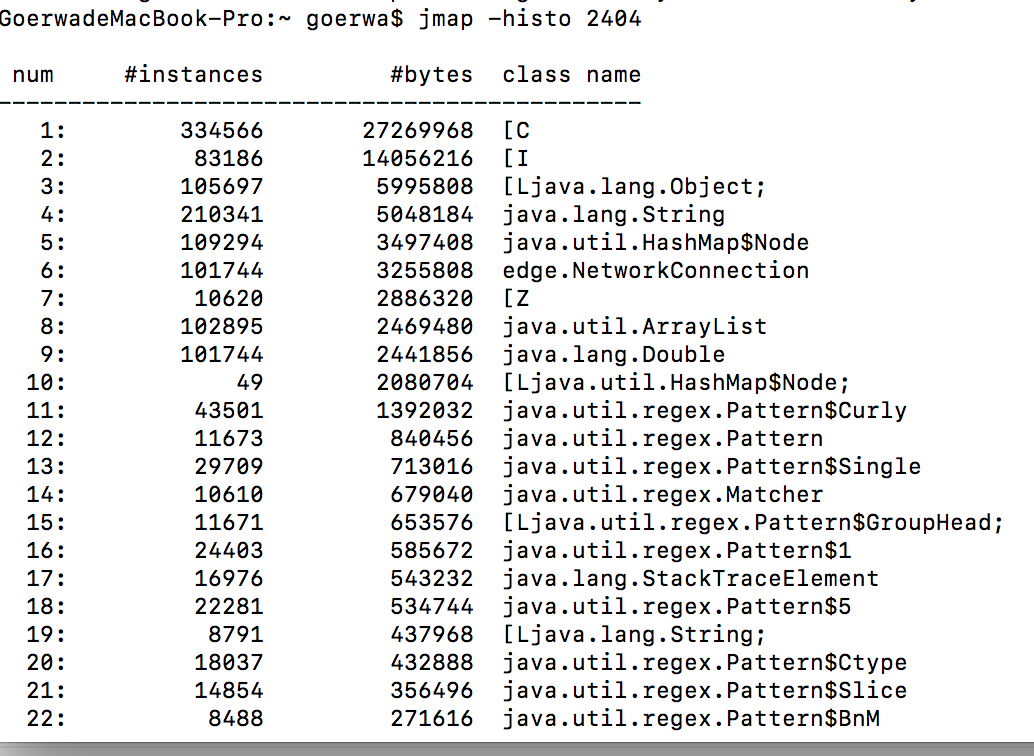


使用-gc；-gcutil；-gccapacity；-gcnew指令观察发现在读取过程中不发生FULL GC，原因应该是老生代占用过少。

### 使用jmap -heap命令行工具

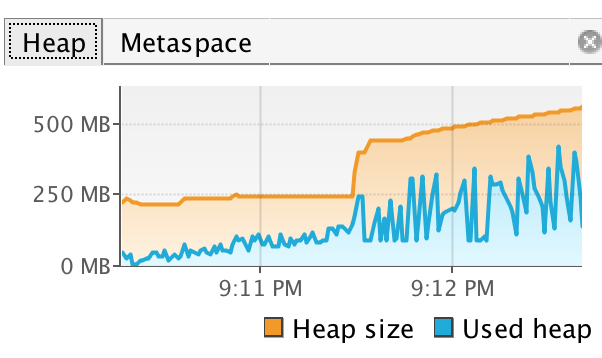


### 使用jmap -histo命令行工具（可选）

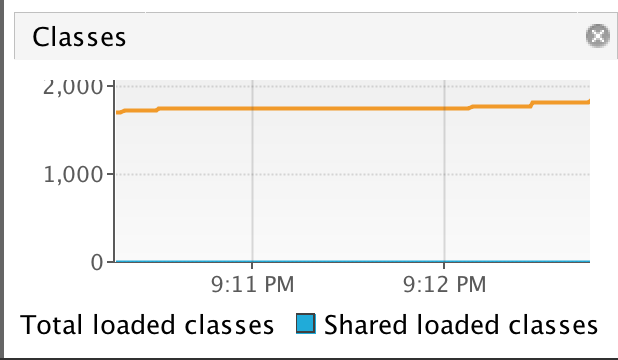


### 使用jmap -permstat命令行工具（可选）

### 使用jconsole或VisualVM工具







### 分析垃圾回收过程是否正常、异常

使用的堆区在增大过程中伴随着减小，说明垃圾回收正常进行；堆区在满后自动扩大，内存使用情况正常。

### 配置JVM参数并发现最优参数配置

1.-Xmx<n> 指定 jvm 的最大 heap 大小 , 经过测试发现，设置大小过小会使发生FULL GC变得更频繁，所以逐渐调大该部分，最后确定为2G.

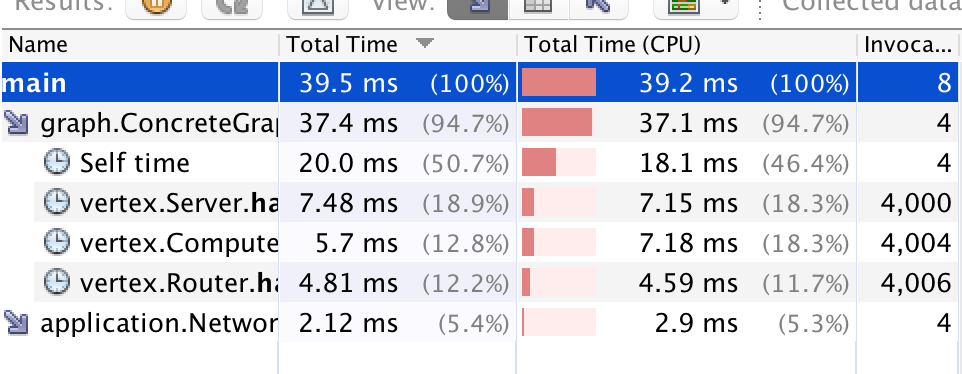
2.-Xms<n> 指定 jvm 的最小 heap 大小 ,高并发应用，为了防止因为内存收缩／突然增大带来的性能影响，逐步调大该部分，但过大也会影响性能，最后确定为1500M.   
3.-Xmn<n> 指定 jvm 中 New Generation 的大小 ,这个参数很影响性能，因为读取程序需要比较多的临时内存，但太大会增加MINOR GC的时间，经过逐步尝试后最后确定为1000M。

最后确定的参数为-Xmx2000m -Xms1500m -Xmn1000m -XX:MetaspaceSize=10m -XX:MaxMetaspaceSize=20m

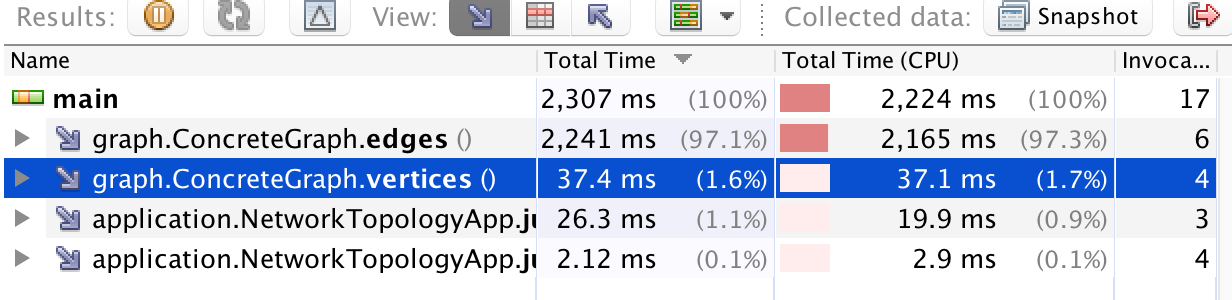
## Dynamic Program Profiling

### 使用Visual VM进行CPU Profiling

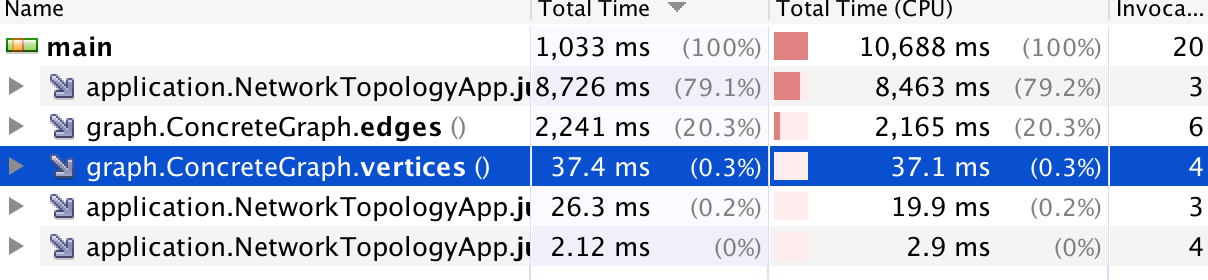
加入顶点



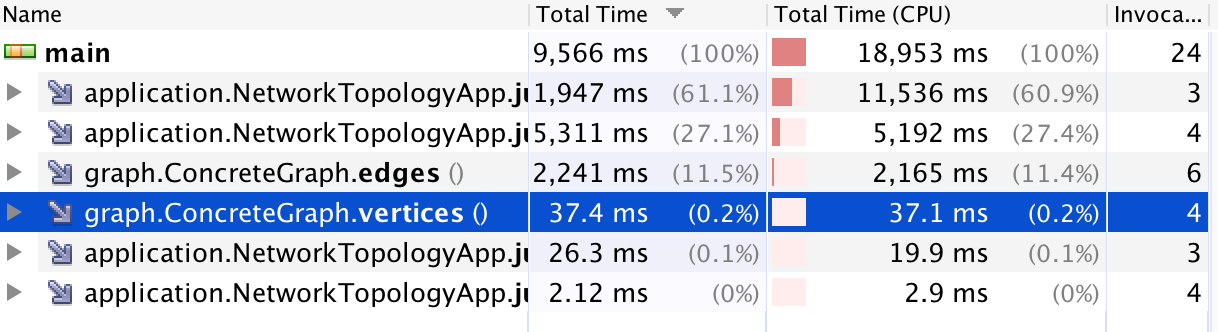
加入边



删除边

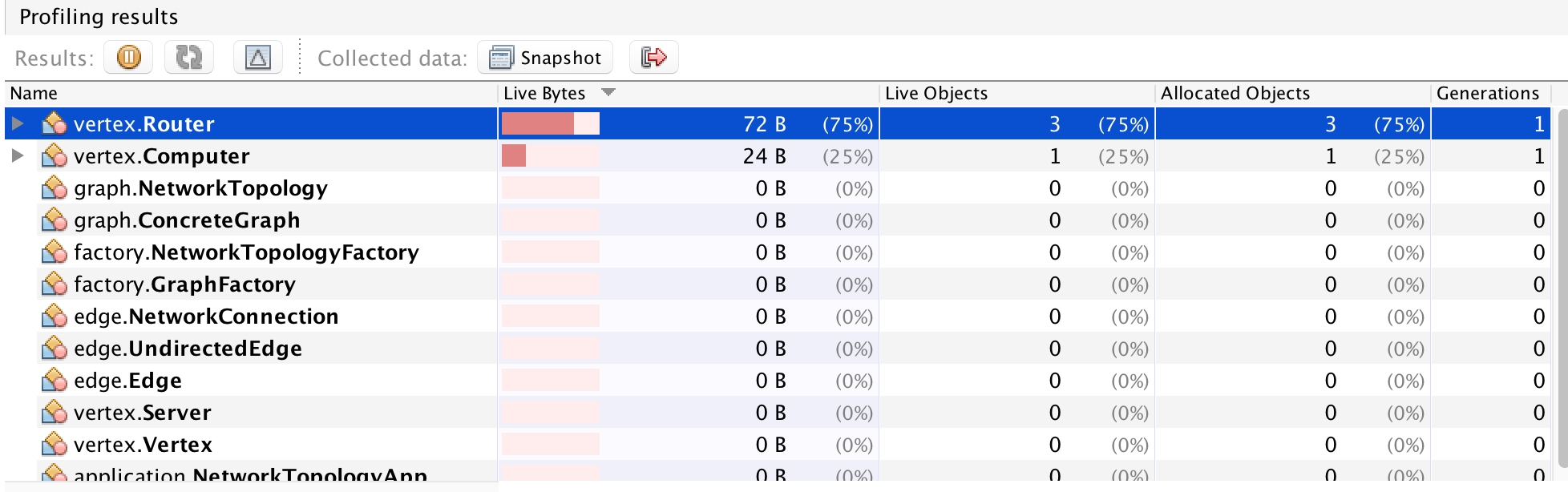


删除顶点

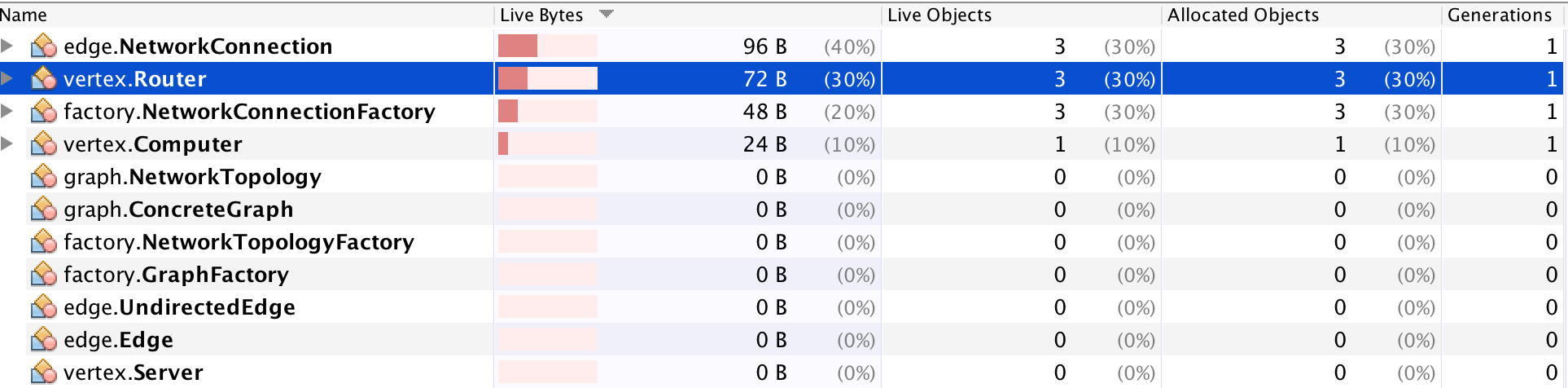


### 使用Visual VM进行Memory profiling

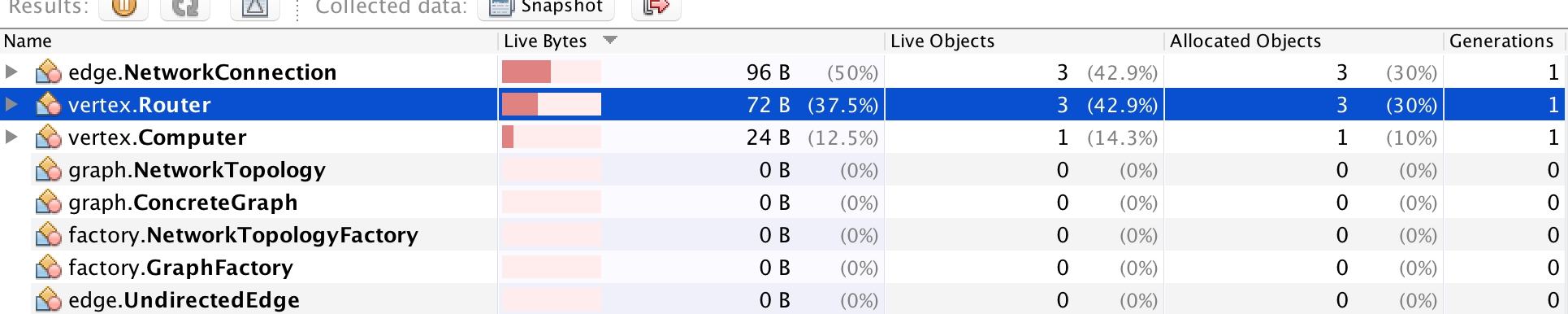
加入顶点



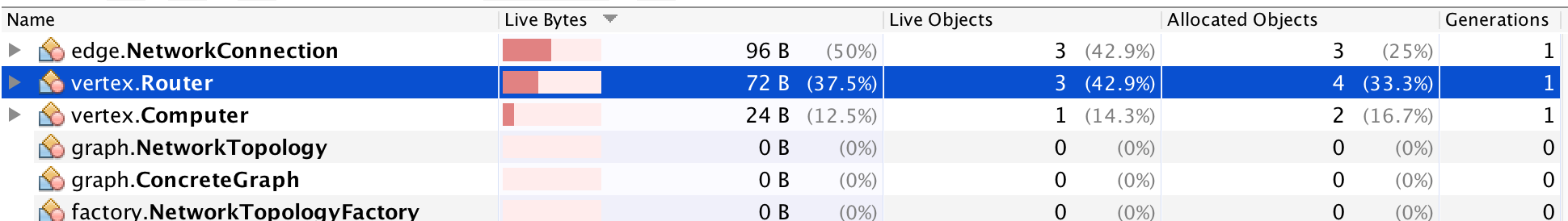
加入边



删除边



删除顶点



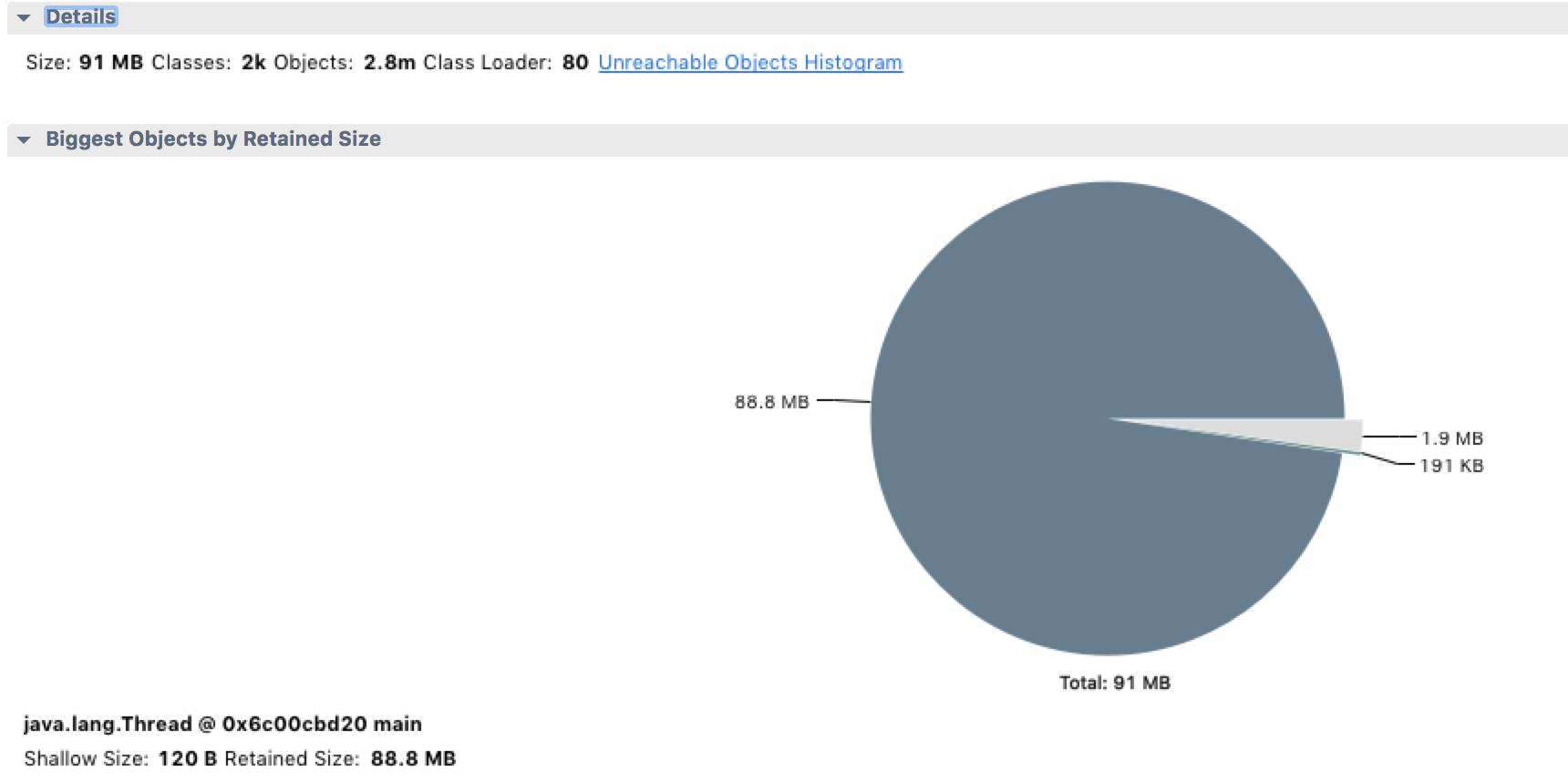
## Memory Dump Analysis and Performance Optimization

### 内存导出(memory dump)

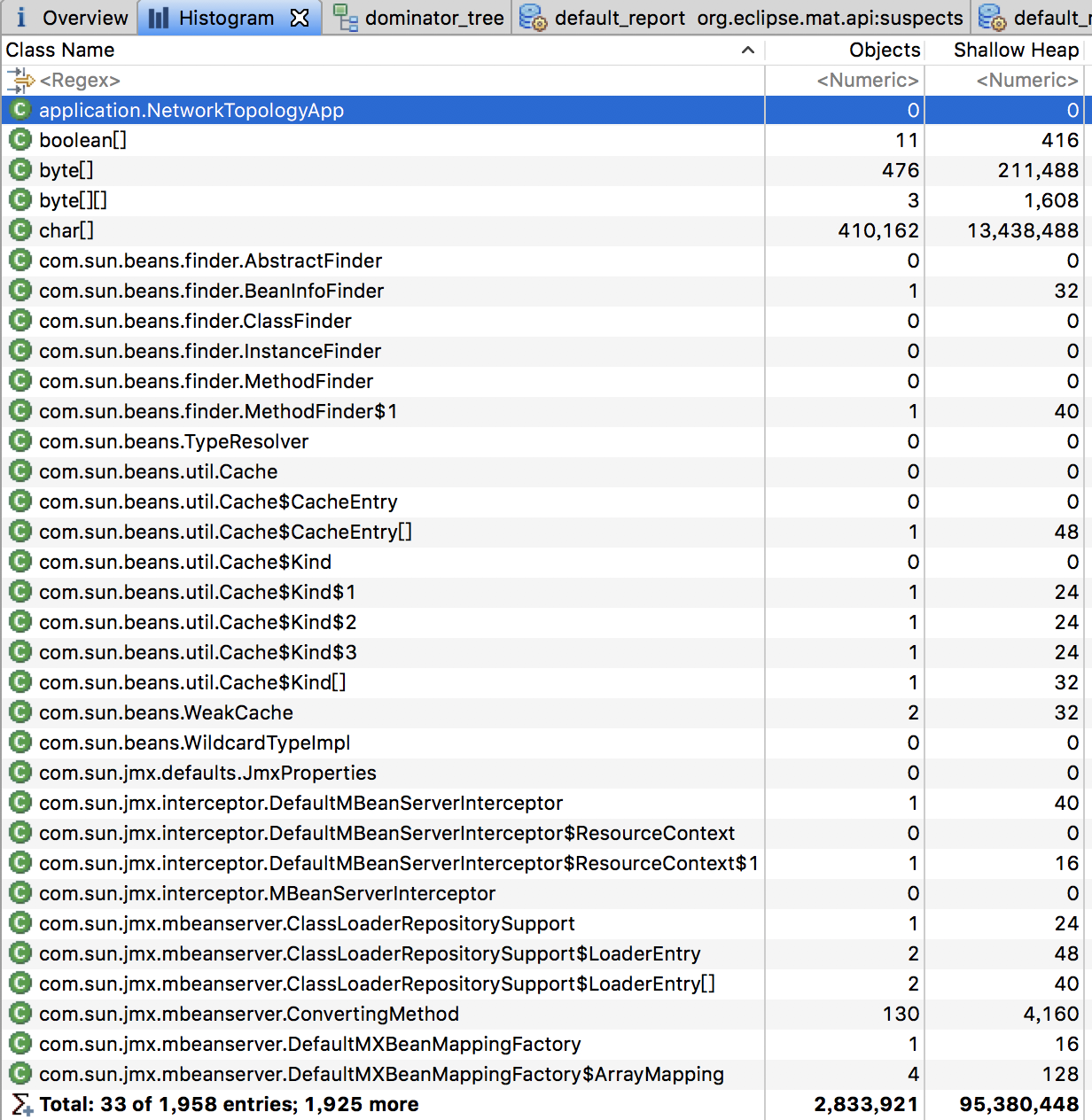
在Monitor（监视）子标签页中点击Heap Dump（堆Dump）按钮。本地应用程序的Heap dumps作为应用程序标签页的一个子标签页打开。同时，heap dump在左侧的Application（应用程序）栏中对应一个含有时间戳的节点。右击这个节点选择save as（另存为）即可将heap dump保存到本地。

### 使用MAT分析内存导出文件

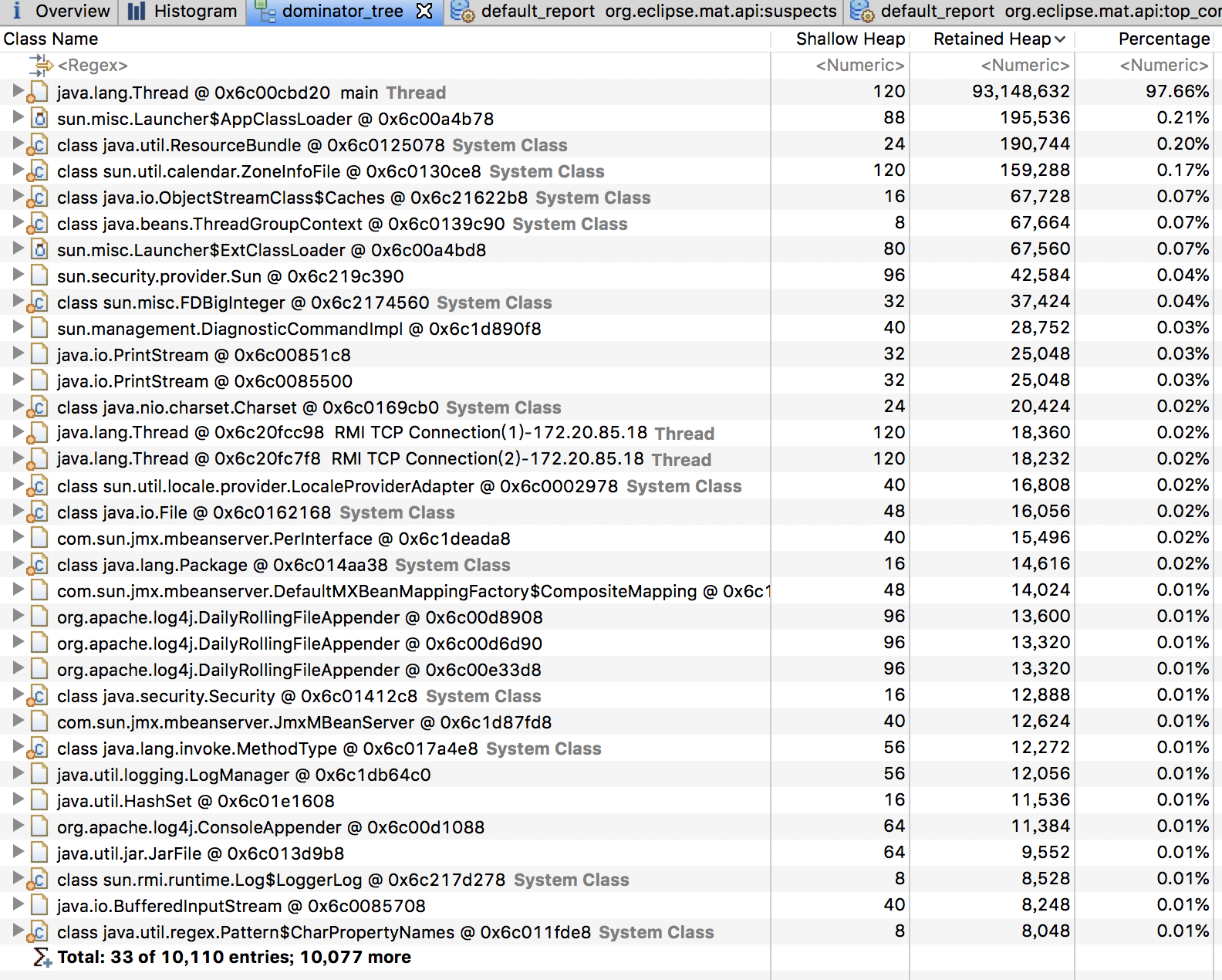
overview



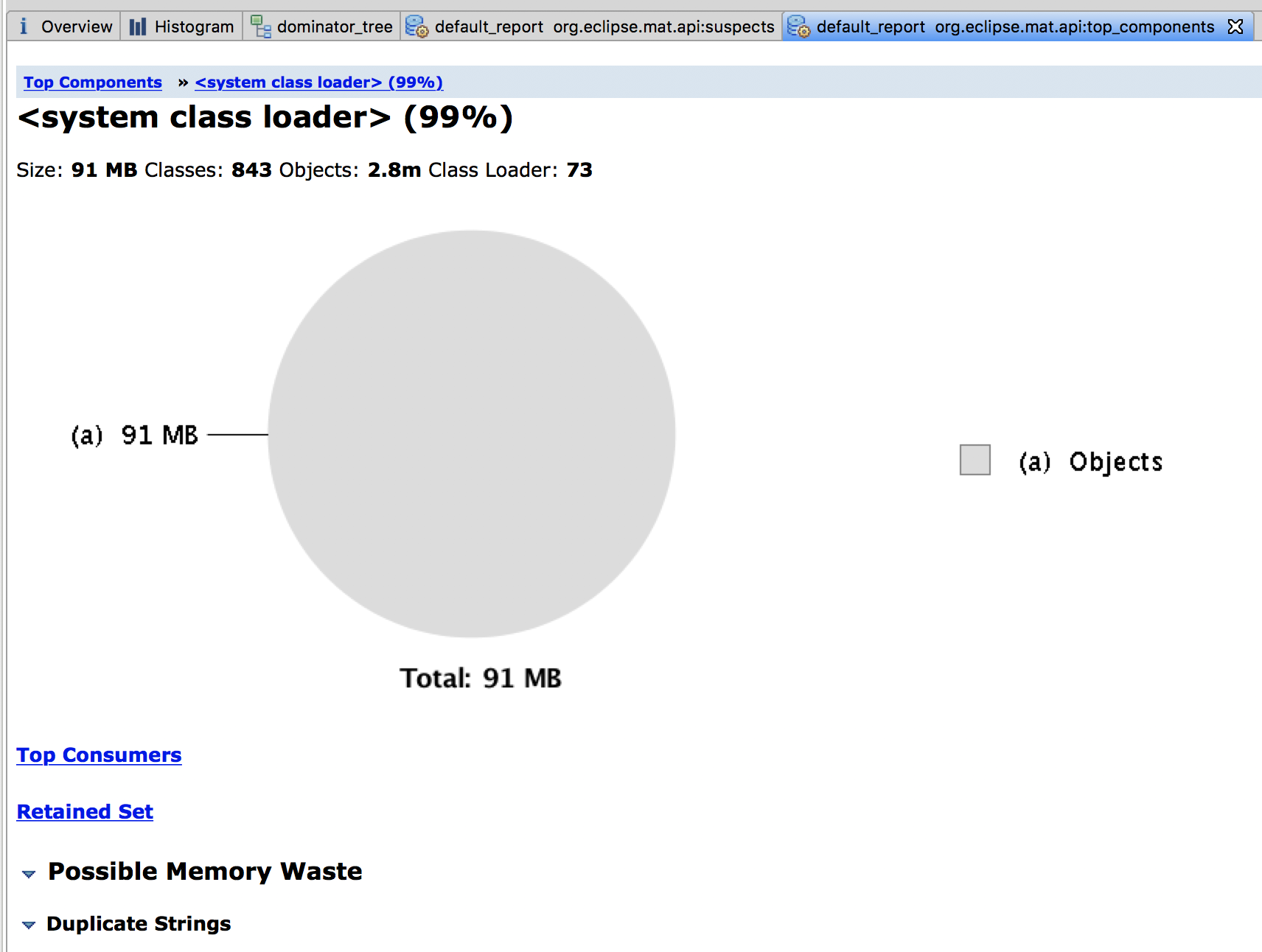
histogram



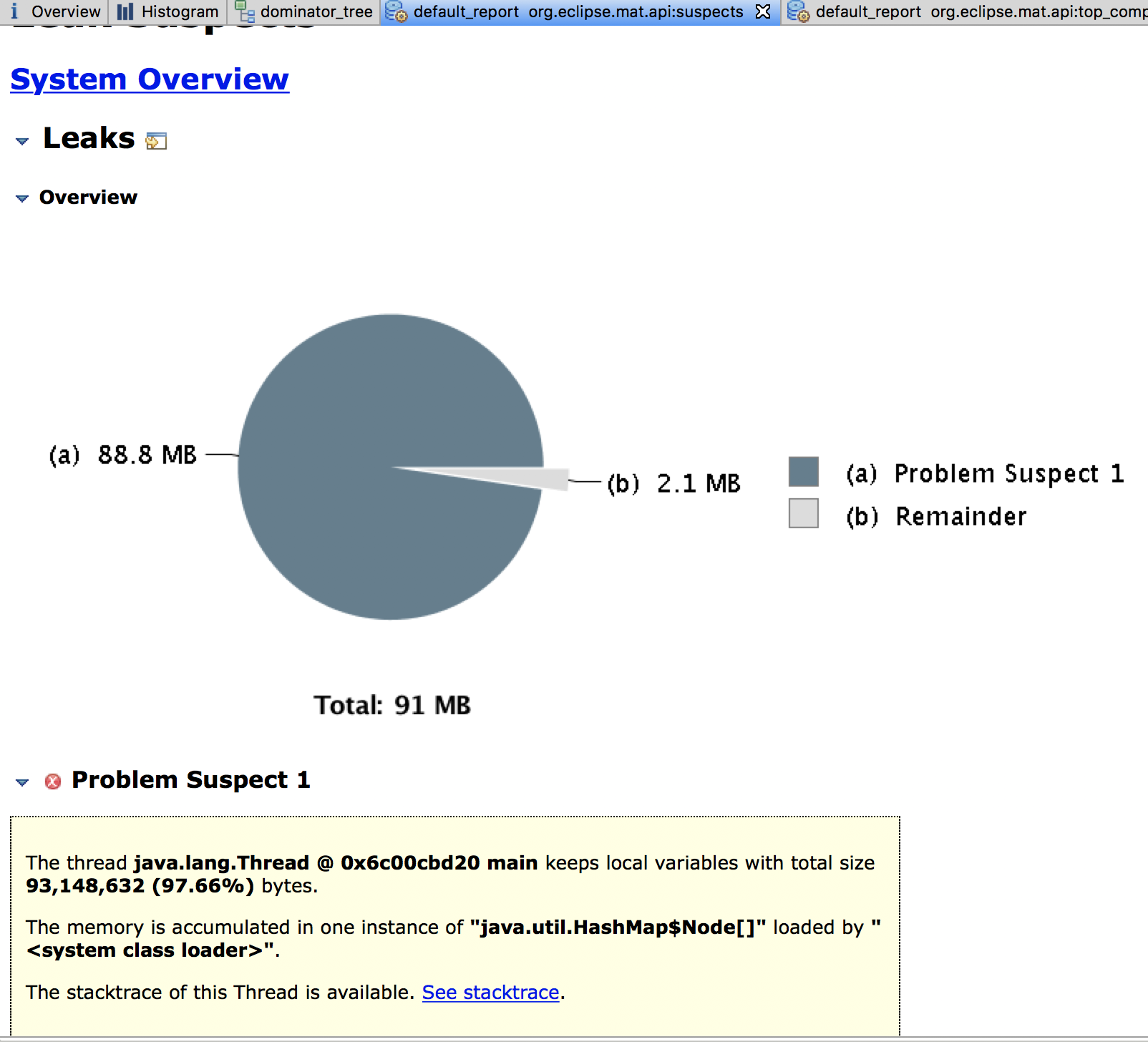
dominator tree



top consumer



leak suspects report



### 发现热点/瓶颈并改进、改进前后的性能对比分析

### jhat和OQL查询内存导出（可选）

### jstack导出java程序运行时的调用栈（可选）

# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2018-05-21 | 13：45-15：30 | 下载checksty和findbugs并了解其基本用法 | 完成 |
| 2018-05-26 | 14：00-17：00 19：00-21：00 | 根据checkstyle的提示修改代码 | 完成 |
| 2018-05-27 | 15：00-17：00 | 根据findbugs的提示修改代码 | 完成 |
| 2018-05-28 | 13：45-15：30 19：00-21：00 | 完成I/O读写的部分 | 完成 |
| 2018-05-29 | 15：30-17：30 19：00-20：00 | 继续完善I/O读写的部分并学习利用jfreechar绘图 | 完成 |
| 2018-05-30 | 18：30-21：00 | 完成3.3的部分 | 对GC部分内容不熟悉，未完成 |
| 2018-05-31 | 18：30-20：00 | 完成3.3的部分 | 完成 |
| 2018-06-01 | 19：30-21：00 | 完成3.4的部分 | 完成 |
| 2018-06-02 | 14：30-17：30 | 完成3.5的部分 | 完成 |
| 2018-06-03 | 10：00-12；00 | 完成实验报告 | 完成 |

# 实验过程中遇到的困难与解决途径

1. 在之前的编程中为注意代码格式，导致checkstyle提示大量错误，需要大量时间都进行修改。

解决：对vertex，edge，graph中的代码进行修改，了解了java doc的写法和规范的代码格式，在今后的编写过程中加以重视。

1. 在使用stream和scanner读入文件时，尝试一次读取多个字节或字符，但难以分清各种指令的界限

解决：多次尝试后，未找出通用的方法，于是只能使用一次读取一个字节或字符的方式。

1. 对GC部分的内容不熟悉，完成3.3部分时不知如何分析

解决：复习ppt，查阅相关资料对GC的机制有基本了解后再进行实验。

1. 出现错误提示：Java HotSpot(TM) 64-Bit Server VM warning: ignoring option PermSize=50m; support was removed in 8.0

解决：JDK8中用metaspace代替permsize，于是将-XX:PermSize=10m;-XX:MaxPermSize=50m;修改为：-XX:MetaspaceSize=10m;- XX:MaxMetaspaceSize=50m;

# 实验过程中收获的经验、教训、感想

本节除了总结你在实验过程中收获的经验和教训，也可就以下方面谈谈你的感受（非必须）：

1. 了解了java doc的格式，知道了代码风格的重要性，掌握checksty和findbugs工具的基本用法。
2. 对java读写文件的策略有了更深入的了解，知道了他们各种的优劣。
3. 知道了jstat、jmap、VisualVM、MAT，但不知道如何分析利用工具获得的信息，所以不太能利用这些工具。
4. 诸如FindBugs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，我认为这些工具改善代码质量的作用有限，在我看来checkstyle的要求太苛刻，findbugs也只能找到一些基本的bug，可能一些和内部实现的bug它未能帮我们找到。
5. Java提供了这么多种I/O的实现方式，每种都有自己适合的应用场景，本次实验主要是读取文本，所以读取字节的I/O方式就不太适用，但在读取图片的应用中，读取字节会更加高效。所以在今后的学习中应该知道如何选择最优的I/O策略。
6. 基于在实验中的体会，我认为“通过配置JVM内存分配和GC参数来提高程序运行性能”有很多的意义，对heap、特别是New Generation的大小的设置对读写文件效率有很大的影响。