



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

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	张景润
学号	1172510217
班号	1703002
电子邮件	2584363094@qq.com
手机号码	18530272728

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	4
3.1 Static Program Analysis	4
3.1.1 人工代码走查 (walkthrough)	4
3.1.1.1 代码缩进问题 (2998 处)	6
3.1.1.2 本行字符数 X 个：一行字符数超过 100 个	6
3.1.1.3 javadoc 第一句缺少一个结束日期	7
3.1.1.4 Javadoc 缺少	7
3.1.1.5 名称 X 必须匹配表达式 'X'	7
3.1.1.6 导入语句 X 字典顺序错误	8
3.1.1.7 其他一些错误	9
3.1.1.8 最终结果 (解决所有规范问题)	10
3.1.2 使用 CheckStyle 和 SpotBugs 进行静态代码分析	10
3.2 Java I/O Optimization	11
3.2.1 新功能-持久化储存系统信息	11
3.2.2 多种 I/O 实现方式	14
3.2.2.1 实现三个 I/O 方式+具体实现方法	14
3.2.2.2 strategy 设计模式实现策略之间的切换	16
3.2.3 多种 I/O 实现方式的效率对比分析	16
3.2.3.1 收集 I/O 时间的方式	16
3.2.3.2 表格方式对比性能	17
3.2.3.3 图形方式对比性能	17
3.3 Java Memory Management and Garbage Collection (GC)	18
3.3.1 使用-verbose:gc 参数	18
3.3.2 用 jstat 命令行工具的-gc 和-gcutil 参数	20
3.3.3 使用 jmap -heap 命令行工具	20
3.3.4 使用 jmap -histo 命令行工具	21
3.3.5 使用 jmap -clstats 命令行工具	22
3.3.6 使用 jmap -permstat 命令行工具	22
3.3.7 使用 JMC/JFR、jconsole 或 VisualVM 工具 (我选择 VisualVM 工具)	23
3.3.8 分析垃圾回收过程	24
3.3.9 配置 JVM 参数并发现优化的参数配置	25

3.4 Dynamic Program Profiling	25
3.4.1 使用 VisualVM 进行 CPU Profiling	26
3.4.2 使用 VisualVM 进行 Memory profiling.....	29
3.5 Memory Dump Analysis and Performance Optimization	30
3.5.1 内存导出	30
3.5.2 使用 MAT 分析内存导出文件	31
3.5.3 发现热点/瓶颈并改进、改进前后的性能对比分析	35
3.5.4 在 MAT 内使用 OQL 查询内存导出	36
3.5.4.1 CircularOrbit 的所有对象实例	36
3.5.4.2 大于特定长度 n 的 String 对象：我选择长度为 10.....	37
3.5.4.3 大于特定大小的任意类型对象实例：我选择大小为 100 000.....	38
3.5.4.4 PhysicalObject（及其子类）对象实例的数量和总占用内存大小	38
3.5.4.5 所有包含元素数量大于 100 的 Collections 实例	39
3.5.4.6 我感兴趣的其他查询	40
3.5.5 观察 jstack 导出程序运行时的调用栈	41
3.5.6 使用设计模式进行代码性能优化	43
3.6 Git 仓库结构	44
4 实验进度记录	45
5 实验过程中遇到的困难与解决途径	46
6 实验过程中收获的经验、教训、感想	46
6.1 实验过程中收获的经验教训	46
6.2 针对以下方面的感受	46

1 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”。

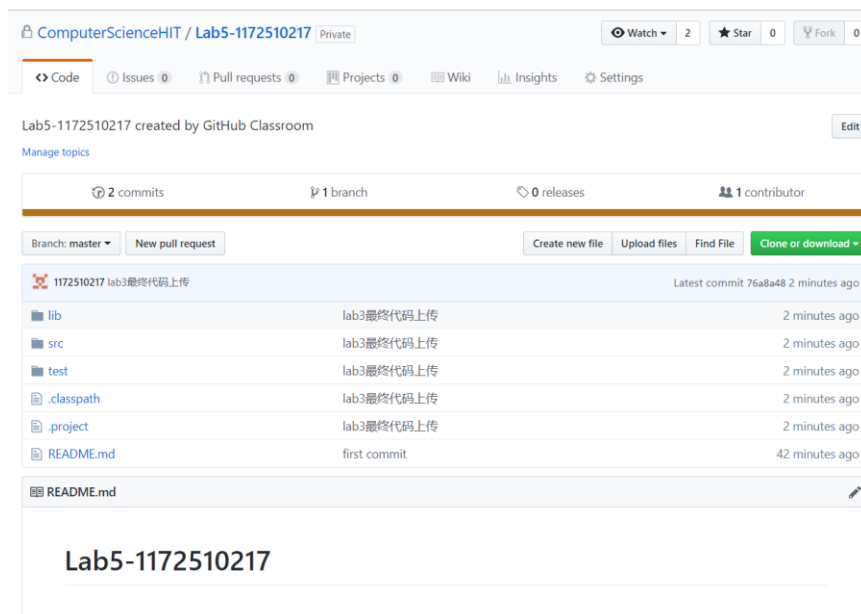
具体训练的技术包括：

- 静态代码分析（CheckStyle 和 SpotBugs）
- 动态代码分析（Java 命令行工具 jstat、jmap、jcmd、VisualVM、JMC、JConsole 等）
- JVM 内存管理与垃圾回收（GC）的优化配置
- 运行时内存导出(memory dump)及其分析（Java 命令行工具 jhat、MAT）
- 运行时调用栈及其分析（Java 命令行工具 jstack）；
- 高性能 I/O
- 基于设计模式的代码调优
- 代码重构

2 实验环境配置

- **配置建立本地仓库**

- 1, echo "# Lab5-1172510217" >> README.md
- 2, git init
- 3, git add README.md
- 4, git commit -m "first commit"
- 5, git remote add origin <https://github.com/ComputerScienceHIT/Lab5-1172510217.git>
- 6, git push -u origin master
- 7, 将 lab4 代码推送到远程仓库

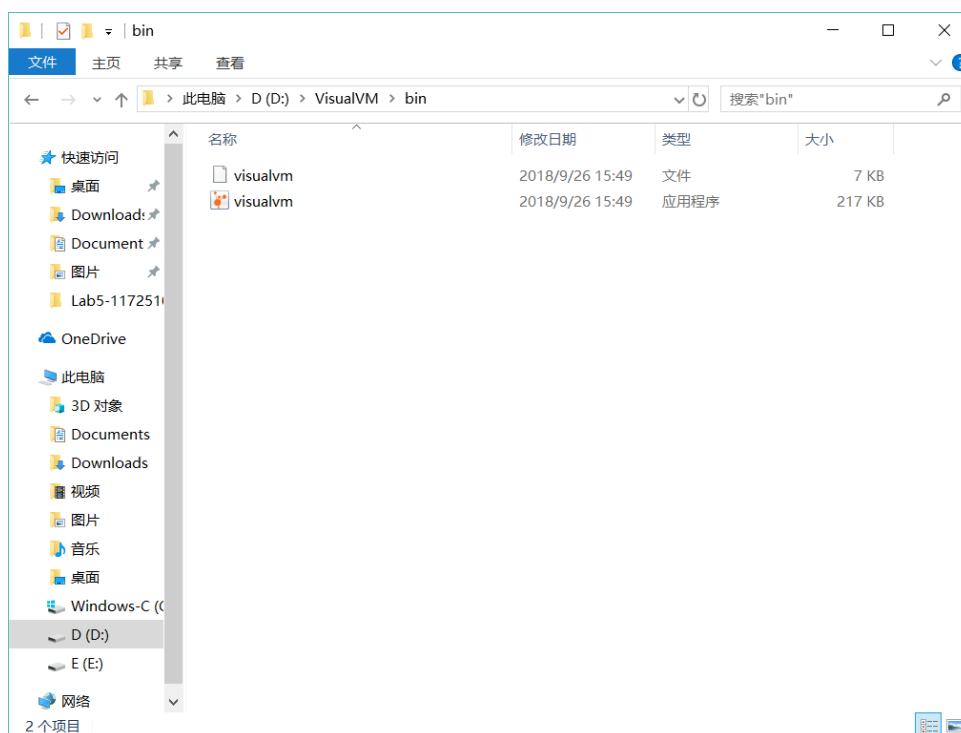


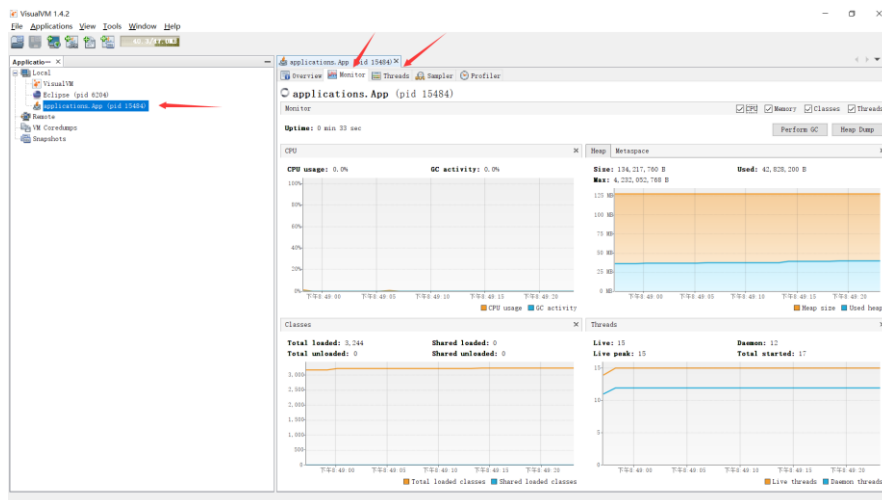
- 本地仓库地址

<https://github.com/ComputerScienceHIT/Lab5-1172510217>

- 配置 VisualVM

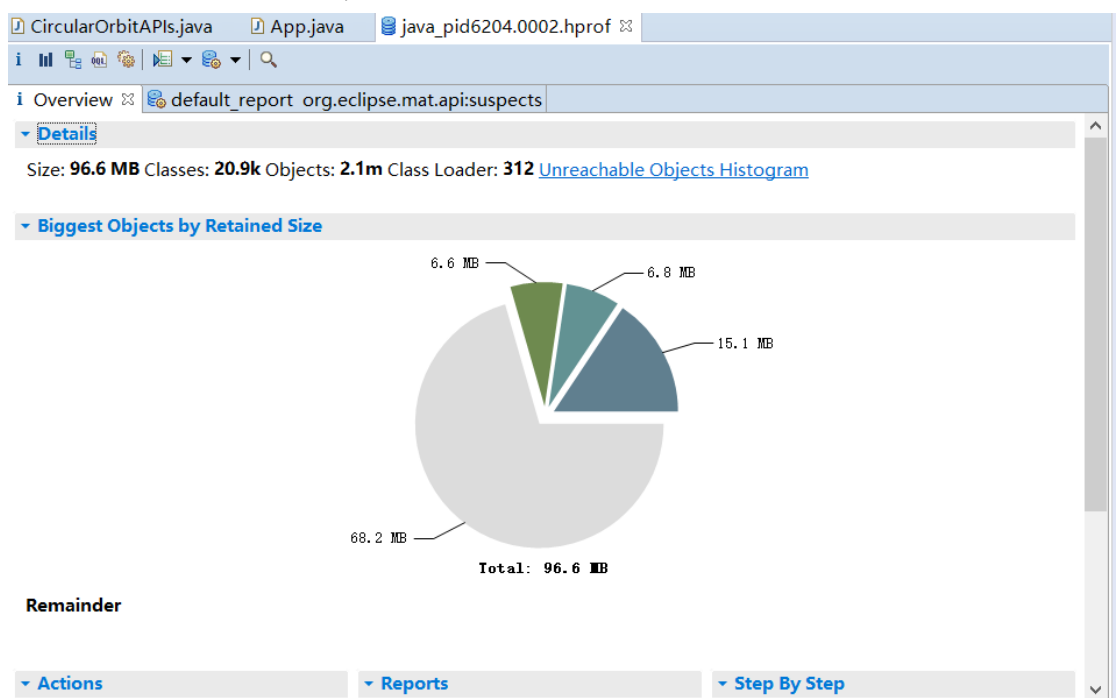
- 1, 从网站上下载最新版 VisualVM 压缩包
- 2, 解压缩此文件
- 3, 在 bin 文件中找到 visualvm 可执行程序文件, 如截图
- 4, 运行此程序, 然后运行 lab5 中的 GUI 主程序, 可以看到我们的 App 程序, 双击即可生成相应的动态详细信息。如截图。





● 配置 MemoryAnalyzer。

- 1, 在 Eclipse Market Place 中搜索 MemoryAnalyzer
- 2, 点击安装即可，安装完成如截图

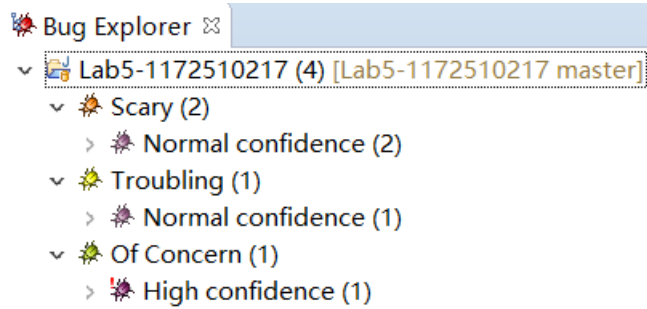


● 配置 CheckStyle。

- 1, 在 Eclipse Market Place 中搜索 CheckStyle
- 2, 点击安装即可。
- 3, 选择默认的 style: 有 Google checks 和 Sun Checks 以及默认的 Caesar 风格

Check Configuration	Location	Type	Def...
6.031 Caesar	https://raw.git...	Remote Confi...	
Google Checks	google_check...	Built-In Config...	✓
Sun Checks	sun_checks.xml	Built-In Config...	

● 配置 SpotBugs。已在 lab4 中配置完成。



3 实验过程

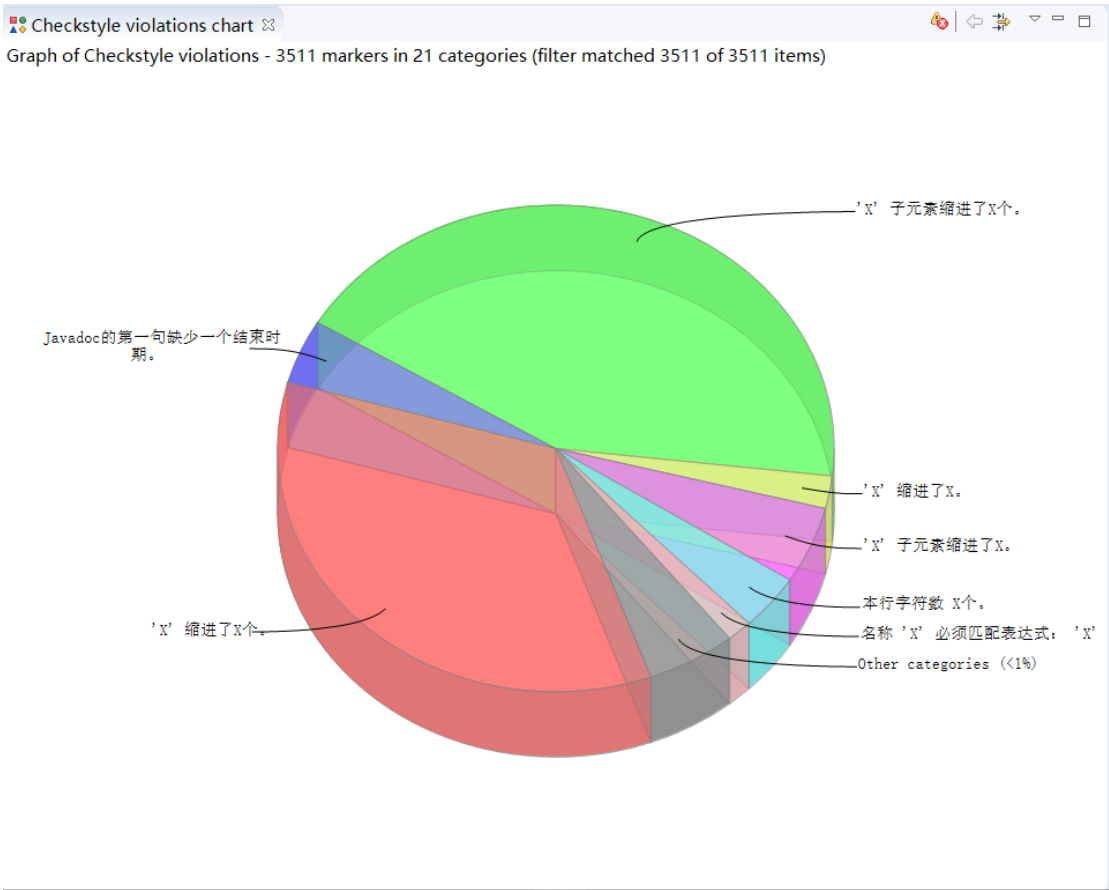
3.1 Static Program Analysis

3.1.1 人工代码走查 (walkthrough)

两点说明

- 我使用的是 Google 的代码规范进行 checkStyle 检测。
- 一共检测到了 3511 处不规范之处，运行情况如截图。
- 不规范比例分布如下表

不规范项	个数
总和	3511
代码缩进问题	2998
javadoc 第一句缺少一个结束日期	150
本行字符数 X 个：行字符超 100 个	93
Javadoc 缺少	54
名称 X 必须匹配表达式 'X'	52
导入语句 X 字典顺序错误	31
其他一些错误	84



Checkstyle violations

Overview of Checkstyle violations - 3511 markers in 21 categories (filter matched 3511 of 3511 items)

Checkstyle violation type	Occurrences
'X' 后应有空格。	19
'X' 结构必须使用大括号 '{'。	5
'X' 前应有空格。	7
'X' 前应有空行。	1
'X' 缩进了X。	76
'X' 缩进了X个。	1227
'X' 子元素缩进了X。	180
'X' 子元素缩进了X个。	1515
@标签应有非空说明。	9
Javadoc的第一句缺少一个结束时期。	150
本行字符数 X个。	129
变量'X'行)。若需要存储该变量的值, 请将其声明为fina...	17
不应使用 '*' 形式的导入 - X。	13
导入语句'X' 字典顺序错误。应在'X'之前。	31
顶级类 X 应位于它自己的源文件中。	1
空行后应有 <p> 标签。	4
每一个变量的定义必须在它的声明处, 且在同一行。	11
名称 'X' 必须匹配表达式: 'X'。	52
名称 'X' 中不能出现超过 'X' 个连续大写字母。	10
缺少 Javadoc。	23
缺少摘要javadoc。	31

3.1.1.1 代码缩进问题（2998 处）

1，注：代码缩进问题包括以下 4 个问题

- ‘X’ 缩进了 X；
- ‘X’ 缩进了 X 个；
- ‘X’ 子元素缩进了 X；
- ‘X’ 子元素缩进了 X 个；

⚠ 'X' 缩进了X。	76
⚠ 'X' 缩进了X个。	1227
⚠ 'X' 子元素缩进了X。	180
⚠ 'X' 子元素缩进了X个。	1515

2，解决办法

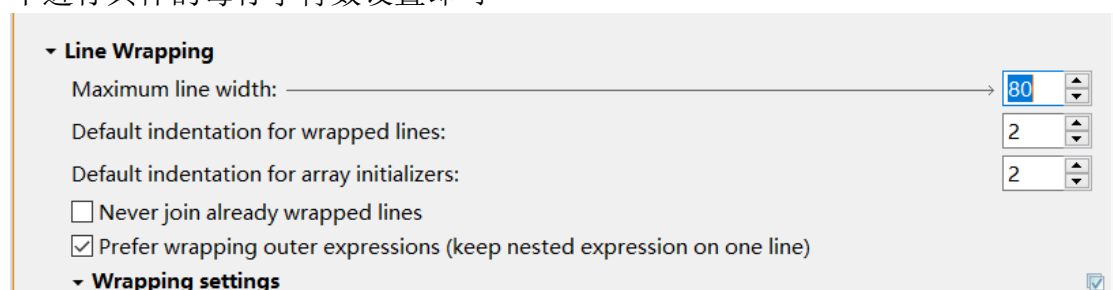
- 将所有的默认缩进换为 2 空格缩进。采用快捷键 CTRL+SHIFT+F 进行设置
- 将 tab 键换位 2 空格代替。采用快捷键 CTRL+SHIFT+F 进行设置

3，解决结果

- 风格提示处由 3511 变为 464。
- 风格提示窗口不再含有代码缩进问题。

3.1.1.2 本行字符数 X 个：一行字符数超过 100 个

- **解决办法：**在 eclipse 的 window->preference->codeStyle->Line Wrapping 中进行具体的每行字符数设置即可



- **解决结果：**
 - 1，规范风格提示变为 373 处
 - 2，风格提示窗口不再有行字符超过限制的问题
- **具体来看：**将下图图 1 所代表的一类问题的行长度进行默认格式修改，改为图 2 所示。

```
ImageIcon imageIcon = new ImageIcon("src/img/Stellar.jpg");
imageIcon.setImage(imageIcon.getImage().getScaledInstance(getWidth(), getHeight(), Image.SCALE_AREA_AVERAGING));
```

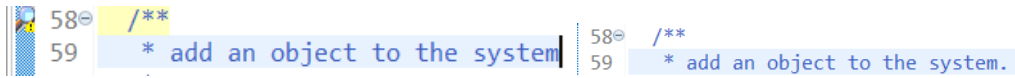
图 1

```
ImageIcon imageIcon = new ImageIcon("src/img/Stellar.jpg");
imageIcon.setImage(imageIcon.getImage().getScaledInstance(getWidth(),
getHeight(), Image.SCALE_AREA_AVERAGING));
```

图 2

3.1.1.3 javadoc 第一句缺少一个结束日期

- **解决办法及示例：**在注释第一行后面加上字符.即可。截图右边为已修改



- **解决结果**

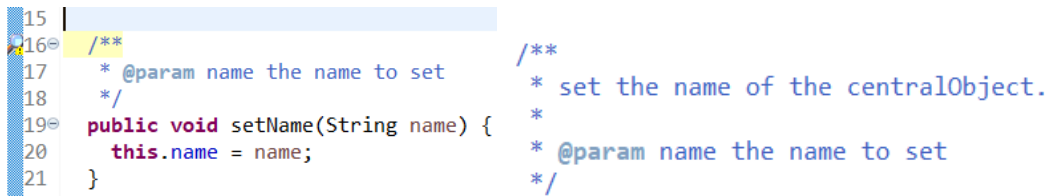
- 1, 代码风格提示减少了 150 处
- 2, 风格提示窗口不再含有 javadoc 第一句缺少结束日期的提示

3.1.1.4 Javadoc 缺少

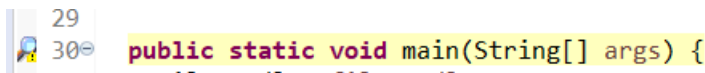
此问题包括：缺少摘要 javadoc+缺少 javadoc

- **解决办法及示例**

- 1, 针对缺少摘要 javadoc：为每一个 javadoc 添加摘要，修改后如截图



- 2, 针对缺少 javadoc：为每一个方法书写 javadoc，修改后如截图



- **解决结果**

- 1, 代码风格提示减少了 54 处
- 2, 风格提示窗口不再含有 javadoc 缺少的提示。

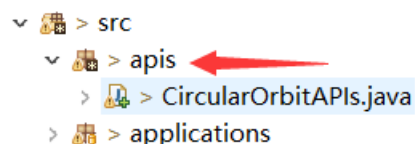
3.1.1.5 名称 X 必须匹配表达式 'X'

此问题包括：包名+方法+变量名+参数名+成员名 不规范

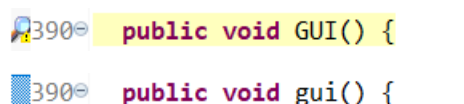
- **解决办法及示例**

- 1, 针对包名不规范：右键 refactor->rename 将所有的包名出现的地方全修改为小写字母格式。

如原来我有名为 APIS 的包，后来命名重构为 apis 即可，如截图



- 2, 针对方法名不规范：pattern：`^[a-z][a-z0-9][a-zA-Z0-9_]*$`。快捷键 Shift+Alt+R 对方法名进行重命名，如截图所示。



- 3, 针对变量名不规范：快捷键 Shift+Alt+R 进行局部变量重命名，如截图（将数组 A 重命名为 a 即可）

```
int[] a = null, b = { 1, 2, 3, 4 };
```

- 4, 针对参数名不规范：快捷键 Shift+Alt+R 进行方法参数重命名，如截图（将数组 A 重命名为 a 即可）

```
public double findMedianSortedArrays(int[] A, int[] B) {
```

```
public double findMedianSortedArrays(int[] a, int[] b) {
```

- 5, 针对成员名不规范：pattern: `^[a-z][a-z0-9][a-zA-Z0-9]*$`。快捷键 Shift+Alt+R 进行成员名重命名，如截图。

```
List<List<Vote>> A;
```

```
List<List<Vote>> alist;
```

● 解决结果

- 1, 代码规范提示地方变为 325 处
- 2, 代码规范窗口不再存在命名不符合规范的问题

3.1.1.6 导入语句 X 字典顺序错误

- 解决办法及示例：将顺序调整为正确的顺序即可，如截图

```
10 import org.junit.jupiter.api.Test;
11
12 import circularorbit.AtomStructure;
13 import myexception.FileChooseException;
```

图一：调整前

```
import static org.junit.Assert.assertFalse;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import circularorbit.AtomStructure;
import java.io.File;
import java.io.IOException;
import myexception.FileChooseException;
import org.junit.jupiter.api.Test;
import physicalobject.PhysicalObject;
```

图二：调整后

● 解决结果：

- 1, 代码规范提示地方变为 288 处
- 2, 代码规范窗口不再存在导入顺序不合规的问题

3.1.1.7 其他一些错误

- **while、if、else 结构必须使用大括号**：结构加上大括号即可

```

58 while (alist.size() <= c) {
59     alist.add(new ArrayList<Vote>());
60 }

69 if (alist.get(mi).get(0).time <= t) {
70     lo = mi + 1; // 此处修改：二分法小的值加1
71 } else {
72     hi = mi;
73 }
74 }

```

- **静态导入应该与其余的导入组分隔一个空行**：加上一个空行即可

```

3 import static org.junit.jupiter.api.Assertions.*;
4 import org.junit.jupiter.api.Test;

```

- **Javadoc 注释空行后应有 <p> 标签**：执行如下修改（右图为修改后）
格式化操作可以执行：window-preference-codestyle-javadocs-取消勾选
Format HTML tags，然后添加标签<p>并对代码执行：source-format 即可

<pre> 11 * The output would be 2.0 12 * 13 * Example 2: </pre>	<pre> 11 * The output would be 2.0 12 * 13 *<p>Example 2: </pre>
--	--

- **名称中不能出现超过 2 个连续大写字母**：快捷键重命名即可（右图为修改后）

```

37 public class StellarSystemGUI { public class StellarSystemGui {

```

- **不应使.*形式的导入**：修改导入为具体的一个个的导入（下面的图为修改后）

<pre> 2 3 import static org.junit.jupiter.api.Assertions.*; 4 </pre>	<pre> 2 3 import static org.junit.Assert.assertEquals; 4 </pre>
--	---

- **每个变量定义必须在声明处，且在同一行**：将代码修改为一行一个变量声明

<pre> 82 String group1 = matcher.group(1), group2 = matcher.group(2), 83 group3 = matcher.group(3), group4 = matcher.group(4); 84 String group5 = matcher.group(5), group6 = matcher.group(6), 85 group7 = matcher.group(7), group8 = matcher.group(8); </pre>	<pre> 82 String group1 = matcher.group(1); 83 String group2 = matcher.group(2); 84 String group3 = matcher.group(3); </pre>
--	---

- **大括号与后面注释间应有空格**：注释前大括号后添加空格（修改后为下图右）

```

80 while (i < line.length()) { // 数组越界 while (i < line.length()) { // 数组越界

```

- **变量声明及其使用之间不超过三行**：变量在使用时的前一行定义（两张图片下面的一张为修改后）

```

56 RemoveComments removeComments = new RemoveComments();

```

```

29 RemoveComments removeComments = new RemoveComments();
30 assertEquals(res, removeComments.removeComments(sources));

```

- @标签后应有非空说明：对每一个@标签添加说明（修改后如较下面的图片）

```

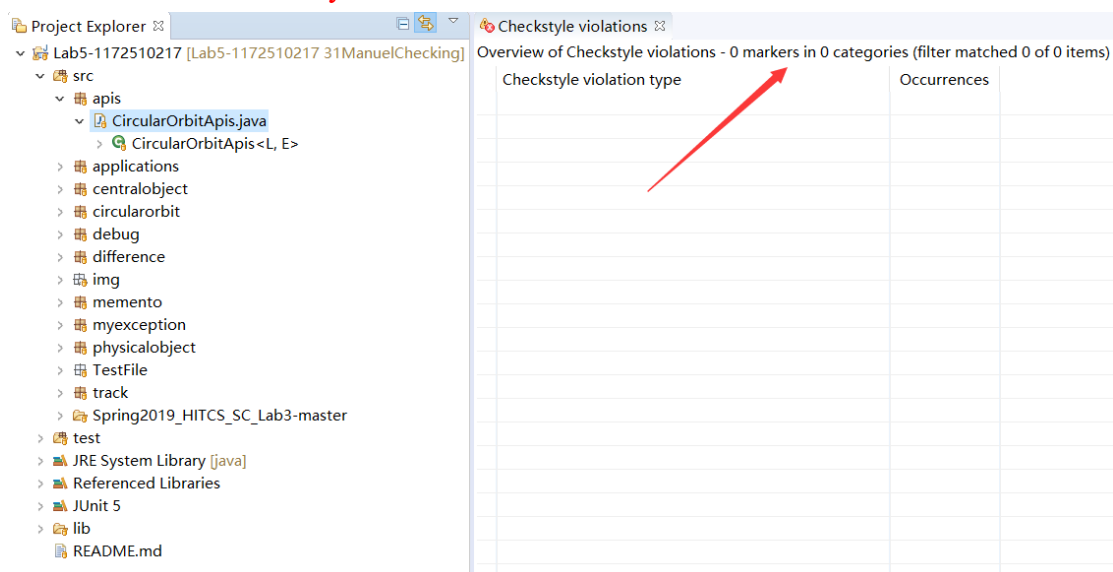
33 /**
34  * @throws FileChooseException
35  * @throws IOException
36  */

34  * @throws FileChooseException 抛出的自定义的文件选取异常
35  * @throws IOException 抛出的IO异常

```

3.1.1.8 最终结果（解决所有规范问题）

- 下图为最终 checkstyle 截图



- 合并分支+推送分支
 - 1, 在 31ManuelChecking 分支输入指令切回主分支: `git checkout master`
 - 2, 合并本地分支: `git merge 31ManuelChecking`
 - 3, 推送分支: `git push`

3.1.2 使用 CheckStyle 和 SpotBugs 进行静态代码分析

- 关于 CheckStyle

代码风格已在上文 3.1.1 进行了详细的介绍和讲解。

- 关于 SpotBugs

- 1, 取余对负数不成立：应改为 $(m + n) \% 2 \neq 1$ （修改后为右图）

```

92 if ((m + n) % 2 == 1) {
93     return minRight;
94 }

if ((m + n) % 2 != 0) {
    return minRight;
}

```

- 2, 对象判断不可用 `==`：应用 `equals` 方法（修改后如右图）

```

246         / (2 * Math.PI * planet.getTrackRadius());
247         if (planet.getRevolutionDiretion() == "CCW") {
247         if (planet.getRevolutionDiretion().equals("CCW")) {

```

3, 不使用 `Double.compareTo` 方法: 使用 `Double` 的 `compare` 方法 (下图)

```

106 @Override public int compare(Track<E> track1, Track<E> track2) {
107     return Double.compare(track1.getRadius(), track2.getRadius());

```

4, 不可以使用直接 `new String()`: 改为 `String string = ""` (如右图)

解决了部分潜在的 bug (约 10 个!!!)

```

40 String lineString = new String(); String string = "";

```

5, 不建议使用 `System.exit`: 删除掉使用其他策略

```

66 System.exit(0);

```

6, 分配一个对象, 只是为了得到它的类: 改用类名 `.class` 获取类 (如下图)

解决了大部分潜在的 bug (约 40 个!!!)

```

55     } catch (Exception e) {
56         assertEquals(e.getClass(), new FileChooseException().getClass());
57     }
55     } catch (Exception e) {
56         assertEquals(e.getClass(), FileChooseException.class);
57     }

```

7, `String` 字符串连接最好用 `StringBuilder`: 使用 `StringBuilder` 即可 (如下图)

```

48 string += lineString + "\n";
49 stringBuilder.append(lineString);
50 stringBuilder.append("\n");

```

● 分支合并+分支推送

1, 在 31ToolChecking 分支输入指令切回主分支: `git checkout master`

2, 合并本地分支: `git merge 31ToolChecking`

3, 推送分支: `git push`

3.2 Java I/O Optimization

3.2.1 新功能-持久化储存系统信息

● 在 `circularorbit` 包中添加新功能: 主要是遍历

1, 在 `StellarSystem` 中添加新方法: `saveSystemInfoInFile` 和 `enotationTransform`

1) 其中 `enotationTransform`: 将编译器存储的科学记数法表示的数字转化为语法要求的格式;

```

/**
 * note:将数值表示成科学记数法的形式.
 *
 * @param number 要转换形式的数值
 * @return 返回科学记数法正确格式的字符串形式
 */
private static String enotationTransform(double number) {
    if (number < 10000) {
        return "" + number;
    } else {
        double res = number;
        int count = 0;
        while (res >= 10) {
            res /= 10;
            count++;
        }
        return res + "e" + count;
    }
}

```

2) saveSystemInfoInFile: 将系统按照 lab3 语法格式输出。

```

public void saveSystemInfoInFile(String filePath) throws IOException {
    StringBuilder stringBuilder = new StringBuilder();
    System.out.println("开始构建字符串");
    stringBuilder.append("Stellar ::= <" + getCentralPoint().getName() + ","
        + enotationTransform(getCentralPoint().getRadius()) + ","
        + enotationTransform(getCentralPoint().getMess()) + ">\n"); // 恒星行
    // 下面书写行星行
    Iterator<StellarSystemObject> iterator = new PhysicalObjectItertor();
    while (iterator.hasNext()) {
        StellarSystemObject planet = iterator.next();
        stringBuilder.append("Planet ::= <" + planet.getPlanetName() + ","
            + planet.getPlanetState() + "," + planet.getPlanetColor() + ","
            + enotationTransform(planet.getPlanetRadius()) + ","
            + enotationTransform(planet.getTrackRadius()) + ","
            + enotationTransform(planet.getRevolutionSpeed()) + ","
            + planet.getRevolutionDiretion() + "," + planet.getAngle() + ">\n");
    }
    System.out.println("开始写入文件! ");
    FileWriter fileWriter = new FileWriter(filePath);
    BufferedWriter bfWriter = new BufferedWriter(fileWriter);
    bfWriter.write(stringBuilder.toString());
    bfWriter.close();
}

```

2, 在 AtomStructure 添加新方法: saveSystemInfoInFile

saveSystemInfoInFile: 将系统按照 lab3 语法格式输出。


```

public void saveSystemInfoInFile(String filePath) throws IOException {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("ElementName ::= " + getCentralPoint().getName()); // 元素第一行
    stringBuilder.append("\n" + "NumberOfTracks ::= " + getTracksNumber()); // 元素第二行
    stringBuilder.append("\nNumberOfElectron ::= "); // 元素第三行
    for (int i = 1; i <= getTracksNumber(); i++) {
        stringBuilder.append(i + "/" + getTrackObjectsNumber(i));
        if (i < getTracksNumber()) {
            stringBuilder.append(";");
        }
    }
    FileWriter fileWriter = new FileWriter(filePath);
    BufferedWriter bwWriter = new BufferedWriter(fileWriter);
    bwWriter.write(stringBuilder.toString());
    bwWriter.close();
}

```

3, 在 SocialNetworkCircle 添加新方法: saveSystemInfoInFile

saveSystemInfoInFile: 将系统按照 lab3 语法格式输出。

```

public void saveSystemInfoInFile(String filePath) throws IOException {
    StringBuilder stringBuilder = new StringBuilder();
    // 中心用户行
    stringBuilder.append("CentralUser ::= <" + getCentralPoint().getFriendName()
        + "," + getCentralPoint().getAge() + "," + getCentralPoint().getSex()
        + ">");
    // 所有的朋友用户行
    for (Friend friend : allFriends) {
        if (!friend.equals(getCentralPoint())) {
            stringBuilder.append("\nFriend ::= <" + friend.getFriendName() + ","
                + friend.getAge() + "," + friend.getSex() + ">");
        }
    }
    // 所有亲密度关系行
    int size = allFriends.size();
    for (int i = 0; i < size; i++) {
        for (int j = i; j < size; j++) {
            double intimacy = allFriends.get(i).getSocialTie(allFriends.get(j));
            if (intimacy != 0) {
                stringBuilder.append(
                    "\nSocialTie ::= <" + allFriends.get(i).getFriendName() + ","
                        + allFriends.get(j).getFriendName() + "," + intimacy + ">");
            }
        }
    }
    FileWriter fileWriter = new FileWriter(filePath);
    BufferedWriter bwWriter = new BufferedWriter(fileWriter);
    bwWriter.write(stringBuilder.toString());
    bwWriter.close();
}

```

● 分支合并+分支推送

- 1, 在 32AddOutput 分支输入指令切回主分支: git checkout master
- 2, 合并本地分支: git merge 32AddOutput
- 3, 推送分支: git push

3.2.2 多种 I/O 实现方式

3.2.2.1 实现三个 I/O 方式+具体实现方法

- **实现方法**

1, 设计了一个抽象类 `public interface IoStrategy {`, 其中有 6 个方法, 分别对应 3 个系统的读和写。

```
public void readFileAndCreateSystem(StellarSystem stellarSystem, File file)
public void readFileAndCreateSystem(AtomStructure atomStructure, File file)
public void readFileAndCreateSystem(SocialNetworkCircle socialNetworkCircle, File file)
public void saveSystemInfoInFile(StellarSystem stellarSystem, String filePath)
public void saveSystemInfoInFile(AtomStructure atomStructure, String filePath)
public void saveSystemInfoInFile(SocialNetworkCircle socialNetworkCircle, String filePath)
```

2, 设计了 3 个具体的 IO 策略, 实现该抽象类

```
public class NioStrategy implements IoStrategy
public class BufferedIoStrategy implements IoStrategy
public class ScannerIoStrategy implements IoStrategy
```

3, 在 `CircularOrbit` 的 3 个子类中各实现读和写的方法, 传入参数为 `IoStrategy`, 如截图。

```
public void readFileAndCreateSystem(IoStrategy ioStrategy)
    throws IOException, FileChooseException {
    this.initSystem();
    ioStrategy.readFileAndCreateSystem(this, readFile);
}

public void saveSystemInfoInFile(IoStrategy ioStrategy) throws IOException {
    ioStrategy.saveSystemInfoInFile(this, writeFilePath);
}
```

4, 客户端具体的调用方法如截图

```
StellarSystem stellarSystem = new StellarSystem();
stellarSystem.setReadFile(
    new File("src/Spring2019_HITCS_SC_Lab5-master/StellarSystem.txt"));
stellarSystem.setWriteFilePath("src/outputFile/StellarSystem.txt");

stellarSystem.readFileAndCreateSystem(new NioStrategy());
stellarSystem.saveSystemInfoInFile(new NioStrategy());
```

行星系统选择 `Nio` 的文件读取写入方式

- **BufferedIoStrategy**

此方法是前面实验中一直在使用的方法。读取思想是：按行读取处理数据。典型的操作如截图。

基本写入思想是：使用 `StringBuilder` 的 `append` 方法，构建完成输出字符串后，整个的输出。

```
InputStreamReader reader =
    new InputStreamReader(new FileInputStream(file));
BufferedReader bfReader = new BufferedReader(reader);
while ((lineString = bfReader.readLine()) != null) {
    lineCount++;
}

public void saveSystemInfoInFile(AtomStructure atomStructure, String filePath)
    throws IOException {
    StringBuilder stringBuilder = new StringBuilder("ElementName ::= ");
    stringBuilder.append(atomStructure.getCentralPoint().getName()); // 元素第一行
    stringBuilder.append("\nNumberOfTracks ::= ")
        .append(atomStructure.getTracksNumber()); // 元素第二行
    stringBuilder.append("\nNumberOfElectron ::= "); // 元素第三行
    for (int i = 1; i <= atomStructure.getTracksNumber(); i++) {
        stringBuilder.append(i).append("/")
            .append(atomStructure.getTrackObjectsNumber(i));
        if (i < atomStructure.getTracksNumber()) {
            stringBuilder.append(";");
        }
    }
    FileWriter fileWriter = new FileWriter(filePath);
    BufferedWriter bfWriter = new BufferedWriter(fileWriter);
    bfWriter.write(stringBuilder.toString());
    bfWriter.close();
}
```

● ScannerIoStrategy

基本读取思想是：按行读取数据+按行处理数据，典型操作如截图。

基本写入思想是：使用 `StringBuilder` 的 `append` 方法，构建完成输出字符串后，整个的输出。

```
Scanner scanner = new Scanner(file);
long startTime = System.currentTimeMillis();
while (scanner.hasNextLine()) {
    lineString = scanner.nextLine();

    StringBuilder stringBuilder = new StringBuilder("Stellar ::= <");
    stringBuilder.append(stellarSystem.getCentralPoint().getName()).append(",")
        .append(stellarSystem
            .enotationTransform(stellarSystem.getCentralPoint().getRadius()))
        .append(",")
        .append(stellarSystem
            .enotationTransform(stellarSystem.getCentralPoint().getMess()))
        .append(">\n"); // 恒星行
}
```

● NioStrategy

基本读取思想：按行读取处理数据。这个按行读取数据的方法与上述两个读取方式有差异，因为是字节读取，所以需要判断换行符的存在以及文本结尾存在。

基本写入思想：根据 `buffer` 可用容量进行输出，当构建的 `StringBuilder` 的字符串字节长度即将大于容量时，进行写出。

```

FileInputStream fileInputStream = new FileInputStream(file);
FileChannel fileChannel = fileInputStream.getChannel();
ByteBuffer buffer = ByteBuffer.allocate(1024 * 1024);
int bytesRead = fileChannel.read(buffer);
ByteBuffer stringBuffer = ByteBuffer.allocate(80);
while (bytesRead != -1) {
    buffer.flip();
    while (buffer.hasRemaining()) {
        byte b = buffer.get();
        if (b == 13 || b == 10) {
            stringBuffer.flip();
            lineString =
                Charset.forName("utf-8").decode(stringBuffer).toString();
            lineCount++;
        }
        if (stringBuilder.length() > 1024 * 1024 - 100) {
            buffer.put(stringBuilder.toString().getBytes());
            buffer.flip();
            startTime = System.currentTimeMillis();
            fileChannel.write(buffer);
            endTime = System.currentTimeMillis();
            sumTime += endTime - startTime;
            buffer.clear();
            stringBuilder = new StringBuilder();
        }
    }
}

```

3.2.2.2 strategy 设计模式实现策略之间的切换

客户端通过传入具体的策略即可实现不同策略的切换, 如截图

```

StellarSystem stellarSystem = new StellarSystem();
stellarSystem.setReadFile(
    new File("src/Spring2019_HITCS_SC_Lab5-master/StellarSystem.txt"));
stellarSystem.setWriteFilePath("src/outputFile/StellarSystem.txt");

stellarSystem.readFileAndCreateSystem(new NioStrategy());
stellarSystem.saveSystemInfoInFile(new NioStrategy());

```

3.2.3 多种 I/O 实现方式的效率对比分析

3.2.3.1 收集 I/O 时间的方式

- 通过注入少量代码的方式收集时间: 注入终端输出代码通过计算时间差得到读写的近似时间, 如截图.

```

long startTime = System.currentTimeMillis();
long endTime = System.currentTimeMillis();
System.out.println("行星系统BufferedIO读文件:" + (endTime - startTime) + "ms");

```

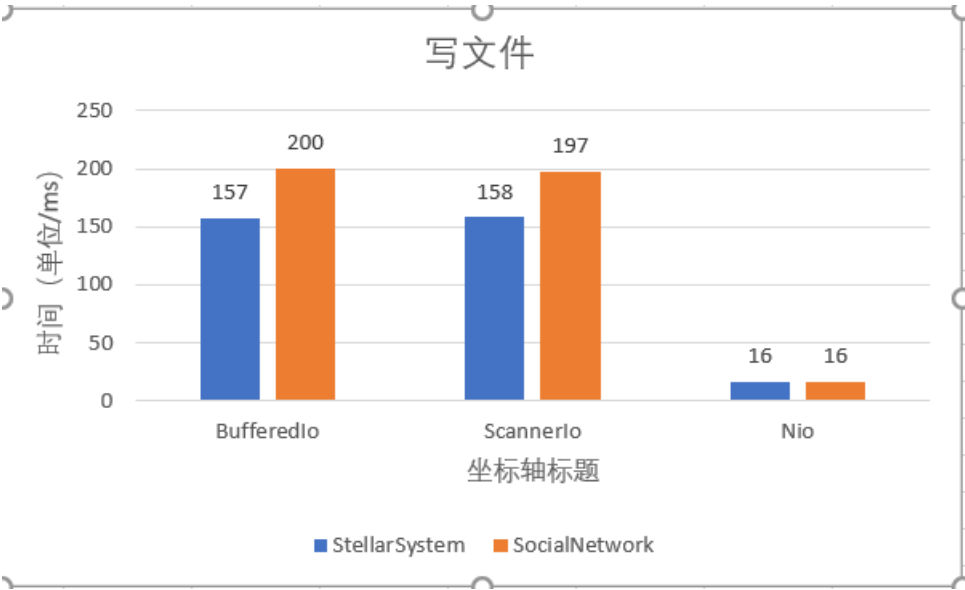
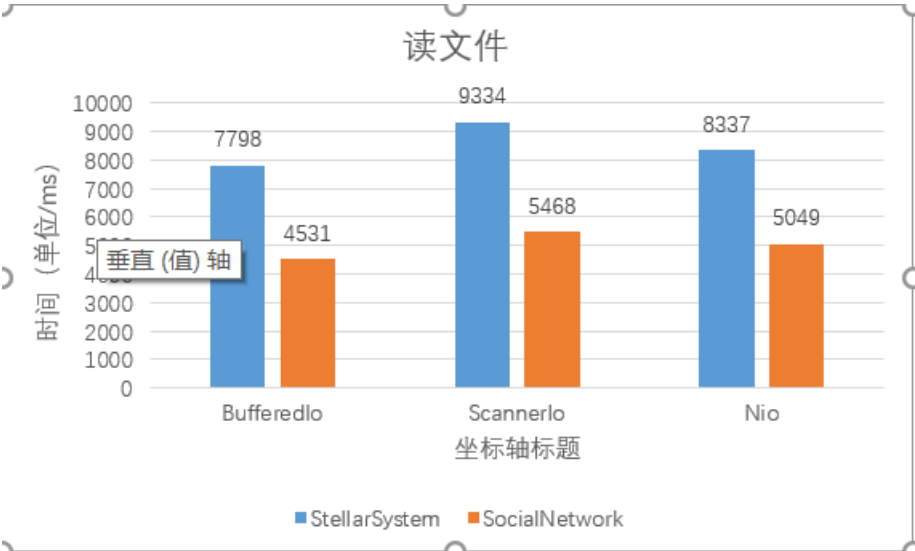
- 为了保证数据的准确性, 进行了多次实验, 选取其中数据波动不太明显的三组作为参考组
- 具体的执行结果如截图.

3.2.3.2 表格方式对比性能

		StellarSystem.txt	SocialNetworkCircle.txt
BufferedIoStrategy	读文件	7798ms	4531ms
	写文件	157ms	200ms
ScannerIoStrategy	读文件	9334ms	5468ms
	写文件	158ms	197ms
NioStrategy	读文件	8337ms	5049ms
	写文件	16ms	16ms

3.2.3.3 图形方式对比性能

总体来说:读取文件由于是按行处理,所以说三者差别不大;但是写出文件可以明显地看到 Nio 策略具有明显的优势.



3.3 Java Memory Management and Garbage Collection (GC)

常用：在控制台输入 `jps` 可以获取进程 `pid`，便于下面一系列命令的输入

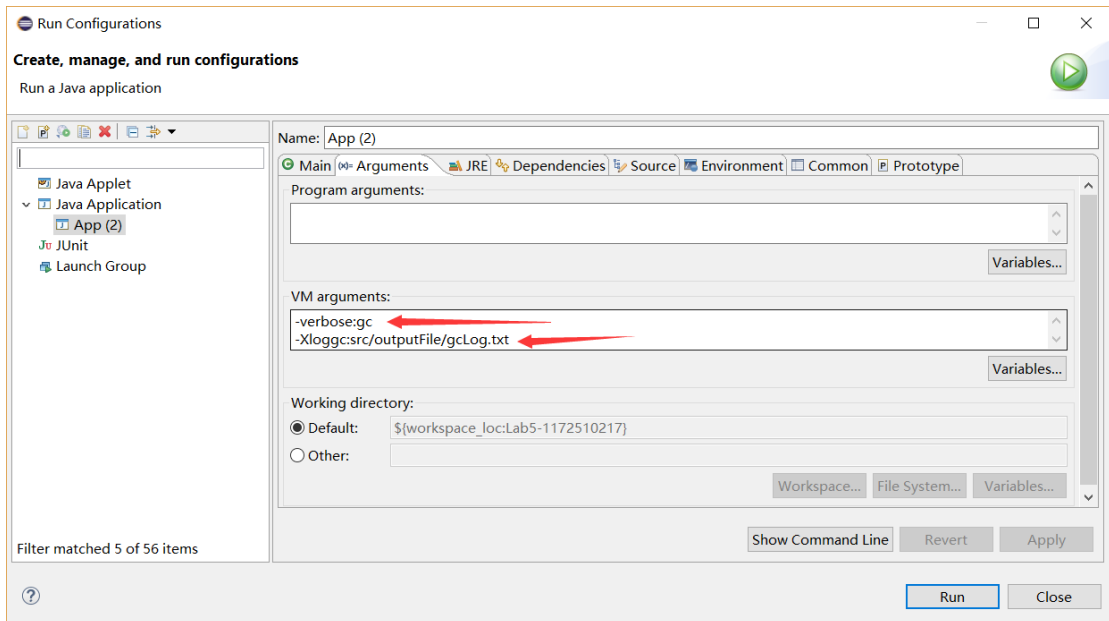
3.3.1 使用 `-verbose:gc` 参数

- 设置相关参数

1, `-verbose:gc` 参数

2, `-Xloggc:src/outputFile/gcLog.txt` 参数

3, `-XX:+PrintGCDetails` 参数



- 分析输出 `gc` 文本结果

1, Minor GC 和 Full GC 发生的频率

Minor GC 在 8.2s 的时间内发生了 17 次，大约 2 次/s；

Full GC 大于 10s 进行一次，大约 0.1 次/s。

```
1.027: [GC (Allocation Failure)
2.174: [GC (Allocation Failure)
3.894: [GC (Allocation Failure)
4.037: [GC (Allocation Failure)
4.200: [GC (Allocation Failure)
4.372: [GC (Allocation Failure)
4.542: [GC (Allocation Failure)675.417: [GC (Allocation Failure)
4.724: [GC (Allocation Failure)677.013: [GC (Allocation Failure)
4.937: [GC (Allocation Failure)677.126: [Full GC (Ergonomics) [P
5.200: [GC (Allocation Failure)681.764: [GC (Allocation Failure)
5.414: [GC (Allocation Failure)683.376: [GC (Allocation Failure)
5.870: [GC (Allocation Failure)685.111: [GC (Allocation Failure)
6.397: [GC (Allocation Failure)686.663: [GC (Allocation Failure)
7.073: [GC (Allocation Failure)687.877: [GC (Allocation Failure)
7.715: [GC (Allocation Failure)687.960: [Full GC (Ergonomics) [P
8.506: [GC (Allocation Failure)
9.290: [GC (Allocation Failure)
```

2, 两种 GC 单次耗费的时间

Minor GC 单次耗费时间不唯一，从 0.01s 到 0.08s 不等。如截图

Full GC 单词耗费时间也不唯一，第一次耗费时间较少为 0.02s，第二次为

0.45s，第三次为 0.96s，如截图

```
[Times: user=0.13 sys=0.00, real=0.02 secs]
[Times: user=0.00 sys=0.02, real=0.01 secs]
[Times: user=0.11 sys=0.01, real=0.03 secs]
[Times: user=0.09 sys=0.02, real=0.02 secs]
[Times: user=0.13 sys=0.00, real=0.02 secs]
[Times: user=0.13 sys=0.01, real=0.02 secs]
[Times: user=0.08 sys=0.02, real=0.03 secs]
[Times: user=0.28 sys=0.08, real=0.05 secs]
[Times: user=0.27 sys=0.08, real=0.05 secs]
```

Full GC

```
[Times: user=0.02 sys=0.00, real=0.02 secs]

[Times: user=2.86 sys=0.06, real=0.45 secs]
[Times: user=5.09 sys=0.06, real=0.96 secs]
```

3, 每次 GC 前后 heap 中各区域占用情况的变化

任取一次 Minor GC，分析其堆中各区域变化情况（下面加深色的为源数据）

```
[PSYoungGen: 64483K->2021K(67072K)] 105303K->44811K(186880K),
0.0027480 secs]
```

年轻代内存使用由 64483K 降低到 2021K，总空间大小为 67072K；

整个堆内存的使用由 105303K 降低到 44811K，总空间大小为 186880K

GC 前	年轻代已用	年轻代未用	老年代已用	老年代未用
单位 K	64483	2589	40820	78988
GC 后	年轻代已用	年轻代未用	老年代已用	老年代未用
单位 K	2021	65051	42790	77018

任取一次 Full GC，分析其堆中各区域的变化情况（下面加深色的为源数据）

```
[PSYoungGen:224K->0K(579584K)]
[ParOldGen:294055K->293506K(487424K)]
294279K->293506K(1067008K),
[Metaspace: 13462K->13462K(1060864K)], 1.0579956 secs]
```

年轻代内存使用由 224K 降到 0K，总空间大小为 579584K；

老年代内存使用由 294055K 降到 293506K，总空间大小为 487424K；

堆总内存由 294279K 降到 293506K，总空间大小为 1067008K；

Metaspace 内存使用不变为 13462K，总空间大小为 1060864K。

GC 前	年轻代已用	年轻代未用	老年代已用	老年代未用	Metaspace已用	Metaspace未用
单位 K	224	579360	294055	193369	13462K	13462K
GC 后	年轻代已用	年轻代未用	老年代已用	老年代未用	Metaspace已用	Metaspace未用
单位 K	0	579584	293506	193918	1047402	1047402

3.3.2 用 jstat 命令行工具的-gc 和-gcutil 参数

- 使用工具的方法: `jstat -gc 进程 pid 监控周期(ms)`

我的输入是：`jstat -gc 22376 10000`（每隔 10s 输出一次 gc 情况）

[illegible]

其中各部分代表的含义如下

S0C: 第一个幸存区的大小, S1C: 第二个幸存区的大小

S0U: 第一个幸存区的使用大小, S1U: 第二个幸存区的使用大小

EC: 伊甸园区的大小, EU: 伊甸园区的使用大小

0C: 老年代大小, 0U: 老年代使用大小

MC: 方法区大小, MU: 方法区使用大小

CCSC:压缩类空间大小, CCSU:压缩类空间使用大小

YGC: 年轻代垃圾回收次数, YGCT: 年轻代垃圾回收消耗时间

FGC: 老年代垃圾回收次数, FGCT: 老年代垃圾回收消耗时间

GCT: 垃圾回收消耗总时间

- 使用工具的方法: `jstat -gcutil 进程 pid 监控周期(ms)`

我的输入是：`jstat -gcutil 22376 10000`（每隔 10s 输出一次 gc 情况）

```
C:\Users\86185>jstat -gcutil 22376 3000
```

S0	S1	E	O	M	CCS	YGC	YGCT	FGC	FGCT	GCT
0.00	99.90	48.32	73.86	96.48	93.70	25	0.611	3	1.341	1.952
0.00	99.90	48.32	73.86	96.48	93.70	25	0.611	3	1.341	1.952

3.3.3 使用 jmap -heap 命令行工具

- 使用工具的方法: jmap -heap 进程 pid

我的输入是: `jump -heap 11052`

```
C:\Users\86185>jmap -heap 11052
Attaching to process ID 11052, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.202-b08

using thread-local object allocation.
Parallel GC with 8 thread(s)

Heap Configuration:
  MinHeapFreeRatio          = 0
  MaxHeapFreeRatio          = 100
  MaxHeapSize                = 2116026368 (2018.0MB)
  NewSize                    = 44564480 (42.5MB)
  MaxNewSize                 = 705167360 (672.5MB)
  OldSize                    = 89653248 (85.5MB)
  NewRatio                   = 2
  SurvivorRatio              = 8
  MetaspaceSize              = 21807104 (20.796875MB)
  CompressedClassSpaceSize   = 1073741824 (1024.0MB)
  MaxMetaspaceSize           = 17592186044415 MB
  G1HeapRegionSize           = 0 (0.0MB)

Heap Usage:
PS Young Generation
Eden Space:
  capacity = 606601216 (578.5MB)
  used     = 143736272 (137.0775909423828MB)
  free     = 462864944 (441.4224090576172MB)
  23.69534847750783% used
From Space:
  capacity = 49283072 (47.0MB)
  used     = 49250336 (46.968780517578125MB)
  free     = 32736 (0.031219482421875MB)
  99.93357556931515% used
To Space:
  capacity = 49283072 (47.0MB)
  used     = 0 (0.0MB)
  free     = 49283072 (47.0MB)
  0.0% used
PS Old Generation
  capacity = 347078656 (331.0MB)
  used     = 251549784 (239.89656829833984MB)
  free     = 95528872 (91.10343170166016MB)
  72.47630462185494% used

7252 interned Strings occupying 589312 bytes.
```

3.3.4 使用 jmap -histo 命令行工具

- 使用工具的方法：jmap -histo 进程 pid
我的输入是：jmap -histo 11052


```
C:\Users\86185>jmap -histo 11052
```

num	#instances	#bytes	class name
1:	707774	76440576	[F
2:	640160	57521760	[Ljava.util.HashMap\$Node;
3:	1603013	51296416	java.util.HashMap\$Node
4:	1610384	47328712	[C
5:	1609967	38639208	java.lang.String
6:	640184	30728832	java.util.HashMap
7:	8308	26489240	[I
8:	320000	20480000	physicalobject.StellarSystemObject
9:	353880	19817280	java.awt.geom.EllipseIterator
10:	353880	16986240	java.awt.geom.Ellipse2D\$Double
11:	354444	14319496	[B
12:	817394	13078304	java.lang.Integer
13:	353880	11324160	java.awt.geom.Path2D\$Float
14:	320000	10240000	track.Track
15:	320001	7680024	java.lang.Double
16:	1959	1427912	[Ljava.lang.Object;
17:	2416	276664	java.lang.Class
18:	776	55872	java.lang.reflect.Field
19:	1196	38272	java.util.Hashtable\$Entry
20:	407	35816	java.lang.reflect.Method
21:	837	33480	java.util.TreeMap\$Entry
22:	112	28608	[J
23:	856	27392	java.util.concurrent.ConcurrentHashMap\$Node

3.3.5 使用 jmap -clstats 命令行工具

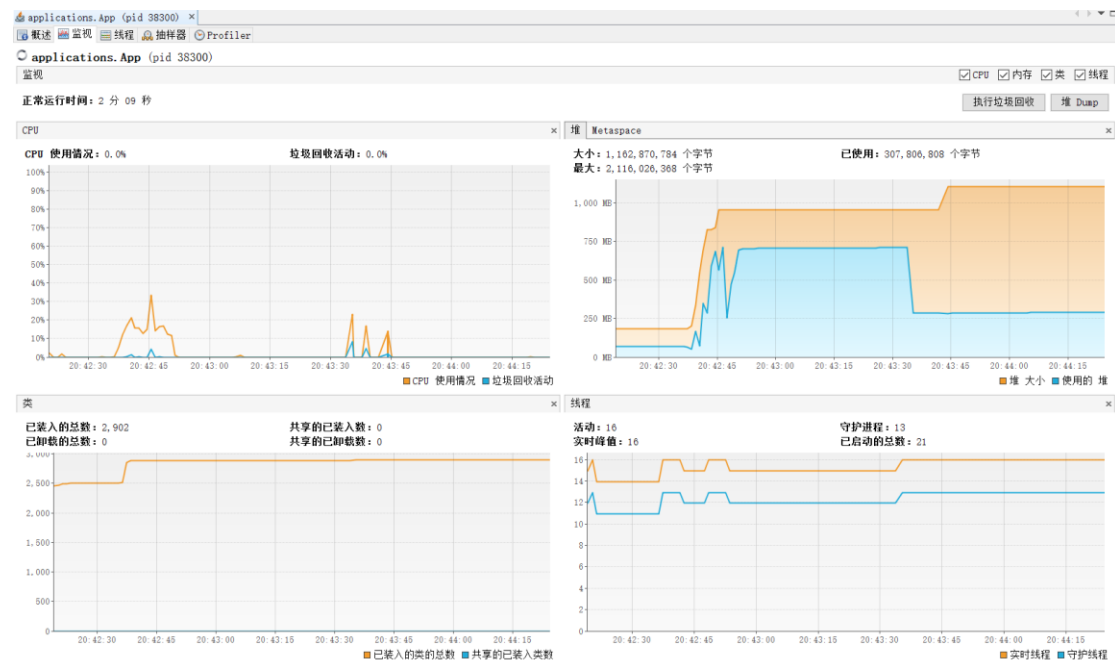
- 使用工具的方法：jmap -clstats 进程 pid
我的输入是：jmap -clstats 11052

```
C:\Users\86185>jmap -clstats 24336
Attaching to process ID 24336, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.202-b08
finding class loader instances ..done.
computing per loader stat ..done.
please wait.. computing liveness.liveness analysis may be inaccurate ...
class_loader  classes bytes  parent_loader  alive?  type
<bootstrap>      2067   3901072    null          live    <internal>
0x0000000082db9038    1    1084    0x0000000082dc9048    dead    sun/reflect/misc/MethodUtil@0x000000001000f4cb8
0x00000000da430000    1     878      null          dead    sun/reflect/DelegatingClassLoader@0x0000000010000a028
0x00000000da430190    1     878      null          dead    sun/reflect/DelegatingClassLoader@0x0000000010000a028
0x00000000da430320    1    1471      null          dead    sun/reflect/DelegatingClassLoader@0x0000000010000a028
0x0000000082dc9048   46   74757    0x0000000082dc90b8    dead    sun/misc/Launcher$AppClassLoader@0x0000000010000f8d8
0x0000000082dc90b8    4    4572      null          dead    sun/misc/Launcher$ExtClassLoader@0x0000000010000fc80
0x00000000da4300c8    1     878      null          dead    sun/reflect/DelegatingClassLoader@0x0000000010000a028
0x00000000da430258    1     878      null          dead    sun/reflect/DelegatingClassLoader@0x0000000010000a028
0x00000000da4303e8    1     889    0x0000000082db9038    dead    sun/reflect/DelegatingClassLoader@0x0000000010000a028
0x0000000082f5a3d8    0      0    0x0000000082dc9048    dead    java/util/ResourceBundle$RBCLoader@0x000000001000a1110
total = 11      2124   3987357      N/A      alive=1, dead=10      N/A
```

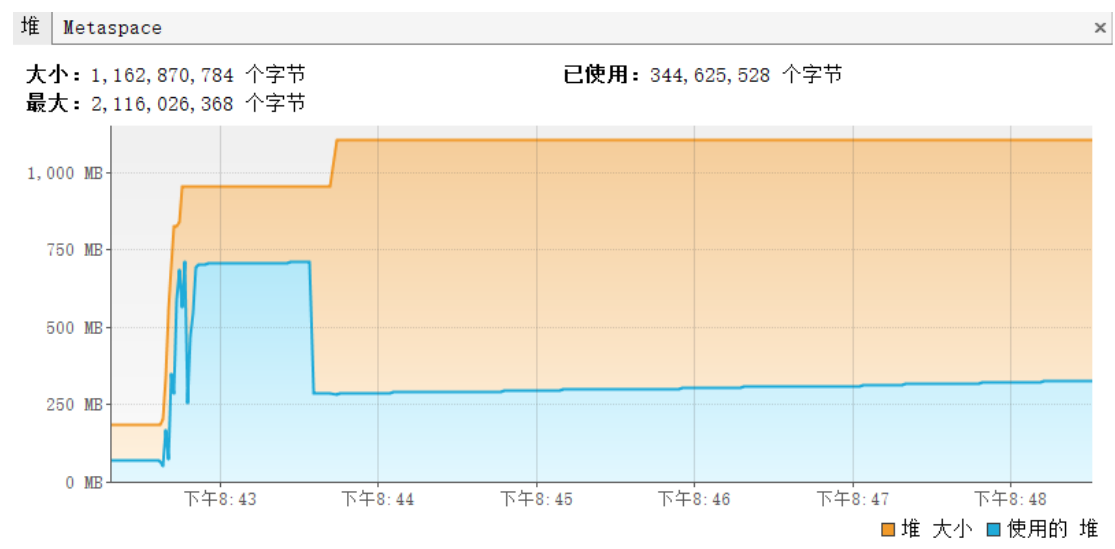
3.3.6 使用 jmap -permstat 命令行工具

此命令为 JDK8 以前的版本使用，我的 JDK 版本为 JDK8，命令已换为 3.3.5 节内容。

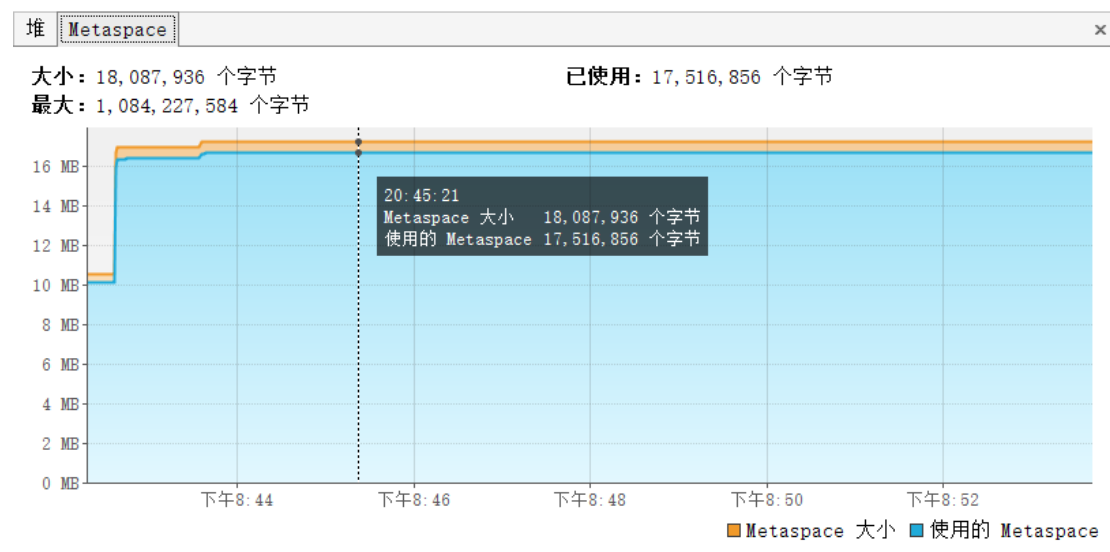
3.3.7 使用 JMC/JFR、jconsole 或 VisualVM 工具（我选择 VisualVM 工具）



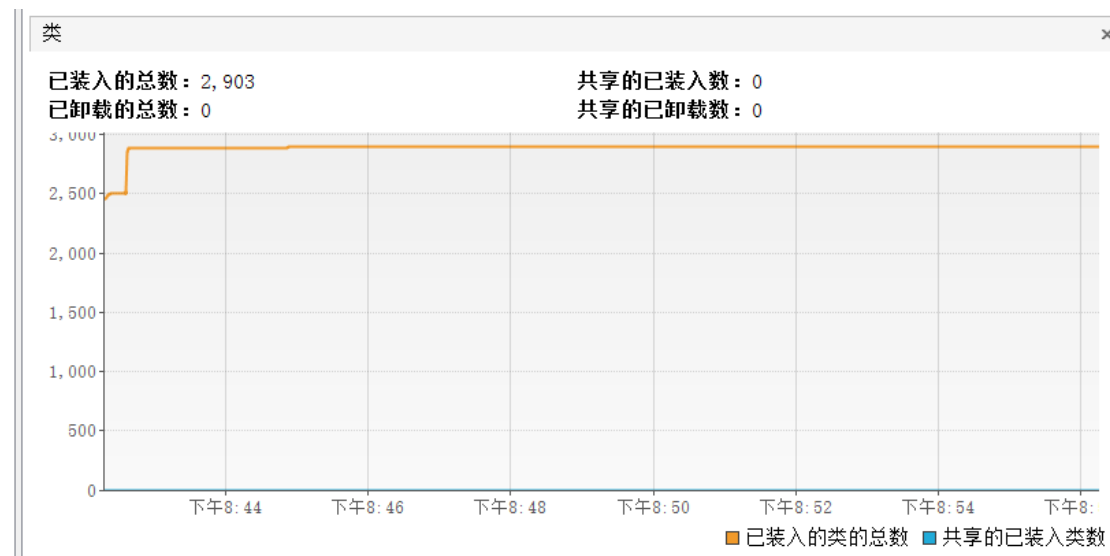
堆内存最大为 2116026368 字节, 当前大小为 1162870784 字节, 已使用 344625528 字节。



metaspace 最大为 1084227584 字节, 当前大小为 18087936 字节, 已使用 17516856 字节。

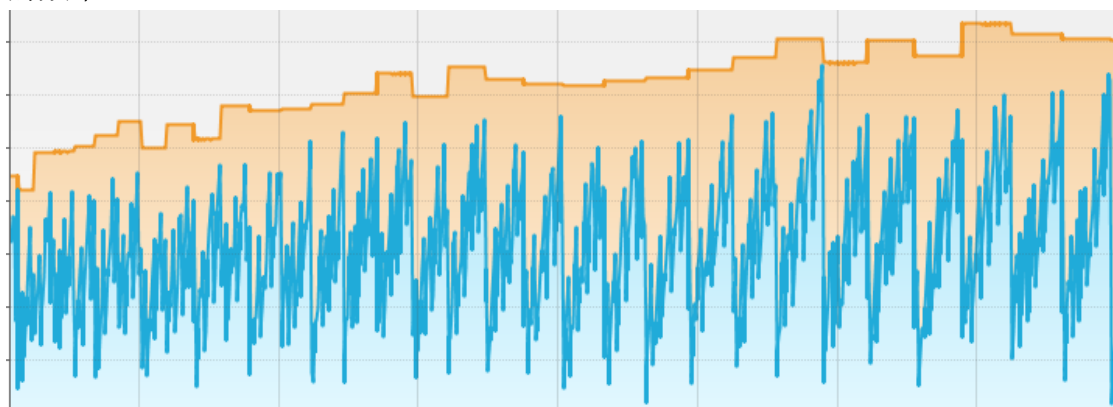


装载的类实例为 2903 个，如下截图。



3.3.8 分析垃圾回收过程

- **minor GC 频率比较频繁**：适当增加年轻代的内存空间大小来降低 minor GC 的频率。



- **同时发现 Full GC 也相对比较频繁**：由于执行一次 Full GC 的时间比较短，可以考虑适当增加老年代的内存空间。

3.3.9 配置 JVM 参数并发现优化的参数配置

- **最终我选择的虚拟机配置如下：**

```
-XX:+UseG1GC -XX:NewRatio=2
-Xms1536m -Xmx1536m
```

- **选择的垃圾回收算法是 G1GC，老年代和新生代的比例为 2，总空间初始值为 1536M，最大值为 1536M。**

VM arguments:

```
-XX:+UseG1GC -XX:NewRatio=2
-Xms1536m -Xmx1536m
```

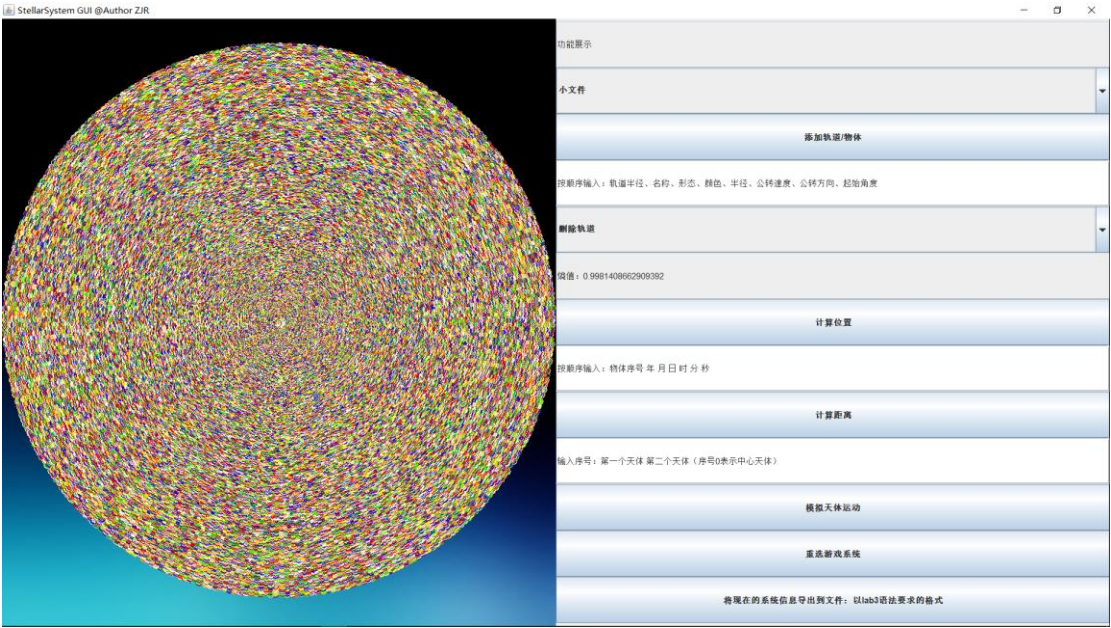
- **具体表现**

在相当长的测试时间内，完全没有发生 Full GC 的操作；
minor GC 频率比较合适，且每次时间为 50ms 左右，较为正常；

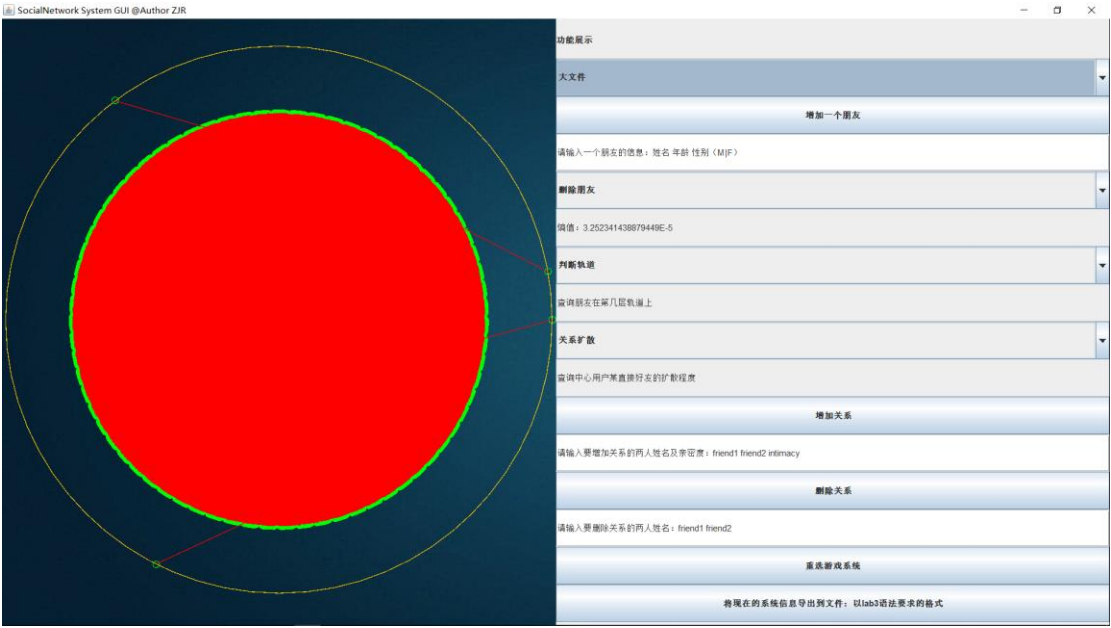
```
C:\Users\86185>jstat -gcutil 38752 ls
S0    S1    E    O    M    CCS    YGC    YGCT    FGC    FGCT    GCT
0.00 100.00 46.61 68.90 94.36 84.29 111    5.441    0    0.000    5.441
0.00 100.00 92.63 23.10 94.39 84.29 112    5.487    0    0.000    5.487
0.00 100.00 44.42 25.20 94.39 84.29 114    5.572    0    0.000    5.572
0.00 100.00 92.83 25.20 94.39 84.29 115    5.622    0    0.000    5.622
0.00 100.00 39.64 27.10 94.39 84.29 117    5.717    0    0.000    5.717
0.00 100.00 94.82 27.71 94.41 84.29 118    5.717    0    0.000    5.717
0.00 100.00 32.79 31.32 94.47 84.29 119    5.907    0    0.000    5.907
0.00 100.00 69.72 34.32 94.47 84.29 120    5.979    0    0.000    5.979
0.00 100.00 17.73 38.08 94.47 84.29 122    6.064    0    0.000    6.064
0.00 100.00 67.13 40.09 94.47 84.29 123    6.100    0    0.000    6.100
0.00 100.00 15.34 43.89 94.52 84.29 125    6.177    0    0.000    6.177
0.00 100.00 63.55 45.80 94.54 84.29 126    6.218    0    0.000    6.218
0.00 100.00 10.76 49.70 94.54 84.29 128    6.303    0    0.000    6.303
0.00 100.00 58.76 51.61 94.55 84.29 129    6.348    0    0.000    6.348
0.00 100.00 78.09 53.77 94.55 84.29 130    6.394    0    0.000    6.394
0.00 100.00 2.04 57.78 94.55 84.29 132    6.551    0    0.000    6.551
0.00 100.00 47.81 60.73 94.56 84.29 133    6.627    0    0.000    6.627
0.00 100.00 94.82 62.64 94.56 84.29 135    6.676    0    0.000    6.676
0.00 100.00 49.00 66.64 94.56 84.29 136    6.747    0    0.000    6.747
```

3.4 Dynamic Program Profiling

综述，下图 1 是我的超大文件的 GUI 图形界面，我选择的工具是 VisualVM
行星系统如下

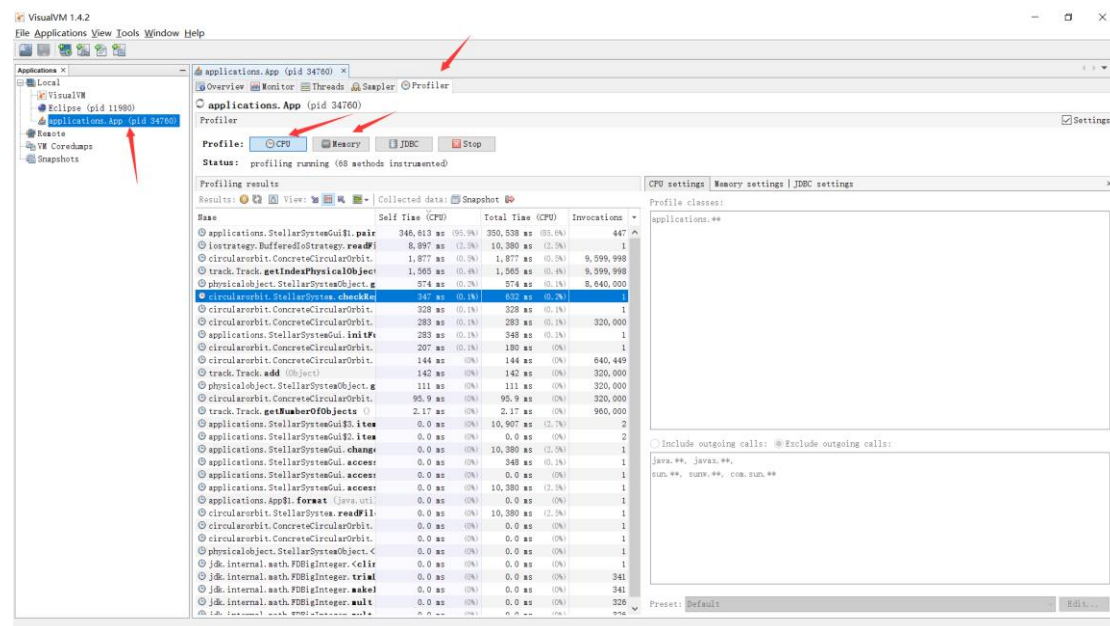


社交系统如下



3.4.1 使用 VisualVM 进行 CPU Profiling

- VisualVM 的 CPU Profiling 界面



● 得到的结果分析

1, 我要监控的类是 App 类, 这是我的客服端主类。

我选择读取文件路径是: `src/Spring2019_HITCS_SC_Lab5-master/StellarSystem.txt`

Name	Total Time	Invocations
applications.StellarSystemGui\$1.paint (java.awt.Graphics)	802,903 ms (91.5%)	481
circularorbit.ConcreteCircularOrbit.getTrack (int)	12,194 ms (1.4%)	20,396,889
applications.StellarSystemGui\$3.itemStateChanged (java.a	11,458 ms (1.3%)	2
applications.StellarSystemGui.access\$10 (applications.Ste	10,893 ms (1.2%)	1
applications.StellarSystemGui.changeStellarSystem (int)	10,893 ms (1.2%)	1
circularorbit.StellarSystem.readFileAndCreateSystem (iostrategy.IOStrategy)	10,889 ms (1.2%)	1
iostrategy.BufferedIOStrategy.readFileAndCreateSystem (10,889 ms (1.2%)	1
track.Track.getIndexPhysicalObject (int)	3,036 ms (0.3%)	20,396,888
physicalobject.StellarSystemObject.getRevolutionSpeed ()	1,559 ms (0.2%)	19,436,890
circularorbit.StellarSystem.checkRep ()	615 ms (0.1%)	1
applications.StellarSystemGui.access\$8 (applications.Stel	345 ms (0%)	1
applications.StellarSystemGui.initFunc3 ()	345 ms (0%)	1
circularorbit.ConcreteCircularOrbit.sortTrack ()	334 ms (0%)	1
circularorbit.ConcreteCircularOrbit.getSystemEntropy ()	212 ms (0%)	1
circularorbit.ConcreteCircularOrbit.addTrack (track.Track	165 ms (0%)	320,000
circularorbit.ConcreteCircularOrbit.getTrackByRadius (do	103 ms (0%)	320,000
track.Track.getNumberOfObjects ()	92.9 ms (0%)	960,000
track.Track.add (Object)	81.4 ms (0%)	320,000
circularorbit.ConcreteCircularOrbit.getTracksNumber ()	61.6 ms (0%)	640,483
physicalobject.StellarSystemObject.getPlanetName ()	26.6 ms (0%)	320,000
jdk.internal.math.FDBigInteger.<clinit> ()	0.949 ms (0%)	1
jdk.internal.math.FDBigInteger.mult (int)	0.600 ms (0%)	326
applications.App\$1.format (java.util.logging.LogRecord)	0.403 ms (0%)	1
jdk.internal.math.FDBigInteger.mult (int[], int, int, int)	0.144 ms (0%)	326
jdk.internal.math.FDBigInteger.trimLeadingZeros ()	0.075 ms (0%)	341
physicalobject.StellarSystemObject.<clinit> ()	0.067 ms (0%)	1
jdk.internal.math.FDBigInteger.makeImmutable ()	0.044 ms (0%)	341
applications.StellarSystemGui\$2.itemStateChanged (java.a	0.022 ms (0%)	2
circularorbit.ConcreteCircularOrbit.initSystem ()	0.017 ms (0%)	1
circularorbit.ConcreteCircularOrbit.addCentralPoint (Obj	0.012 ms (0%)	1

2，图表形式展示执行的方法所耗费的时间、比例

paint	158,253ms	96.6%
getTrack	2,508ms	1.5%
itemStateChanged	921ms	0.6%
getIndexPhysicalObject	571ms	0.3%
sortTrack	326ms	0.2%
access\$8	320ms	0.2%
initFunc3	320ms	0.2%
getRevolutionSpeed	301ms	0.2%
getSystemEntropy	271ms	0.2%
getNumberOfObjects	51.0ms	0%
getTracksNumber	25.1ms	0%
App\$1.format	0.328ms	0%
deleteTrack	0.063ms	0%
access\$7	0.005ms	0%

3，耗时方法分析

- a) 经过分析发现，耗时最多的是 GUI 画板的 paint 方法，占据时常 96.6%：因为这个画板是实时更新的，所以占比最大，因为他的程序周期是最长

的，从用户选择读取这个超大文件开始，就开始运行。

- b) 其次是 ConcreteCircularOrbit 类的 getTrack 方法。因为在画图的时候疯狂调用获取轨道的方法，所以占据时长较长。
- c) itemStateChanged 耗时其次。是正常行为，因为接受来自用户端的鼠标输入。

相关的解决办法

- a) 发现画板是实时更新的。

做了相应的修改：画板只有在改变时才会更新。

applications.StellarSystemGui\$.paint (java.awt.Graphics)	77,437 ms
iostrategy.BufferedIoStrategy.saveSystemInfoInFile (circularorbit.StellarSystemGui)	35,935 ms

- b) 进行 a) 的修改后，发现将信息储存耗时较长。

原因是因为忘记了将 3.2 节做的 IO 优化分支合并到当前 master 上。合并分支后发现 saveSystemInfoInFile 耗时大幅下降，直观表现如终端截图。

行星系统BufferedIO写文件:154ms

5月30, 2019 3:41:31 下午 applications.StellarSystemGui\$9.actionPerformed
信息：成功将系统信息写入到默认文件:src/outputFile/StellarSystem.txt

3.4.2 使用 VisualVM 进行 Memory profiling

● VisualVM 的 memory 的 profile 界面截图

Name	Live Bytes	Live Objects	Allocated Objects	Gener...
java.awt.geom.AffineTransform	8,318,016 B (34.5%)	115,528 (18.1%)	961,658 (0.5%)	8
java.awt.Rectangle	6,478,936 B (26.5%)	202,498 (31.5%)	5,761,695 (40.8%)	6
java.awt.Insets	2,150,336 B (8.5%)	67,198 (10.6%)	1,920,583 (13.6%)	6
java.beans.PropertyChangeEvent	2,149,312 B (8.5%)	67,166 (10.5%)	1,920,166 (13.6%)	5
sun.swing.SwingUtilities2\$KeyPair	1,618,272 B (6.7%)	67,428 (10.6%)	960,353 (0.6%)	5
java.awt.Dimension	1,617,552 B (6.7%)	67,398 (10.6%)	1,920,166 (13.6%)	6
java.util.TreeMap\$Entry	565,920 B (2.3%)	14,148 (2.2%)	166,494 (1.2%)	7
java.awt.geom.Point2D\$Float	147,144 B (0.6%)	6,131 (1%)	6,949 (0%)	1
java.lang.Object	132,000 B (0.5%)	375 (0.1%)	5,616 (0%)	10
java.io.ObjectStreamClass\$WeakClassKey	89,184 B (0.4%)	3,716 (0.6%)	124,260 (0.9%)	5
java.util.TreeMap	60,576 B (0.2%)	1,262 (0.2%)	15,132 (0.1%)	7
java.security.AccessControlContext	50,200 B (0.2%)	1,255 (0.2%)	4,217 (0%)	12
byte[]	49,640 B (0.2%)	13 (0%)	32 (0%)	1
java.io.ObjectStreamClass	47,632 B (0.2%)	458 (0.1%)	5,684 (0.1%)	6
java.util.TreeMap\$KeyIterator	40,320 B (0.2%)	1,260 (0.2%)	33,456 (0.2%)	7
java.lang.String	38,224 B (0.2%)	2,389 (0.4%)	40,282 (0.3%)	8
java.awt.event.MouseEvent	30,560 B (0.1%)	382 (0.1%)	1,041 (0%)	11
java.lang.Long	29,952 B (0.1%)	1,248 (0.2%)	20,608 (0.1%)	7
javax.management.openmbean.CompositeDataSupport	28,512 B (0.1%)	1,188 (0.2%)	15,132 (0.1%)	7
java.util.HashMap\$Node	27,520 B (0.1%)	860 (0.1%)	6,658 (0%)	8
java.util.HashMap	27,264 B (0.1%)	568 (0.1%)	8,585 (0.1%)	6
java.util.Collections\$UnmodifiableCollection\$1	22,080 B (0.1%)	552 (0.1%)	18,325 (0.1%)	5
java.util.TreeMap\$KeySet	20,192 B (0.1%)	1,262 (0.2%)	15,132 (0.1%)	7
sun.java2d.SunGraphics2D	19,968 B (0.1%)	96 (0%)	299 (0%)	1
java.util.TreeMap\$EntrySet	19,824 B (0.1%)	1,239 (0.2%)	15,109 (0.1%)	7
java.util.TreeMap\$EntryIterator	16,864 B (0.1%)	527 (0.1%)	17,717 (0.1%)	5
java.lang.StringBuilder	14,064 B (0.1%)	586 (0.1%)	16,214 (0.1%)	4
java.io.SerialCallbackContext	13,896 B (0.1%)	579 (0.1%)	20,065 (0.1%)	5
sun.awt.EventQueue\$tes	11,184 B (0%)	466 (0.1%)	1,388 (0%)	7
java.io.ObjectOutputStream\$HandleTable	10,760 B (0%)	269 (0%)	3,990 (0%)	7
java.io.DataOutputStream	10,320 B (0%)	258 (0%)	5,500 (0%)	8
java.util.concurrent.ConcurrentHashMap	9,472 B (0%)	145 (0%)	1,472 (0%)	3
java.io.ObjectInputStream\$BlockDataInputStream	9,016 B (0%)	181 (0%)	2,321 (0%)	6
java.awt.geom.Rectangle2D\$Float	8,384 B (0%)	262 (0%)	292 (0%)	1
java.util.concurrent.BFB	7,680 B (0%)	745 (0%)	9,681 (0%)	7

● 得到的结果统计

对象	分配对象个数
java.awt.Rectangle	5,761,695 (40.8%)
java.awt.Insets	1,920,583 (13.6%)
java.awt.Dimension	1,920,166 (13.6%)
java.lang.String	40,282 (0.3%)
java.lang.Long	20,608 (0.1%)
java.lang.StringBuilder	16,214 (0.1%)

java.util.HashMap	8,585 (0.1%)
-------------------	--------------

● 结果分析

- 1, 发现 java.awt 库中的类对象最多。是因为我选择使用 GUI 将这个超大文件进行行星系统绘制，占据了大量的空间内存资源。
- 2, 发现 String 对象也很多，大约有 4 万个。因为行星名、行星状态、行星颜色、输出系统信息的文件路径、公转方向以及日志的输入 message 等都是构建的 String 对象。
- 3, StringBuilder 对象也很多，大约 16000 个。因为输出系统信息的时候构建的是 StringBuilder 对象，输出科学记数法的时候也是构建的 String Builder 对象。
- 4, HashMap 对象数目达 8585 个。为了提高读取文件、构建系统、查找信息等操作的速度，许多数据结构都是使用的 hashMap，如截图

```
private Map<Track<E>, Integer> allTracks = new HashMap<>(); // 储存所有的轨道及对应的轨道序号
private Map<Double, Track<E>> allRadius = new HashMap<>(); // 储存所有的半径及对应的轨道
private Map<Integer, Track<E>> allOrders = new HashMap<>(); // 储存所有的序号和对应的轨道
```

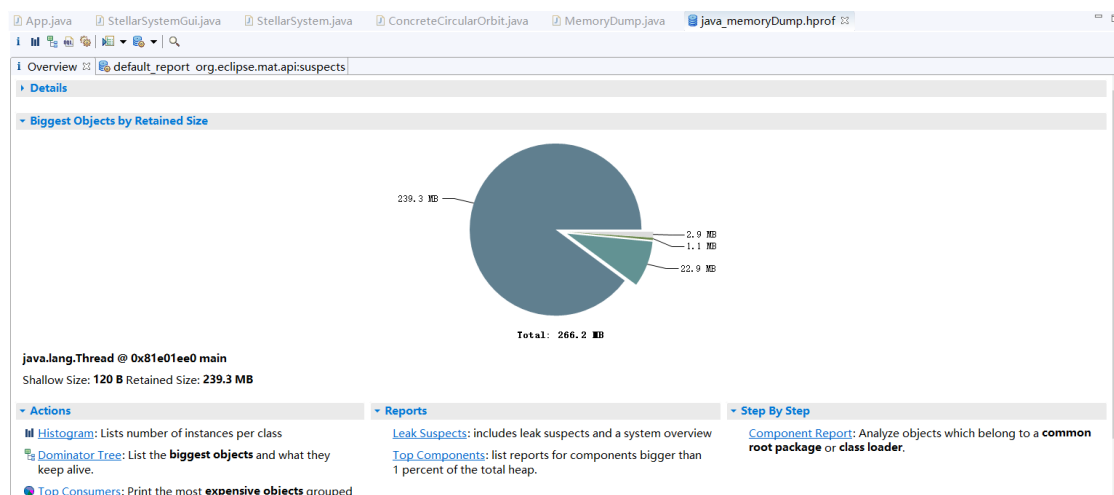
3.5 Memory Dump Analysis and Performance Optimization

3.5.1 内存导出

- 1, 建立一个专门的类 recordiotime.MemoryDump：便于导出构建完成大文件系统后的内存占用情况；
- 2, 程序中使用简单的 Scanner 代码：保证程序可以等待用户进行内存导出。

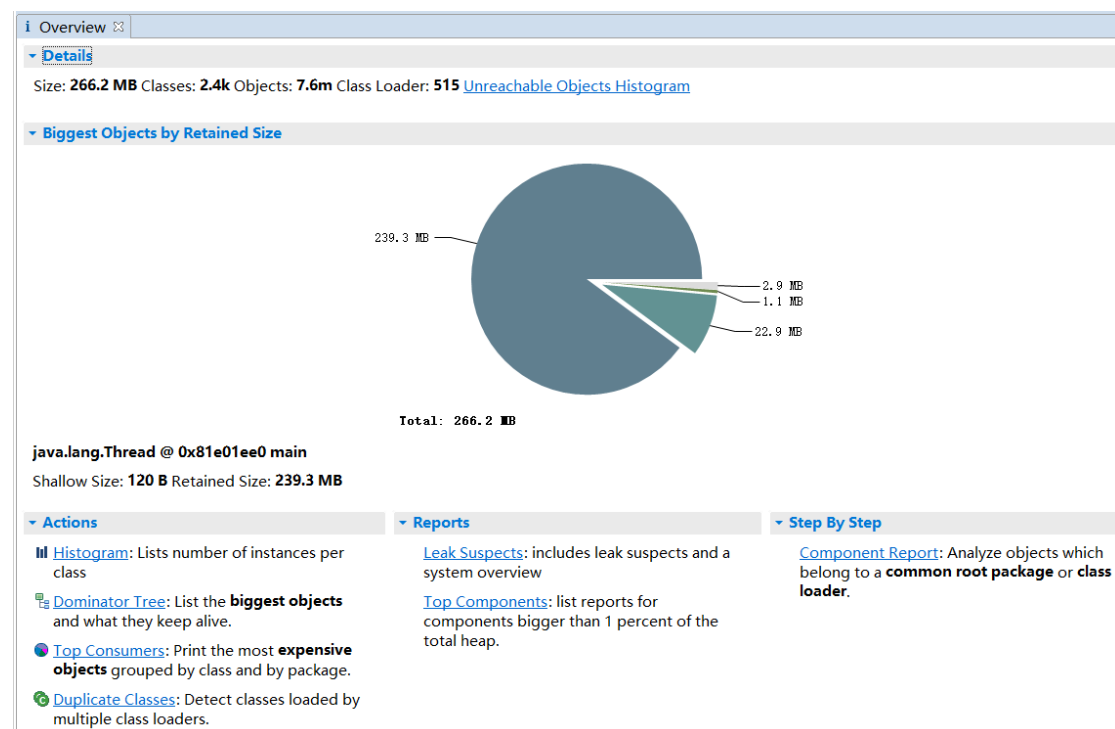
```
public static void main(String[] args)
    throws IOException, FileChooseException {
    StellarSystem stellarSystem = new StellarSystem();
    stellarSystem.setReadFile(
        new File("src/Spring2019_HITCS_SC_Lab5-master/StellarSystem.txt"));
    stellarSystem.setWriteFilePath("src/outputFile/StellarSystem.txt");
    stellarSystem.readFileAndCreateSystem(new BufferedIoStrategy());
    // 让程序停止下来，便于获取当前内存占用情况
    System.out.println("可以导出内存占用情况");
    Scanner scanner = new Scanner(System.in);
    String string = scanner.next();
    scanner.close();
    System.out.println(string);
}
```

- 3, 使用 eclipse 的内存导出功能导出内存状态：项目右键->import->HeapDump->选择当前执行程序即可



3.5.2 使用 MAT 分析内存导出文件

- 1, 使用 eclipse 的 mat 打开上一步生成的 heap dump 文件: file->openfile->选择内存导出文件即可;
- 2, overview 视图



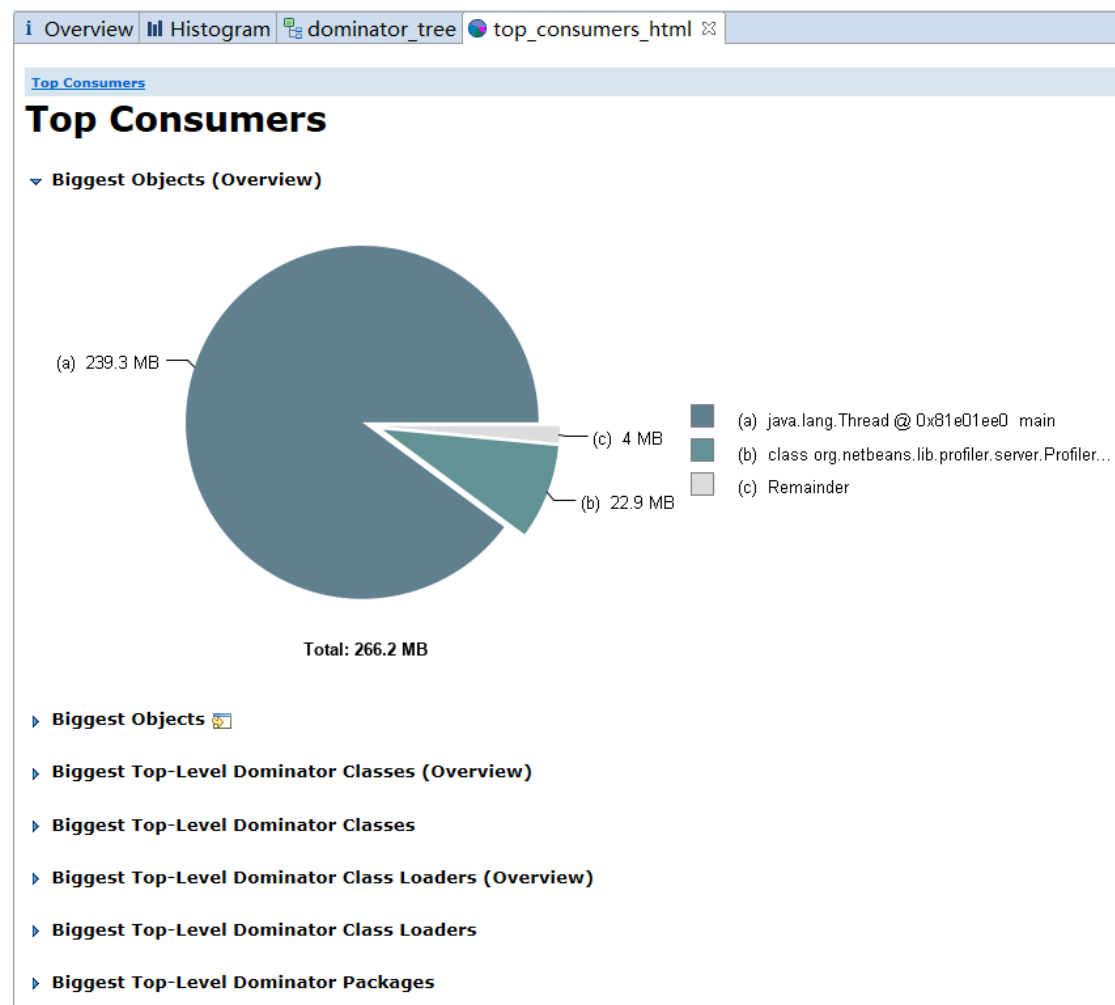
- 3, histogram 视图

i Overview Histogram				
Class Name	Objects	Shallow H...	Retained Heap	
<Regex>	<Numer...	<Numeric>	<Numeric>	
java.util.HashMap\$Node[]	642,024	57,640,544	>= 127,158,496	
java.util.HashMap\$Node	1,607,060	51,425,920	>= 69,587,920	
byte[]	1,324,010	35,763,968	>= 35,763,968	
java.lang.String	1,307,742	31,385,808	>= 63,213,824	
java.util.HashMap	643,757	30,900,336	>= 158,046,064	
physicalobject.StellarSyste...	320,000	20,480,000	>= 81,920,232	
java.lang.Integer	641,179	10,258,864	>= 10,259,464	
track.Track	320,000	10,240,000	>= 112,640,136	
org.netbeans.lib.profiler.se...	226,367	9,054,680	>= 12,990,952	
java.lang.Double	320,001	7,680,024	>= 7,680,408	
java.lang.Object[]	9,910	2,127,424	>= 2,276,016	
java.lang.ref.WeakReferenc...	2	2,064,968	>= 24,104,960	
java.util.TreeMap\$Entry	51,456	2,058,240	>= 2,058,272	
java.io.ObjectStreamClass\$...	40,061	1,281,952	>= 1,281,952	
char[]	2,781	1,217,288	>= 1,217,288	
int[]	3,872	533,352	>= 533,352	
java.io.ObjectStreamClass	3,449	358,696	>= 404,976	
java.util.TreeMap\$KeyIterat...	10,061	321,952	>= 321,952	
java.util.TreeMap	4,619	221,712	>= 225,792	
java.lang.management.Thre...	1,984	206,336	>= 252,336	
java.util.TreeMap\$EntryIter...	5,615	179,680	>= 179,680	
java.lang.Long	7,208	172,992	>= 173,272	
java.io.SerialCallbackContext	6,537	156,888	>= 157,024	
java.lang.StringBuilder	6,347	152,328	>= 431,448	
java.util.Collections\$Unmo...	5,470	131,280	>= 131,280	
javax.management.openm...	6,080	119,296	>= 119,296	
javax.management.openm...	4,591	110,184	>= 111,560	
java.lang.StackTraceElemen...	3,971	91,184	>= 91,184	
java.util.concurrent.Concurr...	2,706	86,592	>= 228,280	
java.io.DataOutputStream	2,142	85,680	>= 137,320	
java.lang.StackTraceElement	1,728	82,944	>= 83,024	
java.lang.ref.WeakReference	2,404	76,928	>= 77,056	
java.util.TreeMap\$EntrySet	4,599	73,584	>= 73,584	
java.util.TreeMap\$KeySet	4,594	73,504	>= 73,504	

4, dominator tree 视图

i Overview Histogram dominator_tree			
Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
> java.lang.Thread @ 0x81e01ee0 main Thread	120	250,942,960	89.89%
> class org.netbeans.lib.profiler.server.ProfilerRuntime	16	24,026,328	8.61%
> class org.netbeans.lib.profiler.server.ProfilerRuntime	32	1,200,216	0.43%
> class sun.util.calendar.ZoneInfoFile @ 0xab92b980 S	120	151,176	0.05%
> class org.netbeans.lib.profiler.server.ProfilerInterface	200	90,992	0.03%
> class java.lang.invoke.MethodType @ 0x81e14358 S	48	85,912	0.03%
> class sun.util.locale.provider.LocaleProviderAdapter	32	68,256	0.02%
> java.util.HashSet @ 0xab95ceb8	16	63,152	0.02%
> char[28672] @ 0x81e603d8 \ufffd\ufffd\ufffd\ufffd\u	57,360	57,360	0.02%
> class sun.nio.cs.GBK @ 0x81f48c08 System Class	32	55,584	0.02%
> class sun.util.resources.Bundles @ 0xabe9c468 Syste	24	53,904	0.02%
> char[256][] @ 0x81ed0228	1,040	51,440	0.02%
> java.util.zip.ZipFile\$Source @ 0x81e9e3e8	64	51,160	0.02%
> class sun.util.cldr.CLDRBaseLocaleDataMetaInfo\$TZC	8	51,040	0.02%
> com.sun.management.internal.DiagnosticCommandl	40	48,744	0.02%
> class org.netbeans.lib.profiler.server.ProfilerRuntime	80	45,072	0.02%
> class jdk.internal.loader.BuiltinClassLoader @ 0x81f2	16	38,280	0.01%
> java.util.zip.ZipFile\$Source @ 0x81e7d9e0	64	33,896	0.01%
> sun.security.provider.Sun @ 0xaba89158	104	32,168	0.01%
> class java.lang.System @ 0x81ef95f0 System Class	40	28,008	0.01%
> class java.io.ObjectStreamClass\$Caches @ 0xab90c9	16	27,072	0.01%
> jdk.internal.loader.ClassLoaders\$AppClassLoader @	104	25,752	0.01%
> java.util.zip.ZipFile\$Source @ 0x81e75f30	64	25,720	0.01%
> java.io.PrintStream @ 0x81f09b98	40	25,168	0.01%
> int[6284] @ 0xa703f868	25,152	25,152	0.01%
> java.lang.Module @ 0x81e06c10	48	22,608	0.01%
> java.lang.ModuleLayer @ 0x81e03750	40	20,760	0.01%
> java.util.logging.LogManager @ 0x81e8d2a0	56	18,080	0.01%
> class org.netbeans.lib.profiler.server.ClassLoaderMa	40	15,904	0.01%
> java.util.zip.ZipFile\$Source @ 0x81e98c48	64	15,104	0.01%
> com.sun.jmx.mbeanserver.PerInterface @ 0xaba48fb	40	14,264	0.01%
> class java.nio.charset.Charset @ 0x81f49d20 System	32	14,200	0.01%
> java.util.zip.ZipFile\$Source @ 0x81ebb628	64	14,088	0.01%
> class java.math.BigInteger @ 0xabd781b0 System Cl	160	13,856	0.00%
> java.util.zip.ZipFile\$Source @ 0xab0136e0	64	13,712	0.00%

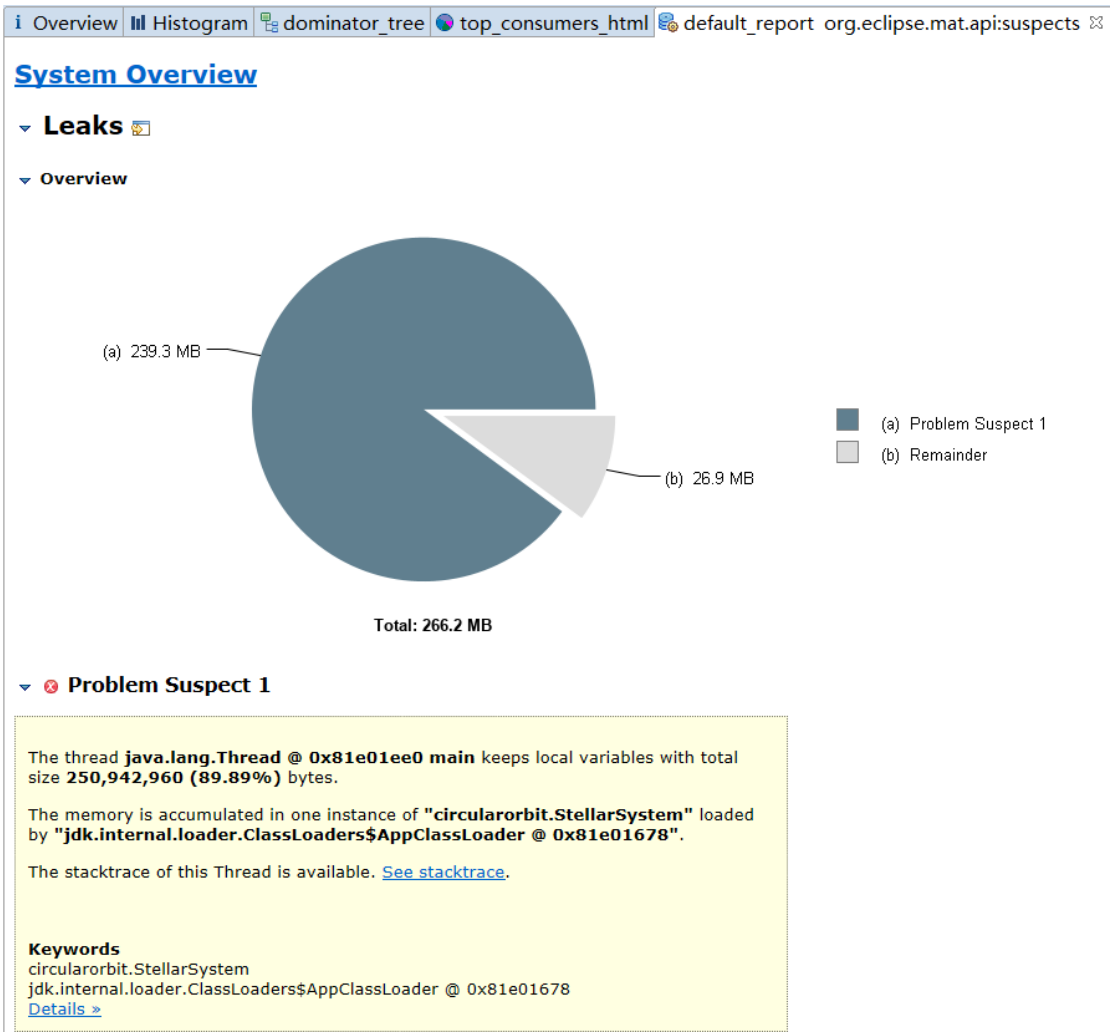
5, top consumers 视图



6, leak suspects report 视图

发现存在一个可能的内存泄漏情况

The thread **java.lang.Thread @ 0x81e01ee0 main** keeps local variables with total size **250,942,960 (89.89%)** bytes



3.5.3 发现热点/瓶颈并改进、改进前后的性能对比分析

1，发现热点/瓶颈

发现三个数据结构 hashMap 以及一个 ArrayList 消耗内存 retainedHeap 较多。

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
java.lang.Thread @ 0x81e01ee0 main Thread	120	250,942,960
<Java Local> circularorbit.StellarSystem @ 0x81e01a10	48	250,928,488
allRadius java.util.HashMap @ 0x81f051f8	48	20,017,216
allTracks java.util.HashMap @ 0x81f2e1c0	48	17,455,200
allOrders java.util.HashMap @ 0x81f051c8	48	17,455,184
planetList java.util.ArrayList @ 0x81f05040	24	1,440,624
<class> class circularorbit.StellarSystem @ 0x81e01b00	16	592
readFile java.io.File @ 0x81e01a40	32	128
centralPoint centralobject.Stellar @ 0x81f05068	32	88
writeFilePath java.lang.String @ 0x81e29950	24	24
Total: 8 entries		

2，改进内存占用热点

由于构建系统必须的是轨道物体、轨道、恒星以及他们相应的属性，因此这

些内存是很难降低的，可以降低的是数据结构的节点数（比如 hashMap 的节点数、比如 String 的对象数目等）

我的操作有：删除了用处不大的数据结构 planetList（原本用于储存所有的行星）+将部分类的属性中的 String 初始设为 null，直到使用时才进行引用。

3，改进前后对比

热点代码部分已经没有了 planetList，整体占用情况大幅下降，下面为对比图，发现内存使用足足减少了 **27MB**，相当于总内存占用的 **10%**；对象数目减少了 **60 万左右**；类数目减少了也大幅减少。

Used heap dump	266.2 MB	Used heap dump	239.2 MB
Number of objects	7,613,769	Number of objects	7,065,221
Number of classes	2,447	Number of classes	1,094
Number of class loaders	515	Number of class loaders	4
Number of GC roots	2,253	Number of GC roots	1,036

左图为修改前，右图为修改后

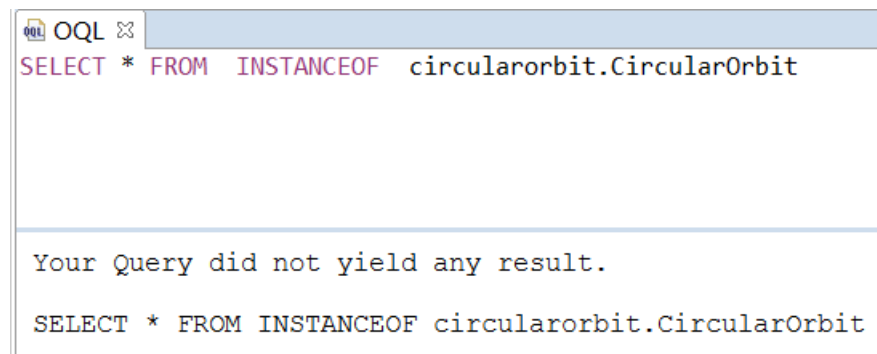
3.5.4 在 MAT 内使用 OQL 查询内存导出

3.5.4.1 CircularOrbit 的所有对象实例

● OQL 查找的输入

```
SELECT * FROM INSTANCEOF circularorbit.CircularOrbit
```

查询结果为空，因为并没有它的示例，如截图。



● OQL 查找的输入

```
SELECT * FROM INSTANCEOF circularorbit.ConcreteCircularOrbit
```

查询结果是找到了一个实例：为 StellarSystem 对象，如截图


```
SELECT * FROM INSTANCEOF circularorbit.ConcreteCircularOrbit
```

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
▼ circularorbit.StellarSystem @ 0x81e00b78	40	249,487,856
> <class> class circularorbit.StellarSystem	16	592
> readFile java.io.File @ 0x81e00ba0	32	128
> writeFilePath java.lang.String @ 0x81ea	24	24
> allTracks java.util.HashMap @ 0x81f0fcc	48	17,455,200
> centralPoint centralobject.Stellar @ 0x8	32	88
> allOrders java.util.HashMap @ 0x81f32c	48	17,455,184
> allRadius java.util.HashMap @ 0x81f32c	48	20,017,216
Σ Total: 7 entries		

3.5.4.2 大于特定长度 n 的 String 对象：我选择长度为 10

- OQL 查找输入为

```
SELECT OBJECTS s.value FROM java.lang.String s WHERE
s.value.@length>10
```

- 查询结果如截图

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
> byte[35] @ 0xa62ac368 java.util.Arrays.useLegacyMergeSort	56	56
> byte[21] @ 0xa62ac1d0 java.util.Collections	40	40
> byte[37] @ 0xa62aadf0 circularorbit.ConcreteCircularOrbit\$1	56	56
> byte[17] @ 0xa62aad60 (this Collection)	40	40
> byte[19] @ 0xa62aabb0 java.util.ArrayList	40	40
> byte[19] @ 0xa62aaa20 java.io.PrintStream	40	40
> byte[23] @ 0xa62aa7c0 java.lang.StringBuilder	40	40
> byte[14] @ 0xa615e9e8 Invalid size:	32	32
> byte[13] @ 0xa615e9b0 > To Index:	32	32
> byte[12] @ 0xa615e978 From Index:	32	32
> byte[48] @ 0xa6100018 Comparison method violates its gener	64	64
> byte[17] @ 0xa4d3c3e8 java.util.HashSet	40	40
> byte[14] @ 0xa4c10818 Illegal size:	32	32
> byte[18] @ 0xa4c10798 Illegal capacity:	40	40
> byte[18] @ 0xa4added0 .Spe.....n.u.l.l.	40	40
> byte[46] @ 0xa4adde78 c.h.e.c.k.R.e.p..c6e0R._8^.....e...b.e.N	64	64
> byte[44] @ 0xa4adde20 L.fh.S.JS._KN.].\N\$N.v...fSOh.S.JS._KN	64	64
> byte[24] @ 0xa4addde0 . .~-N)YSO.TX[(W.v.T.`Q	40	40
> byte[18] @ 0xa4addda0 h.S..NL..fpe.v^..l	40	40
> byte[12] @ 0xa4addd68 -N._R`.f.1Y	32	32
> byte[11] @ 0xa347c7c0 UNNECESSARY	32	32
> byte[19] @ 0xa347c3e8 9223372036854775808	40	40
Σ Total: 22 of 2,695 entries; 2,673 more		

3.5.4.3 大于特定大小的任意类型对象实例：我选择大小为 100 000

- OQL 查找输入为

```
SELECT * FROM instanceof java.lang.Object o WHERE
o.@usedHeapSize >= 100000
```

- 查询结果

搜索到了三个 HashMap 数据结构，如截图

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
> java.util.HashMap\$Node[524288] @ 0x8e600000	2,097,168	17,455,136
> java.util.HashMap\$Node[524288] @ 0x88d00000	2,097,168	17,455,136
> java.util.HashMap\$Node[524288] @ 0x88a00000	2,097,168	20,017,168
Σ Total: 3 entries		

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

- OQL 查找输入为

```
SELECT * FROM instanceof physicalobject.PhysicalObject
```

- 查询结果

查询到的轨道物体对象实例有 320 000 个，分别对应超大文件中的 320 000 个行星。所有对象总大小为 $256 \times 320000 = 20\,480\,000$ ，如截图

Class Name	Objects	Shallow Heap
<Regex>	<Numeric>	<Numeric>
physicalobject.StellarSystemObject	320,000	20,480,000

Class Name	Shallow Heap	Retained Heap
> physicalobject.StellarSystemObject @ 0xb37f7da8	64	256
> physicalobject.StellarSystemObject @ 0xb37f2438	64	256
> physicalobject.StellarSystemObject @ 0xb37ecae8	64	256
> physicalobject.StellarSystemObject @ 0xb37e7180	64	256
> physicalobject.StellarSystemObject @ 0xb37e1810	64	256
> physicalobject.StellarSystemObject @ 0xb37dbe98	64	256
> physicalobject.StellarSystemObject @ 0xb37d6548	64	256
> physicalobject.StellarSystemObject @ 0xb37d0be0	64	256
> physicalobject.StellarSystemObject @ 0xb37cb270	64	256
> physicalobject.StellarSystemObject @ 0xb37c5900	64	256
> physicalobject.StellarSystemObject @ 0xb37bffb0	64	256
> physicalobject.StellarSystemObject @ 0xb37ba648	64	256
> physicalobject.StellarSystemObject @ 0xb37b4cd8	64	256
> physicalobject.StellarSystemObject @ 0xb37af368	64	256
> physicalobject.StellarSystemObject @ 0xb37a9a18	64	256
> physicalobject.StellarSystemObject @ 0xb37a40b0	64	256
> physicalobject.StellarSystemObject @ 0xb379e740	64	256
> physicalobject.StellarSystemObject @ 0xb3798dd0	64	256
> physicalobject.StellarSystemObject @ 0xb3793480	64	256
> physicalobject.StellarSystemObject @ 0xb378db18	64	256
> physicalobject.StellarSystemObject @ 0xb37881a8	64	256
> physicalobject.StellarSystemObject @ 0xb3782838	64	256
> physicalobject.StellarSystemObject @ 0xb377cca0	64	256
> physicalobject.StellarSystemObject @ 0xb3777338	64	256
> physicalobject.StellarSystemObject @ 0xb37719c8	64	256
> physicalobject.StellarSystemObject @ 0xb376c058	64	256
> physicalobject.StellarSystemObject @ 0xb3766708	64	256
> physicalobject.StellarSystemObject @ 0xb3760da0	64	256
> physicalobject.StellarSystemObject @ 0xb375b430	64	256
> physicalobject.StellarSystemObject @ 0xb3755ac0	64	256
> physicalobject.StellarSystemObject @ 0xb3750170	64	256
> physicalobject.StellarSystemObject @ 0xb374a808	64	256
> physicalobject.StellarSystemObject @ 0xb3744e98	64	256
Total: 34 of 320,000 entries; 319,966 more		

3.5.4.5 所有包含元素数量大于 100 的 Collections 实例

- OQL 查找输入为

```
SELECT * FROM INSTANCEOF java.util.Collections c WHERE
c.size > 1000
```

- 查询结果如截图

Your Query did not yield any result.

```
SELECT * FROM INSTANCEOF java.util.Collections c WHERE (c.size > 1000)
```

- OQL 查找输入为

```
SELECT * FROM INSTANCEOF java.util.HashMap map WHERE
```

```
map.size > 1000
```

- 查询结果
查询到了 3 个 HashMap 的示例，如截图

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
> java.util.HashMap @ 0x81f32ca8	48	20,017,216
> java.util.HashMap @ 0x81f32c78	48	17,455,184
> java.util.HashMap @ 0x81f0fcc0	48	17,455,200
Σ Total: 3 entries		

3.5.4.6 我感兴趣的其他查询

- OQL 查找 centralobjectc（及其子类）对象实例的数量和总占用内存大小
查询输入为

```
SELECT * FROM INSTANCEOF centralobject.CentralObject
```

查询结果：找到了一个示例，对应行星系统中的中心天体。如截图

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
> centralobject.Stellar @ 0x81f32c20	32	88

- OQL 查找字符串为“Red”的 String 对象的数目和总占用内存大小
查询输入为

```
SELECT * FROM java.lang.String s WHERE toString(s).equals("Red")
```

查询结果：找到了 29274 个 String 对象，如截图

Class Name	Objects	Shallow Heap
<Regex>	<Numeric>	<Numeric>
java.lang.String	29,274	702,576

> java.lang.String @ 0xb3225928 Red	24	48
> java.lang.String @ 0xb32040d8 Red	24	48
> java.lang.String @ 0xb31f5e88 Red	24	48
> java.lang.String @ 0xb31f0540 Red	24	48
> java.lang.String @ 0xb3196ee0 Red	24	48
> java.lang.String @ 0xb3180948 Red	24	48
> java.lang.String @ 0xb3153bb8 Red	24	48
> java.lang.String @ 0xb3110af0 Red	24	48
> java.lang.String @ 0xb2f1ecf8 Red	24	48
> java.lang.String @ 0xb2f08760 Red	24	48
> java.lang.String @ 0xb2e74200 Red	24	48
> java.lang.String @ 0xb2e41d58 Red	24	48
> java.lang.String @ 0xb2e0f8b8 Red	24	48
Σ Total: 42 of 29,274 entries; 29,232 more		

3.5.5 观察 jstack 导出程序运行时的调用栈

首先应该获取进程的 pid：利用 windows 的资源管理器；

其次导出调用栈状态：在 cmd 中键入 jstack pid 即可获取。

批注：在读取文件并进行 GUI 绘图的时候已经进行了合法性判断+增删轨道/物体等操作

- 读取超大文件+图形化时导出

```

"AWT-EventQueue-0" #19 prio=6 os_prio=0 cpu=13968.75ms elapsed=86.25s tid=0x000001e17ddfc000 nid=0:
  java.lang.Thread.State: RUNNABLE
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$GroupTail.match(java.base@11.0.2/Pattern.java:4850)
    at java.util.regex.Pattern$CharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4293)
    at java.util.regex.Pattern$GroupHead.match(java.base@11.0.2/Pattern.java:4791)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$BmpCharProperty.match(java.base@11.0.2/Pattern.java:3951)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$GroupTail.match(java.base@11.0.2/Pattern.java:4850)
    at java.util.regex.Pattern$CharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4293)
    at java.util.regex.Pattern$GroupHead.match(java.base@11.0.2/Pattern.java:4791)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$BmpCharProperty.match(java.base@11.0.2/Pattern.java:3951)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$GroupTail.match(java.base@11.0.2/Pattern.java:4850)
    at java.util.regex.Pattern$CharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4293)
    at java.util.regex.Pattern$GroupHead.match(java.base@11.0.2/Pattern.java:4791)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$BmpCharProperty.match(java.base@11.0.2/Pattern.java:3951)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$GroupTail.match(java.base@11.0.2/Pattern.java:4850)
    at java.util.regex.Pattern$CharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4293)
    at java.util.regex.Pattern$GroupHead.match(java.base@11.0.2/Pattern.java:4791)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$BmpCharProperty.match(java.base@11.0.2/Pattern.java:3951)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$GroupTail.match(java.base@11.0.2/Pattern.java:4850)
    at java.util.regex.Pattern$CharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4293)
    at java.util.regex.Pattern$GroupHead.match(java.base@11.0.2/Pattern.java:4791)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$BmpCharProperty.match(java.base@11.0.2/Pattern.java:3951)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$GroupTail.match(java.base@11.0.2/Pattern.java:4850)
    at java.util.regex.Pattern$CharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4293)
    at java.util.regex.Pattern$GroupHead.match(java.base@11.0.2/Pattern.java:4791)
    at java.util.regex.Pattern$BmpCharPropertyGreedy.match(java.base@11.0.2/Pattern.java:4331)
    at java.util.regex.Pattern$BnM.match(java.base@11.0.2/Pattern.java:5588)
    at java.util.regex.Matcher.search(java.base@11.0.2/Matcher.java:1729)
    at java.util.regex.Matcher.find(java.base@11.0.2/Matcher.java:746)
    at iostrategy.BufferedIoStrategy.readFileAndCreateSystem(BufferedIoStrategy.java:79)
    at circularorbit.StellarSystem.readFileAndCreateSystem(StellarSystem.java:68)
  
```

- 删除轨道+重画 GUI 时导出（主要是 GUI 是 runnable）

```

"AWT-EventQueue-0" #19 prio=6 os_prio=0 cpu=172656.25ms elapsed=1074.95s tid=0x000001e17ddfc000 nid=0x69d0 runnable [0x00000060deaf
java.lang.Thread.State: RUNNABLE
    at sun.java2d.loops.FillSpans.FillSpans(java.desktop@11.0.2/Native Method)
    at sun.java2d.loops.FillSpans.FillSpans(java.desktop@11.0.2/FillSpans.java:82)
    at sun.java2d.pipe.LoopPipe.fillSpans(java.desktop@11.0.2/LoopPipe.java:332)
    at sun.java2d.pipe.LoopPipe.draw(java.desktop@11.0.2/LoopPipe.java:198)
    at sun.java2d.pipe.PixelToParallelogramConverter.draw(java.desktop@11.0.2/PixelToParallelogramConverter.java:148)
    at sun.java2d.SunGraphics2D.draw(java.desktop@11.0.2/SunGraphics2D.java:2498)
    at applications.StellarSystemGui$1.paint(StellarSystemGui.java:140)
    at javax.swing.JComponent.paintChildren(java.desktop@11.0.2/JComponent.java:907)
    - locked <0x0000000082f6cfff> (a java.awt.Component$AWTTreeLock)
    at javax.swing.JComponent.paint(java.desktop@11.0.2/JComponent.java:1083)
    at javax.swing.JComponent.paintChildren(java.desktop@11.0.2/JComponent.java:907)
    - locked <0x0000000082f6cfff> (a java.awt.Component$AWTTreeLock)
    at javax.swing.JComponent.paint(java.desktop@11.0.2/JComponent.java:1083)
    at javax.swing.JLayeredPane.paint(java.desktop@11.0.2/JLayeredPane.java:590)
    at javax.swing.JComponent.paintChildren(java.desktop@11.0.2/JComponent.java:907)
    - locked <0x0000000082f6cfff> (a java.awt.Component$AWTTreeLock)
    at javax.swing.JComponent.paint(java.desktop@11.0.2/JComponent.java:1083)
    at javax.swing.JComponent.paintToOffscreen(java.desktop@11.0.2/JComponent.java:5255)
    at javax.swing.RepaintManager$PaintManager.paintDoubleBufferedFPScales(java.desktop@11.0.2/RepaintManager.java:1707)
    at javax.swing.RepaintManager$PaintManager.paintDoubleBuffered(java.desktop@11.0.2/RepaintManager.java:1616)
    at javax.swing.RepaintManager$PaintManager.paint(java.desktop@11.0.2/RepaintManager.java:1556)
    at javax.swing.RepaintManager.paint(java.desktop@11.0.2/RepaintManager.java:1323)
    at javax.swing.JComponent._paintImmediately(java.desktop@11.0.2/JComponent.java:5203)
    at javax.swing.JComponent.paintImmediately(java.desktop@11.0.2/JComponent.java:5013)
    at javax.swing.RepaintManager$4.run(java.desktop@11.0.2/RepaintManager.java:865)
    at javax.swing.RepaintManager$4.run(java.desktop@11.0.2/RepaintManager.java:848)
    at java.security.AccessController.doPrivileged(java.base@11.0.2/Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(java.base@11.0.2/ProtectionDomain.java:85)
    at javax.swing.RepaintManager.paintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:848)
    at javax.swing.RepaintManager.paintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:823)
    at javax.swing.RepaintManager.prePaintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:772)
    at javax.swing.RepaintManager$ProcessingRunnable.run(java.desktop@11.0.2/RepaintManager.java:1890)
    at java.awt.event.InvocationEvent.dispatch(java.desktop@11.0.2/InvocationEvent.java:313)
    at java.awt.EventQueue.dispatchEventImpl(java.desktop@11.0.2/EventQueue.java:770)
    at java.awt.EventQueue$4.run(java.desktop@11.0.2/EventQueue.java:721)
    at java.awt.EventQueue$4.run(java.desktop@11.0.2/EventQueue.java:715)
    at java.security.AccessController.doPrivileged(java.base@11.0.2/Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(java.base@11.0.2/ProtectionDomain.java:85)

```

- 增加轨道即增加物体+GUI 重绘（主要是 GUI 是 runnable）

```

"AWT-EventQueue-0" #19 prio=6 os_prio=0 cpu=202328.13ms elapsed=1320.93s tid=0x000001e17ddfc000 nid=0x69d0 runnable [0x00000060deaf
java.lang.Thread.State: RUNNABLE
    at sun.java2d.loops.FillSpans.FillSpans(java.desktop@11.0.2/Native Method)
    at sun.java2d.loops.FillSpans.FillSpans(java.desktop@11.0.2/FillSpans.java:82)
    at sun.java2d.pipe.LoopPipe.fillSpans(java.desktop@11.0.2/LoopPipe.java:332)
    at sun.java2d.pipe.LoopPipe.draw(java.desktop@11.0.2/LoopPipe.java:198)
    at sun.java2d.pipe.PixelToParallelogramConverter.draw(java.desktop@11.0.2/PixelToParallelogramConverter.java:148)
    at sun.java2d.SunGraphics2D.draw(java.desktop@11.0.2/SunGraphics2D.java:2498)
    at applications.StellarSystemGui$1.paint(StellarSystemGui.java:140)
    at javax.swing.JComponent.paintToOffscreen(java.desktop@11.0.2/JComponent.java:5255)
    at javax.swing.RepaintManager$PaintManager.paintDoubleBufferedFPScales(java.desktop@11.0.2/RepaintManager.java:1707)
    at javax.swing.RepaintManager$PaintManager.paintDoubleBuffered(java.desktop@11.0.2/RepaintManager.java:1616)
    at javax.swing.RepaintManager$PaintManager.paint(java.desktop@11.0.2/RepaintManager.java:1556)
    at javax.swing.RepaintManager.paint(java.desktop@11.0.2/RepaintManager.java:1323)
    at javax.swing.JComponent._paintImmediately(java.desktop@11.0.2/JComponent.java:5203)
    at javax.swing.JComponent.paintImmediately(java.desktop@11.0.2/JComponent.java:5013)
    at javax.swing.RepaintManager$4.run(java.desktop@11.0.2/RepaintManager.java:865)
    at javax.swing.RepaintManager$4.run(java.desktop@11.0.2/RepaintManager.java:848)
    at java.security.AccessController.doPrivileged(java.base@11.0.2/Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(java.base@11.0.2/ProtectionDomain.java:85)
    at javax.swing.RepaintManager.paintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:848)
    at javax.swing.RepaintManager.paintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:823)
    at javax.swing.RepaintManager.prePaintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:772)
    at javax.swing.RepaintManager$ProcessingRunnable.run(java.desktop@11.0.2/RepaintManager.java:1890)
    at java.awt.event.InvocationEvent.dispatch(java.desktop@11.0.2/InvocationEvent.java:313)
    at java.awt.EventQueue.dispatchEventImpl(java.desktop@11.0.2/EventQueue.java:770)
    at java.awt.EventQueue$4.run(java.desktop@11.0.2/EventQueue.java:721)
    at java.awt.EventQueue$4.run(java.desktop@11.0.2/EventQueue.java:715)
    at java.security.AccessController.doPrivileged(java.base@11.0.2/Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(java.base@11.0.2/ProtectionDomain.java:85)
    at java.awt.EventQueue.dispatchEvent(java.desktop@11.0.2/EventQueue.java:740)
    at java.awt.EventDispatchThread.pumpOneEventForFilters(java.desktop@11.0.2/EventDispatchThread.java:203)
    at java.awt.EventDispatchThread.pumpEventsForFilter(java.desktop@11.0.2/EventDispatchThread.java:124)
    at java.awt.EventDispatchThread.pumpEventsForHierarchy(java.desktop@11.0.2/EventDispatchThread.java:113)
    at java.awt.EventDispatchThread.pumpEvents(java.desktop@11.0.2/EventDispatchThread.java:109)
    at java.awt.EventDispatchThread.pumpEvents(java.desktop@11.0.2/EventDispatchThread.java:101)
    at java.awt.EventDispatchThread.run(java.desktop@11.0.2/EventDispatchThread.java:90)

```

- 天体动态化（主要是 GUI 是 runnable）


```

"AWT-EventQueue-0" #19 prio=6 os_prio=0 cpu=333890.63ms elapsed=1708.15s tid=0x000001e17ddfc000 nid=0x69d0 runnable [0x00000060des
java.lang.Thread.State: RUNNABLE
    at sun.java2d.loops.FillSpans.fillSpans(java.desktop@11.0.2/Native Method)
    at sun.java2d.loops.FillSpans.fillSpans(java.desktop@11.0.2/FillSpans.java:82)
    at sun.java2d.pipe.LoopPipe.fillSpans(java.desktop@11.0.2/LoopPipe.java:332)
    at sun.java2d.pipe.LoopPipe.draw(java.desktop@11.0.2/LoopPipe.java:198)
    at sun.java2d.pipe.PixelToParallelogramConverter.draw(java.desktop@11.0.2/PixelToParallelogramConverter.java:148)
    at sun.java2d.SunGraphics2D.draw(java.desktop@11.0.2/SunGraphics2D.java:2498)
    at applications.StellarSystemGui$1.paint(StellarSystemGui.java:140)
    at javax.swing.JComponent.paintToOffscreen(java.desktop@11.0.2/JComponent.java:5255)
    at javax.swing.RepaintManager$PaintManager.paintDoubleBufferedFPScales(java.desktop@11.0.2/RepaintManager.java:1707)
    at javax.swing.RepaintManager$PaintManager.paintDoubleBuffered(java.desktop@11.0.2/RepaintManager.java:1616)
    at javax.swing.RepaintManager$PaintManager.paint(java.desktop@11.0.2/RepaintManager.java:1556)
    at javax.swing.RepaintManager.paint(java.desktop@11.0.2/RepaintManager.java:1323)
    at javax.swing.JComponent._paintImmediately(java.desktop@11.0.2/JComponent.java:5203)
    at javax.swing.RepaintManager$4.run(java.desktop@11.0.2/RepaintManager.java:865)
    at javax.swing.RepaintManager$4.run(java.desktop@11.0.2/RepaintManager.java:848)
    at java.security.AccessController.doPrivileged(java.base@11.0.2/Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(java.base@11.0.2/ProtectionDomain.java:85)
    at javax.swing.RepaintManager.paintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:848)
    at javax.swing.RepaintManager.paintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:823)
    at javax.swing.RepaintManager.prePaintDirtyRegions(java.desktop@11.0.2/RepaintManager.java:772)
    at javax.swing.RepaintManager$ProcessingRunnable.run(java.desktop@11.0.2/RepaintManager.java:1890)
    at java.awt.event.InvocationEvent.dispatch(java.desktop@11.0.2/InvocationEvent.java:313)
    at java.awt.EventQueue.dispatchEventImpl(java.desktop@11.0.2/EventQueue.java:770)
    at java.awt.EventQueue$4.run(java.desktop@11.0.2/EventQueue.java:721)
    at java.awt.EventQueue$4.run(java.desktop@11.0.2/EventQueue.java:715)
    at java.security.AccessController.doPrivileged(java.base@11.0.2/Native Method)
    at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(java.base@11.0.2/ProtectionDomain.java:85)
    at java.awt.EventQueue.dispatchEvent(java.desktop@11.0.2/EventQueue.java:740)
    at java.awt.EventDispatchThread.pumpOneEventForFilters(java.desktop@11.0.2/EventDispatchThread.java:203)
    at java.awt.EventDispatchThread.pumpEventsForFilter(java.desktop@11.0.2/EventDispatchThread.java:124)
    at java.awt.EventDispatchThread.pumpEventsForHierarchy(java.desktop@11.0.2/EventDispatchThread.java:113)
    at java.awt.EventDispatchThread.pumpEvents(java.desktop@11.0.2/EventDispatchThread.java:109)
    at java.awt.EventDispatchThread.pumpEvents(java.desktop@11.0.2/EventDispatchThread.java:101)
    at java.awt.EventDispatchThread.run(java.desktop@11.0.2/EventDispatchThread.java:90)

```

3.5.6 使用设计模式进行代码性能优化

- **StringBuilder 优化**: 将需要大量进行字符串连接运算的地方改用 StringBuilder 类，如截图。

```

StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append(
    "Stellar ::= <" + stellarSystem.getCentralPoint().getName() + ","
    + stellarSystem
    .enotationTransform(stellarSystem.getCentralPoint().getRadius())
    + "," + stellarSystem.enotationTransform(
    stellarSystem.getCentralPoint().getMess())
    + ">\n"); // 恒星行
// 下面书写行星行

```

- **StringBuilder 进一步优化**: StringBuilder 类的 append 方法传入的参数中也不可以使用字符串的拼接操作，全部改为 append 操作如截图。

```

stringBuilder.append("Planet ::= <" + planet.getPlanetName() + ","
    + planet.getPlanetState() + "," + planet.getPlanetColor() + ","
    + stellarSystem.enotationTransform(planet.getPlanetRadius()) + ","
    + stellarSystem.enotationTransform(planet.getTrackRadius()) + ","
    + stellarSystem.enotationTransform(planet.getRevolutionSpeed()) + ","
    + planet.getRevolutionDirection() + "," + planet.getAngle() + ">\n");

```

图一：可以将传入 append 方法中的带有+的字串全改为 append 操作

```

stringBuilder.append("Stellar ::= <")
    → .append(stellarSystem.getCentralPoint().getName()).append(",")
    → .append(stellarSystem
        .enotationTransform(stellarSystem.getCentralPoint().getRadius()))
    → .append(",")
    → .append(stellarSystem
        .enotationTransform(stellarSystem.getCentralPoint().getMess()))
    → .append(">\n"); // 恒星行

```

- **实现原型设计模式**: 让 CircularOrbit 类继承 Cloneable 接口，并实现 deep copy 并在三个具体的子类中进行了相应的实现。如截图

```

@SuppressWarnings("unchecked")
@Override public Object clone()
    throws CloneNotSupportedException {
    ConcreteCircularOrbit<L, E> circularOrbit =
        (ConcreteCircularOrbit<L, E>) super.clone();
    circularOrbit.allTracks = new HashMap<> (this.allTracks);
    circularOrbit.allRadius = new HashMap<> (this.allRadius);
    circularOrbit.allOrders = new HashMap<> (this.allOrders);
    return circularOrbit;
}

```

- 一些方法的私有属性不进行初始化：设为 null，如截图

```

private File readFile = null;
private String writeFilePath = null;

```

- 优化社交系统先广确定系统所有朋友所处轨道的方法：原本是对系统中所有的朋友均调用 getDistance 方法，时间复杂度为 n^2 ；现在改为只调用一次，但在这一次里面充分利用每一次搜索的结果，只进行一次广搜，时间复杂度为 $o(n)$ 。如截图

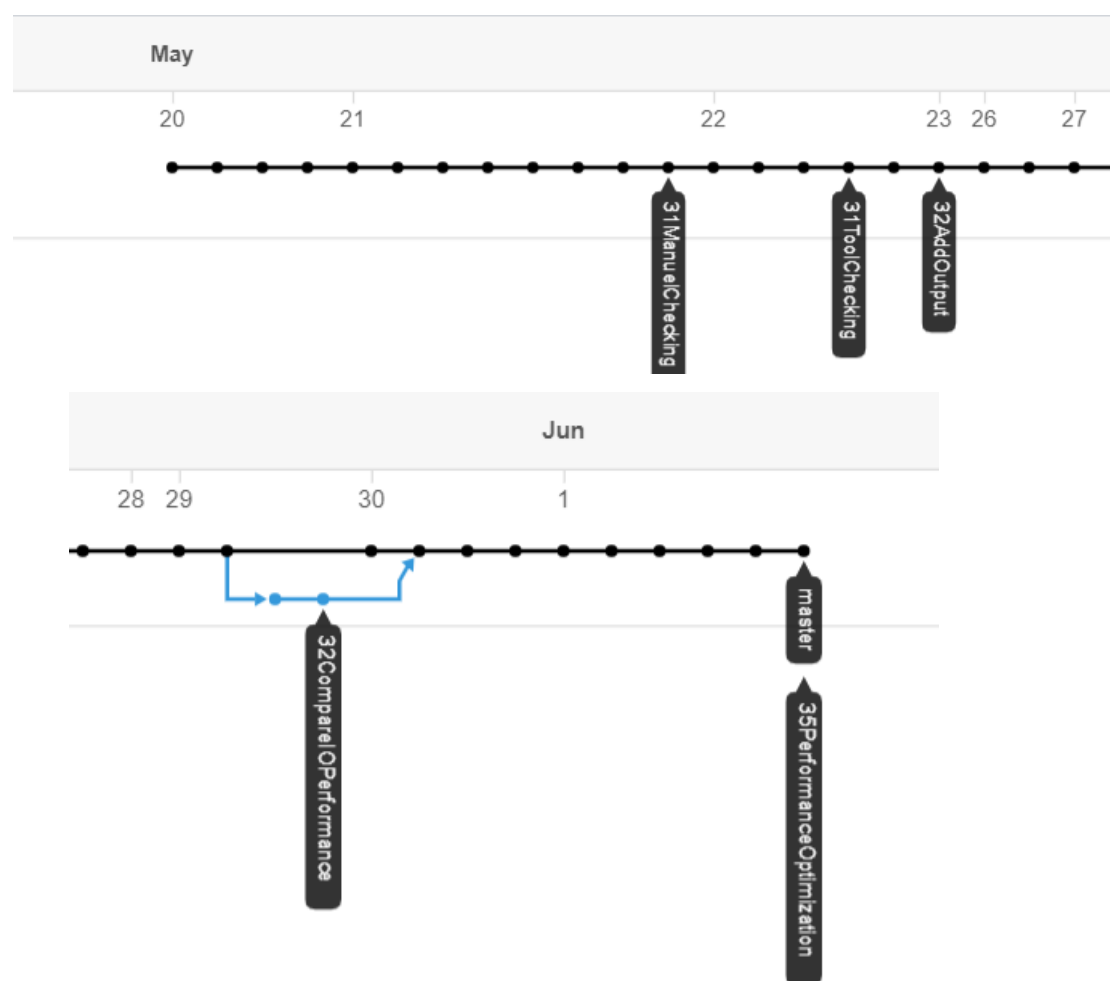
```

public void getAllDistance() {
    Friend centralUser = getCentralPoint();
    Map<Friend, Boolean> visited = new HashMap<>(); // 用于判断是否被访问
    for (Friend friend : allFriends) { // 将所有 person 标记为未被访问
        visited.put(friend, false);
    }
    visited.put(centralUser, true);
    Queue<Friend> queue = new LinkedBlockingQueue<>(); // 先广要用队列来做
    queue.add(centralUser);
    Map<Friend, Integer> distance = new HashMap<>(); // 用于记录最短距离
    distance.put(centralUser, 0);
    while (!queue.isEmpty()) { // 循环直到队列为空
        Friend head = queue.poll(); // 得到队首元素，并将其出队
        Friend friend = head.getFriend(1); // 得到与其有关的第一个人
        int i = 0;
        int size = head.getAllFriends().size();
        while (friend != null) { // 循环直到无人与 head 有直接关系
            if (!visited.get(friend)) { // 若 friend 未被访问
                int dis = distance.get(head) + 1;
                visited.put(friend, true);
                distance.put(friend, dis);
                friend.setTrackRadius(dis);
                this.addFriendOnTrack(friend);
                queue.add(friend); // 将当前 friend 入队
            } else if (++i < size) { // 继续寻找与其有关系的人
                friend = head.getAllFriends().get(i);
            } else {
                break;
            }
        }
    }
}

```

3.6 Git 仓库结构

实验各分支之间的关系通过 github 的 insight 界面显示，如下图。



4 实验进度记录

日期	时间段	计划任务	实际完成情况
2019-05-20	下午	完成仓库配置	完成
2019-05-20	晚上	完成插件配置	完成
2019-05-21	下午+晚上	完成 checkStyle	完成
2019-05-23	下午+晚上	完成 3.1 节	完成
2019-05-24	上午+晚上	完成 3.2 节	未完成，完成模式的框架
2019-05-26	全天	完成 3.2 节	完成各种策略的书写
2019-05-	下午+晚上	完成 3.3 节	遇到问题，无法

27			解决
2019-05-28	晚上	完成 3.3 节	仍无法解决问题
2019-05-29	晚上	完成 3.4 节	完成
2019-05-30	下午+晚上	完成 3.5 节	设计模式部分未完成
2019-05-31	下午	完成 3.5 节	完成
2019-06-01	全天	完成 3.3 节	完成

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
gc 的终端显示中没有 Full GC	jdk 版本问题，选择安装 JDK8
社交系统建立系统超级慢	定位到问题在广搜上，优化了调用方式和建立系统方式
维护了许多数据结构	通过.jprof 的分析逐渐减少无用的数据结构

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验教训

- 对 java 的命令行操作不太熟悉
- 对设计模式的应用还是不太熟练

6.2 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？
- (2) 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？
- (3) 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？

- (4) 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？
- (5) 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？
- (6) 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？
- (7) 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？
- (8) 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？
- (9) 关于本实验的工作量、难度、deadline。
- (10) 到目前为止，你对《软件构造》课程的意见与建议。
 - 运行起来很美更重要；
 - SpotBugs 会提高我的代码质量，但感觉 CheckStyle 帮助不大，因为有这么多的规范，只需要选中一种规范，就可以达到程序的良好可读性；
 - 我感觉 java 的自动垃圾回收机制解放了程序员的双手，但是感觉自动垃圾回收机制还是需要程序员不断地根据具体的代码进行相应的调整；
 - Memory Dump 和 VisualVM 确实好用呀
 - 本实验对于一个 java 命令行基础较差的我来说，感觉工作量有点大；感觉难度适中。