



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	林之浩
学号	1170300817
班号	1703008
电子邮件	630073498@qq.com
手机号码	18065053516

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	2
3.1 Static Program Analysis	2
3.1.1 人工代码走查 (walkthrough)	2
3.1.1.1 错误 1: 命名不规范	2
3.1.1.2 错误 2: 注释不规范	2
3.1.1.3 错误 3: 大括号	3
3.1.2 使用 CheckStyle 和 SpotBugs 进行静态代码分析	3
3.1.2.1 注释	3
3.1.2.2 导入	3
3.1.2.3 单行字符数	3
3.1.2.4 Javadoc	4
3.1.2.5 空格/Tab 错误	4
3.2 Java I/O Optimization	4
3.2.1 多种 I/O 实现方式	4
3.2.2 多种 I/O 实现方式的效率对比分析	7
3.3 Java Memory Management and Garbage Collection (GC)	9
3.3.1 使用-verbose:gc 参数	9
3.3.2 用 jstat 命令行工具的-gc 和-gcutil 参数	10
3.3.3 使用 jmap -heap 命令行工具	12
3.3.4 使用 jmap -clstats 命令行工具	13
3.3.5 使用 jmap -permstat 命令行工具	13
3.3.6 使用 JMC/JFR、jconsole 或 VisualVM 工具	13
3.3.7 分析垃圾回收过程	14
3.3.8 配置 JVM 参数并发现优化的参数配置	15
3.4 Dynamic Program Profiling	16
3.4.1 使用 JMC 或 VisualVM 进行 CPU Profiling	16
3.4.2 使用 VisualVM 进行 Memory profiling	17
3.5 Memory Dump Analysis and Performance Optimization	17
3.5.1 内存导出	17

3.5.2 使用 MAT 分析内存导出文件	18
3.5.3 发现热点/瓶颈并改进、改进前后的性能对比分析	23
3.5.4 在 MAT 内使用 OQL 查询内存导出	24
3.5.5 观察 jstack/jcmd 导出程序运行时的调用栈	29
3.5.6 使用设计模式进行代码性能优化	30
3.6 Git 仓库结构	32
4 实验进度记录	33
5 实验过程中遇到的困难与解决途径	33
6 实验过程中收获的经验、教训、感想	33
6.1 实验过程中收获的经验教训	33
6.2 针对以下方面的感受	33

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

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。

从指导书上的网址下载的 MAT，visualvm 用的是自带的（从 cmd 进入输入 jvisualvm 即可）

在这里给出你的 GitHub Lab5 仓库的 URL 地址（Lab5-学号）。

<https://github.com/ComputerScienceHIT/Lab5-1170300817>

3 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！）。

3.1 Static Program Analysis

3.1.1 人工代码走查 (walkthrough)

3.1.1.1 错误 1：命名不规范

包名应该全部是小写字符

- 田 applications
- 田 centralobject
- 田 circularorbit
- 田 circularorbithelper
- 田 difference
- 田 exception
- 田 logrecord
- 田 phsicalobject
- 田 starter
- 田 track

局部变量命名应该符合： `^[a-z]([a-z0-9][a-zA-Z0-9]*)?$`

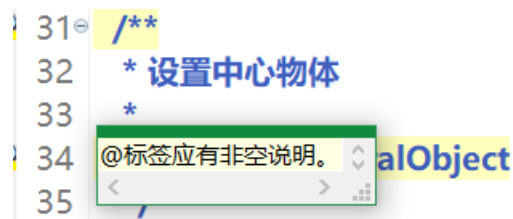
方法名需要符合： `^[a-z][a-z0-9][a-zA-Z0-9_]*$`。

类的命名应该符合 `^[A-Z][a-zA-Z0-9]*$`

- 田 applications
 - 田 atomstructure
 - > 田 AtomCircularOrbit.java
 - > 田 AtomCircularOrbitBuilder.java
 - > 田 AtomGame.java
 - > 田 Memento.java
 - > 田 Particle.java
 - > 田 TransitCareTaker.java

3.1.1.2 错误 2：注释不规范

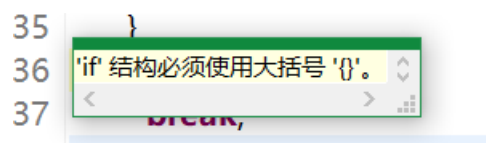
所有@参数后面都必须有说明，不能为空。



尽可能地使用@Override 标记重写。

3.1.1.3 错误 3：大括号

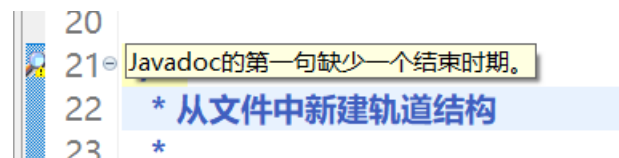
if, else, for, do 和 while 语句后面必须跟有大括号。



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

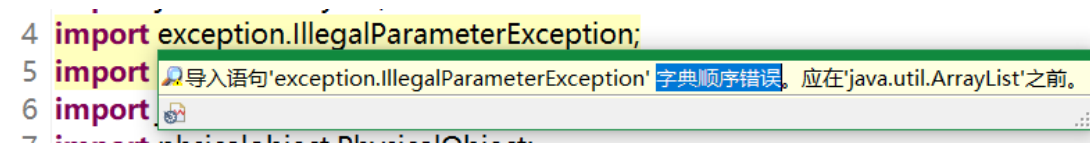
3.1.2.1 注释

Javadoc 第一句描述必须要有一个句点



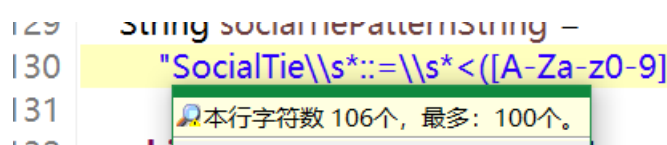
3.1.2.2 导入

导入语句要遵守字典顺序。



3.1.2.3 单行字符数

每行字符不超 100 个。用+连接长字符串。

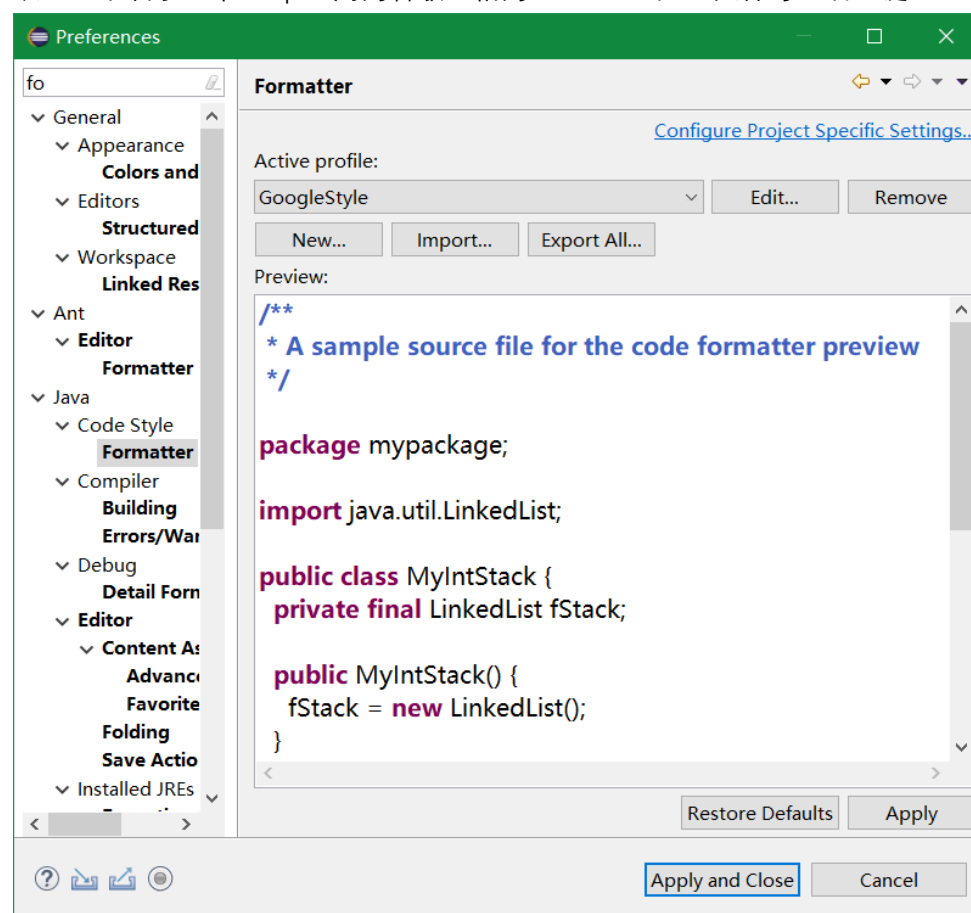


3.1.2.4 Javadoc

每个方法都要有文档，除了简单的 getter 和 setter 方法。

3.1.2.5 空格/Tab 错误

从网上下载了一个 eclipse 用的谷歌风格的 formatter 配置文件导入后一键 format 就行了。



区别：手动检查只能发现一些很明显的错误，明显用工具检查更方便快速，但是手动检查加强一下影响能养成好习惯。

3.2 Java I/O Optimization

3.2.1 多种 I/O 实现方式

实现了 Stream, Reader/Writer, Buffer 三种读写方式，具体实现：先声明两个接口，分别是读写策略。

```

1 package iostrategy;
2
3 import java.io.IOException;
4
5 public interface InputStrategy {
6
7     /**
8      * 当前策略的readline函数.
9      *
10     * @return readline结果
11     * @throws IOException 文件错误
12     */
13     public String readLine() throws IOException;
14
15     /**
16     * 关闭流.
17     */
18     public void close() throws IOException;
19 }

```

```

1 package iostrategy;
2
3 import java.io.IOException;
4
5 public interface OutputStrategy {
6
7     /**
8     * 写入函数.
9     *
10    * @param string 写入的内容
11    * @throws IOException 文件错误
12    */
13    public void write(String string) throws IOException;
14
15    /**
16    * 关闭流.
17    */
18    public void close() throws IOException;
19 }

```

随后实现以下六个子类，分别实现各自的 readline 方法和 close 方法。

```

> BufferedInputStrategy.java
> BufferedOutputStrategy.java
> InputStrategy.java
> OutputStrategy.java
> ReaderInputStrategy.java
> StreamInputStrategy.java
> StreamOutputStrategy.java
> WriterOutputStrategy.java

```

分别使用三种方法实现接口，

使用时，选择要使用的读写策略，使用文件名构造一个新的 strategy，之后传入构造轨道结构的函数当做输入流。

切换只需要在控制台选择


```
try {
    System.out.println("输入需要读取的文件名：例如src/txt/TrackGame.txt\n");
    String filePath = reader.readLine();
    System.out.println("选择读入策略：");
    System.out.println("1.\tBuffer");
    System.out.println("2.\tStream");
    System.out.println("3.\tReader");
    input = reader.readLine();
    InputStrategy strategy = null;
    switch (input) {
        case "1":// Buffer
            strategy = new BufferedInputStrategy(filePath);
            break;
        case "2":// Stream
            strategy = new StreamInputStrategy(filePath);
            break;
        case "3":// Reader
            strategy = new ReaderInputStrategy(filePath);
            break;
        default:
            System.out.println("策略输入错误");
            break;
    }
}
```

控制台演示：

```
1. 读取文件构造系统
2. 跃迁
3. 回退
4. 可视化
5. 打印轨道结构
6. 增加新轨道
7. 增加新电子
8. 删除电子
9. 删除整条轨道
10. 计算熵值
11. 日志查询
12. 文件输出
end. 结束
1
输入需要读取的文件名：例如src/txt/AtomicStructure.txt
src/txt/AtomicStructure.txt
选择读入策略：
1. Buffer
2. Reader
3. Stream
1
track0上有：2个电子
track1上有：8个电子
track2上有：18个电子
track3上有：8个电子
track4上有：1个电子
```

3.2.2 多种 I/O 实现方式的效率对比分析

时间计算方式：

在读取文件的循环的头尾加上

```
long startTime = System.currentTimeMillis();
```

```
long endTime = System.currentTimeMillis();
```

最后输出：

```
System.out.println("读取文件时间： " + (endTime - startTime) + "ms");
```

这里以 socialnetwork 为例：（读取的是 79 万行的测试文件，为了区分以前的，文件名后面加了 Big）

分别测试三种读写方式的用时：

读入：

end. 结束

1

输入需要读取的文件名：例如src/txt/SocialNetworkCircleBig.txt

src/txt/SocialNetworkCircleBig.txt

选择读入策略：

1. Buffer
2. Stream
3. Reader

1

读取文件时间： 5629ms

src/txt/SocialNetworkCircleBig.txt

选择读入策略：

1. Buffer
2. Stream
3. Reader

2

读取文件时间： 52403ms

构建关系时间： 2114ms

```
src/txt/SocialNetworkCircleBig.txt
```

选择读入策略：

1. Buffer
2. Stream
3. Reader

3

读取文件时间： 6474ms

写出：

```
1.2
```

选择输出策略：

1. Buffer
2. Stream
3. Writer

1

文件输出时间： 2090ms

1. Buffer
2. Stream
3. Writer

2

文件输出时间： 153610ms

1. Buffer
2. Stream
3. Writer

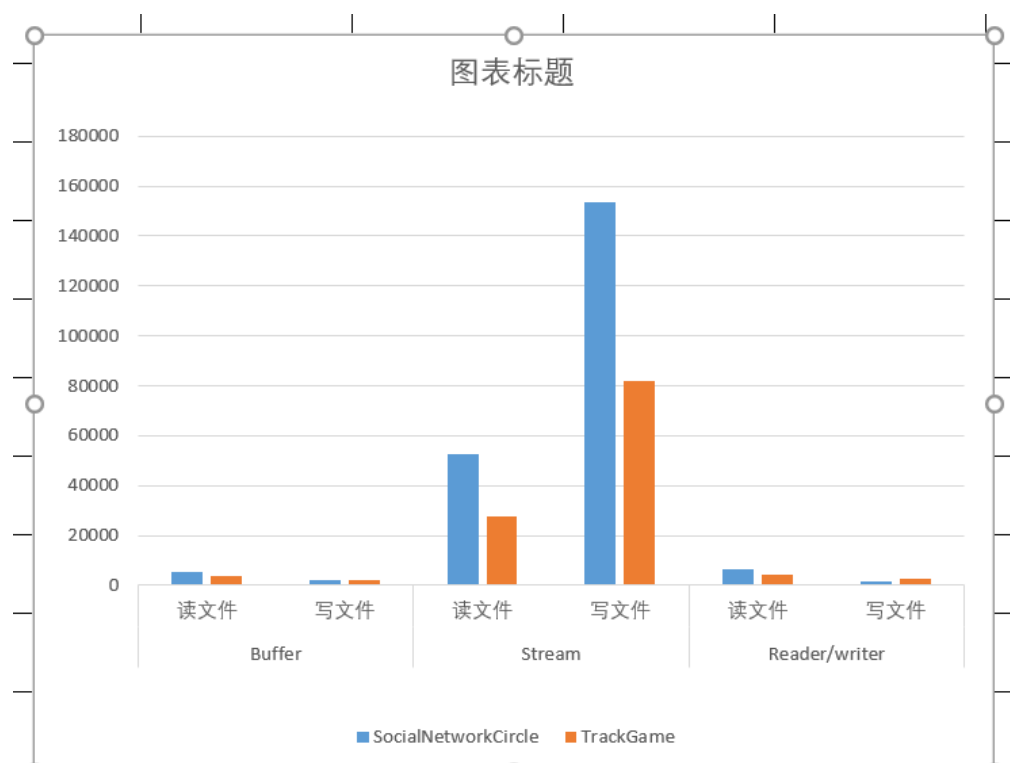
3

文件输出时间： 1762ms

数据统计：

单位: ms		SocialNetworkCircle	TrackGame
Buffer	读文件	5629	3558
	写文件	2090	2164
Stream	读文件	52403	27452
	写文件	153610	82022
Reader/writer	读文件	6474	4296
	写文件	1760	2617

图表：



3.3 Java Memory Management and Garbage Collection (GC)

3.3.1 使用-verbose:gc 参数

使用 SocialNetWork 做测试，读入大文件：

```
[26.571s][info ][gc] GC(38) Pause Young (Allocation Failure) 528M->221M(582M) 15.824ms
[26.723s][info ][gc] GC(39) Pause Young (Allocation Failure) 534M->226M(583M) 15.996ms
[26.873s][info ][gc] GC(40) Pause Young (Allocation Failure) 541M->231M(583M) 16.330ms
[27.023s][info ][gc] GC(41) Pause Young (Allocation Failure) 546M->237M(584M) 16.764ms
[27.170s][info ][gc] GC(42) Pause Young (Allocation Failure) 554M->242M(584M) 17.081ms
[27.329s][info ][gc] GC(43) Pause Young (Allocation Failure) 559M->248M(585M) 16.011ms
[27.477s][info ][gc] GC(44) Pause Young (Allocation Failure) 567M->253M(585M) 15.557ms
[27.630s][info ][gc] GC(45) Pause Young (Allocation Failure) 572M->259M(586M) 16.938ms
[28.511s][info ][gc] GC(46) Pause Full (Ergonomics) 259M->246M(703M) 880.919ms
[28.659s][info ][gc] GC(47) Pause Young (Allocation Failure) 567M->252M(704M) 10.935ms
[28.809s][info ][gc] GC(48) Pause Young (Allocation Failure) 574M->258M(701M) 14.993ms
[28.974s][info ][gc] GC(49) Pause Young (Allocation Failure) 580M->263M(704M) 18.533ms
[29.134s][info ][gc] GC(50) Pause Young (Allocation Failure) 585M->269M(704M) 16.450ms
[29.293s][info ][gc] GC(51) Pause Young (Allocation Failure) 591M->274M(704M) 15.511ms
[29.451s][info ][gc] GC(52) Pause Young (Allocation Failure) 597M->280M(704M) 15.711ms
[29.605s][info ][gc] GC(53) Pause Young (Allocation Failure) 603M->285M(705M) 14.817ms
[29.765s][info ][gc] GC(54) Pause Young (Allocation Failure) 609M->291M(705M) 16.544ms
```

由于本人电脑内存有 24G，jvm 默认最大 heapsize 默认最大内存是物理内存的 1/4，所以修改了最大堆内存

使用：-Xmx1024m。

调整最大 heapsize 为 1024MB，使用 trackgame 的大文件反复读取多次之后终于是出现了 full GC 的情况，可以看见：

1. full gc 的出现频率远远低于 minor gc。
2. 耗费的时间上看，minor gc 的用时也远远低于 full gc，其中的原理也显而易见：full gc 需要整理年轻代，老年代和永久代，代价自然很高

3.3.2 用 jstat 命令行工具的 -gc 和 -gcutil 参数

使用 jps 命令获取进程 ID：

```
C:\Users\Administrator>jps
9024 Jps
1256
2652 TrackGame
```

使用 -gc 参数：

```
C:\Users\Administrator>jstat -gc 3184 50 200
S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT CGC CGCT GCT
```

16384.0	16384.0	0.0	3560.0	98304.0	65503.2	262144.0	16.0	9216.0	8861.2	1024.0	826.4	1	0.004	0	0.000	-	-	0.004
16384.0	16384.0	5528.0	0.0	98304.0	69849.3	262144.0	24.0	9216.0	8864.8	1024.0	826.4	2	0.008	0	0.000	-	-	0.008
16384.0	16384.0	0.0	7752.0	98304.0	64315.1	262144.0	32.0	9216.0	8866.5	1024.0	826.4	3	0.012	0	0.000	-	-	0.012
16384.0	16384.0	9736.0	0.0	196608.0	74016.4	262144.0	40.0	9216.0	8878.2	1024.0	826.4	4	0.018	0	0.000	-	-	0.018
16384.0	16384.0	9736.0	0.0	196608.0	148032.8	262144.0	40.0	9216.0	8878.2	1024.0	826.4	4	0.018	0	0.000	-	-	0.018
16384.0	16384.0	9736.0	0.0	196608.0	194780.5	262144.0	40.0	9216.0	8878.2	1024.0	826.4	5	0.018	0	0.000	-	-	0.018
15872.0	16384.0	0.0	14152.0	196608.0	91169.9	262144.0	40.0	9216.0	8881.8	1024.0	826.4	5	0.027	0	0.000	-	-	0.027
15872.0	19968.0	15848.0	0.0	309248.0	0.0	262144.0	2136.0	9216.0	8897.6	1024.0	826.4	6	0.036	0	0.000	-	-	0.036
15872.0	19968.0	15848.0	0.0	309248.0	111935.8	262144.0	2136.0	9216.0	8897.6	1024.0	826.4	6	0.036	0	0.000	-	-	0.036
15872.0	19968.0	15848.0	0.0	309248.0	228711.3	262144.0	2136.0	9216.0	8897.6	1024.0	826.4	6	0.036	0	0.000	-	-	0.036
15872.0	19968.0	15848.0	0.0	309248.0	277379.7	262144.0	2136.0	9216.0	8897.6	1024.0	826.4	6	0.036	0	0.000	-	-	0.036
21504.0	18432.0	0.0	18400.0	309248.0	12300.8	262144.0	6888.2	9216.0	8901.6	1024.0	826.4	7	0.052	0	0.000	-	-	0.052
21504.0	18432.0	0.0	18400.0	309248.0	135308.2	262144.0	6888.2	9216.0	8901.6	1024.0	826.4	7	0.052	0	0.000	-	-	0.052
21504.0	18432.0	0.0	18400.0	309248.0	246016.1	262144.0	6888.2	9216.0	8901.6	1024.0	826.4	7	0.052	0	0.000	-	-	0.052
21504.0	26112.0	21152.0	0.0	296960.0	11835.7	262144.0	10304.2	9216.0	8901.6	1024.0	826.4	8	0.069	0	0.000	-	-	0.069
21504.0	26112.0	21152.0	0.0	296960.0	130183.8	262144.0	10304.2	9216.0	8901.6	1024.0	826.4	8	0.069	0	0.000	-	-	0.069
21504.0	26112.0	21152.0	0.0	296960.0	242616.1	262144.0	10304.2	9216.0	8901.6	1024.0	826.4	8	0.069	0	0.000	-	-	0.069
27648.0	20480.0	0.0	20416.0	296960.0	0.0	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	0.0	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	0.0	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	0.0	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	0.0	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	0.0	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	23699.9	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	142196.4	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	9	0.084	0	0.000	-	-	0.084
27648.0	20480.0	0.0	20416.0	296960.0	254768.1	262144.0	17080.2	9216.0	8901.6	1024.0	826.4	10	0.084	0	0.000	-	-	0.084
27648.0	26624.0	11968.0	0.0	294912.0	35332.8	262144.0	31456.2	9216.0	8901.6	1024.0	826.4	10	0.100	0	0.000	-	-	0.100
27648.0	26624.0	11968.0	0.0	294912.0	119825.3	262144.0	31456.2	9216.0	8901.6	1024.0	826.4	10	0.100	0	0.000	-	-	0.100
27648.0	26624.0	11968.0	0.0	294912.0	237603.3	262144.0	31456.2	9216.0	8901.6	1024.0	826.4	11	0.100	0	0.000	-	-	0.100
25600.0	10240.0	0.0	10048.0	294912.0	23567.8	262144.0	43400.2	9216.0	8901.6	1024.0	826.4	11	0.116	0	0.000	-	-	0.116
25600.0	10240.0	0.0	10048.0	294912.0	141407.2	262144.0	43400.2	9216.0	8901.6	1024.0	826.4	11	0.116	0	0.000	-	-	0.116
25600.0	25600.0	6048.0	0.0	294912.0	0.0	262144.0	53376.2	9216.0	8901.6	1024.0	826.4	12	0.129	0	0.000	-	-	0.129
25600.0	25600.0	6048.0	0.0	294912.0	53046.0	262144.0	53376.2	9216.0	8901.6	1024.0	826.4	12	0.129	0	0.000	-	-	0.129
25600.0	25600.0	6048.0	0.0	294912.0	176818.8	262144.0	53376.2	9216.0	8901.6	1024.0	826.4	12	0.129	0	0.000	-	-	0.129
25088.0	25600.0	0.0	6048.0	294912.0	0.0	262144.0	59320.2	9216.0	8901.6	1024.0	826.4	13	0.144	0	0.000	-	-	0.144
25088.0	25600.0	0.0	6048.0	294912.0	82534.6	262144.0	59320.2	9216.0	8901.6	1024.0	826.4	13	0.144	0	0.000	-	-	0.144
25088.0	25600.0	0.0	6048.0	294912.0	206335.2	262144.0	59320.2	9216.0	8901.6	1024.0	826.4	14	0.144	0	0.000	-	-	0.158
25088.0	25088.0	6016.0	0.0	297984.0	119150.5	262144.0	65256.2	9216.0	8901.6	1024.0	826.4	14	0.158	0	0.000	-	-	0.158
25088.0	25088.0	6016.0	0.0	297984.0	244259.5	262144.0	65256.2	9216.0	8901.6	1024.0	826.4	14	0.158	0	0.000	-	-	0.158
24064.0	25088.0	0.0	6048.0	297984.0	29790.5	262144.0	71208.2	9216.0	8901.6	1024.0	826.4	15	0.172	0	0.000	-	-	0.172
24064.0	25088.0	0.0	6048.0	297984.0	148953.6	262144.0	71208.2	9216.0	8901.6	1024.0	826.4	15	0.172	0	0.000	-	-	0.172
24064.0	25088.0	0.0	6048.0	297984.0	274074.9	262144.0	71208.2	9216.0	8901.6	1024.0	826.4	15	0.172	0	0.000	-	-	0.172
24064.0	23552.0	6080.0	0.0	299520.0	59892.5	262144.0	77224.2	9216.0	8901.6	1024.0	826.4	16	0.187	0	0.000	-	-	0.187
24064.0	23552.0	6080.0	0.0	299520.0	179675.7	262144.0	77224.2	9216.0	8901.6	1024.0	826.4	16	0.187	0	0.000	-	-	0.187
24064.0	23552.0	6080.0	608.0	299520.0	299520.0	262144.0	77728.2	9216.0	8901.6	1024.0	826.4	17	0.187	0	0.000	-	-	0.187

使用-gcutil 参数:

C:\Users\Administrator>jstat -gcutil 2652 50 200													
S0	S1	E	O	M	CCS	YGC	YGCT	FGC	FGCT	CGC	CGCT	GCT	
0.00	21.44	35.28	0.01	96.10	80.70	1	0.004	0	0.000	-	-	0.004	
34.13	0.00	1.97	0.01	96.14	80.70	2	0.007	0	0.000	-	-	0.007	
0.00	47.41	5.95	0.01	96.16	80.70	3	0.012	0	0.000	-	-	0.012	
59.23	0.00	5.94	0.02	96.29	80.70	4	0.018	0	0.000	-	-	0.018	
59.23	0.00	59.45	0.02	96.29	80.70	4	0.018	0	0.000	-	-	0.018	
59.23	0.00	91.15	0.02	96.29	80.70	4	0.018	0	0.000	-	-	0.018	
0.00	86.18	2.64	0.02	96.33	80.70	5	0.027	0	0.000	-	-	0.027	
0.00	86.18	42.40	0.02	96.33	80.70	5	0.027	0	0.000	-	-	0.027	
99.85	0.00	0.00	0.81	96.48	80.70	6	0.038	0	0.000	-	-	0.038	
99.85	0.00	33.79	0.81	96.48	80.70	6	0.038	0	0.000	-	-	0.038	
99.85	0.00	71.97	0.81	96.48	80.70	6	0.038	0	0.000	-	-	0.038	
0.00	99.65	0.00	2.62	96.54	80.70	7	0.053	0	0.000	-	-	0.053	
0.00	99.65	7.96	2.62	96.54	80.70	7	0.053	0	0.000	-	-	0.053	
0.00	99.65	45.74	2.62	96.54	80.70	7	0.053	0	0.000	-	-	0.053	
98.36	0.00	0.00	3.92	96.54	80.70	8	0.068	0	0.000	-	-	0.068	
98.36	0.00	51.81	3.92	96.54	80.70	8	0.068	0	0.000	-	-	0.068	
98.36	0.00	91.66	3.92	96.54	80.70	8	0.068	0	0.000	-	-	0.068	
0.00	99.53	0.00	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
0.00	99.53	0.00	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
0.00	99.53	0.00	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
0.00	99.53	0.00	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
0.00	99.53	0.00	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
0.00	99.53	0.00	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
0.00	99.53	15.96	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
0.00	99.53	55.87	6.51	96.54	80.70	9	0.083	0	0.000	-	-	0.083	
43.52	0.00	0.00	12.00	96.54	80.70	10	0.099	0	0.000	-	-	0.099	
43.52	0.00	25.96	12.00	96.54	80.70	10	0.099	0	0.000	-	-	0.099	
43.52	0.00	60.60	12.00	96.54	80.70	10	0.099	0	0.000	-	-	0.099	
0.00	98.13	0.00	16.56	96.54	80.70	11	0.114	0	0.000	-	-	0.114	
0.00	98.13	27.97	16.56	96.54	80.70	11	0.114	0	0.000	-	-	0.114	
0.00	98.13	69.93	16.56	96.54	80.70	11	0.114	0	0.000	-	-	0.114	
23.38	0.00	2.00	20.35	96.54	80.70	12	0.127	0	0.000	-	-	0.127	
23.38	0.00	41.97	20.35	96.54	80.70	12	0.127	0	0.000	-	-	0.127	
0.00	23.38	0.00	22.62	96.54	80.70	13	0.141	0	0.000	-	-	0.141	
0.00	23.38	13.99	22.62	96.54	80.70	13	0.141	0	0.000	-	-	0.141	
0.00	23.38	53.97	22.62	96.54	80.70	13	0.141	0	0.000	-	-	0.141	
24.61	0.00	0.00	24.89	96.54	80.70	14	0.155	0	0.000	-	-	0.155	
24.61	0.00	23.99	24.89	96.54	80.70	14	0.155	0	0.000	-	-	0.155	
24.61	12.76	100.00	26.02	96.54	80.70	15	0.155	0	0.000	-	-	0.155	
0.00	24.87	35.99	27.16	96.54	80.70	15	0.169	0	0.000	-	-	0.169	
0.00	24.87	77.98	27.16	96.54	80.70	15	0.169	0	0.000	-	-	0.169	
25.82	0.00	4.00	29.46	96.54	80.70	16	0.185	0	0.000	-	-	0.185	
25.82	0.00	45.99	29.46	96.54	80.70	16	0.185	0	0.000	-	-	0.185	
25.82	0.00	85.98	29.46	96.54	80.70	17	0.185	0	0.000	-	-	0.185	

读取文件操作没发现异常，全部是 minor gc，full gc 次数为 0。

3.3.3 使用 jmap -heap 命令行工具

```
Heap Configuration:
  MinHeapFreeRatio      = 40
  MaxHeapFreeRatio      = 70
  MaxHeapSize           = 6410993664 (6114.0MB)
  NewSize                = 1363144 (1.2999954223632812MB)
  MaxNewSize            = 3846176768 (3668.0MB)
  OldSize                = 5452592 (5.1999969482421875MB)
  NewRatio               = 2
  SurvivorRatio          = 8
  MetaspaceSize          = 21807104 (20.796875MB)
  CompressedClassSpaceSize = 1073741824 (1024.0MB)
  MaxMetaspaceSize       = 17592186044415 MB
  G1HeapRegionSize       = 1048576 (1.0MB)

Heap Usage:
G1 Heap:
  regions = 6114
  capacity = 6410993664 (6114.0MB)
  used = 1397227520 (1332.5MB)
  free = 5013766144 (4781.5MB)
  21.794242721622506% used
G1 Young Generation:
Eden Space:
  regions = 1112
  capacity = 1466957824 (1399.0MB)
  used = 1166016512 (1112.0MB)
  free = 300941312 (287.0MB)
  79.48534667619728% used
Survivor Space:
  regions = 86
  capacity = 90177536 (86.0MB)
  used = 90177536 (86.0MB)
  free = 0 (0.0MB)
  100.0% used
G1 Old Generation:
  regions = 135
  capacity = 915406848 (873.0MB)
  used = 139984896 (133.5MB)
  free = 775421952 (739.5MB)
  15.292096219931272% used
```

可以查看 jvm 此时的一些参数，这里的最大 heap size 是我改动过的 1G。

3.3.4 使用 jmap -clstats 命令行工具

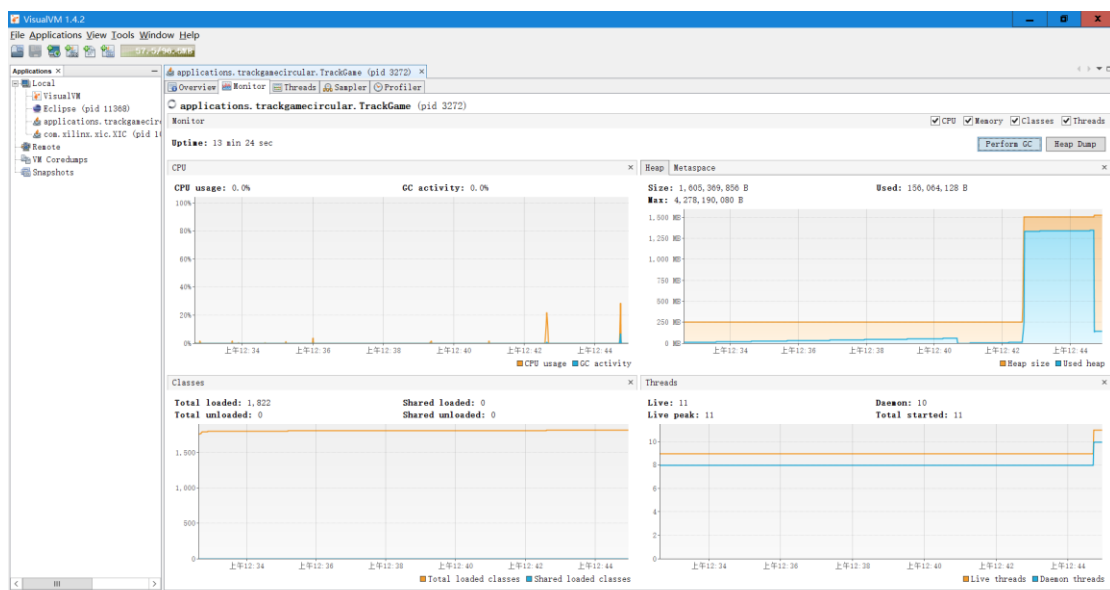
Index	Super	InstBytes	ClassBytes	annotations	CpAll	MethodCount	Bytecodes	MethodAll	ROA11	RWA11	Total	ClassName
1	28	32039008	584	0	1384	7	149	1864	1152	3000	4152	java.util.HashMap\$Node
2	-1	24435952	504	0	0	0	0	0	24	616	640	LB
3	28	24119520	616	128	14216	109	4577	49552	18640	47176	65816	java.lang.String
4	1227	20000000	624	0	1872	10	265	4088	1552	5360	6912	applications.trackgamecircular.Athlete
5	-1	8402032	504	0	0	0	0	0	32	616	648	[Ljava.util.HashMap\$Node;
6	689	8002144	592	0	5848	60	3943	20640	11744	16224	27968	java.lang.Integer
7	28	164136	672	0	22112	139	5682	38784	24616	38848	63464	java.lang.Class
8	-1	160296	504	0	0	0	0	0	24	616	640	C
9	-1	91920	504	0	0	0	0	0	24	616	640	[Ljava.lang.Object;
10	-1	45384	504	0	0	0	0	0	24	616	640	[I
11	28	41216	592	0	1360	9	213	2360	1488	3160	4648	java.util.concurrent.ConcurrentHashMap\$Node
12	829	27016	1192	0	6816	47	1117	9512	6480	11784	18264	java.lang.reflect.Method
13	-1	19536	504	0	0	0	0	0	32	616	648	[Ljava.util.concurrent.ConcurrentHashMap\$Node;
14	28	17328	576	0	11520	89	4308	35760	15088	34128	49216	java.lang.invoke.MemberName
15	28	16416	528	120	5496	37	1783	13400	6552	13552	20104	java.lang.invoke.LambdaForm\$Name
16	-1	15272	504	0	0	0	0	0	56	616	672	[Ljava.lang.Class;
17	-1	12736	504	0	0	0	0	0	48	616	664	[Ljava.lang.String;
18	-1	11696	504	0	0	0	0	0	24	616	640	[Ljava.lang.ref.SoftReference;
19	23	11616	560	0	848	4	83	2144	648	3112	3760	java.lang.invoke.MethodType\$ConcurrentWeakInternSet\$WeakEntry
20	28	10912	688	0	5448	35	5202	28592	12104	23200	35304	jdk.internal.math.FDBigInteger
21	28	9920	528	296	10224	72	2948	24456	12312	24248	36560	java.lang.invoke.MethodType
22	824	9200	560	0	688	3	56	1472	472	2432	2904	java.lang.ref.SoftReference
23	824	8544	560	0	448	2	13	1144	272	2016	2288	java.lang.ref.WeakReference
24	28	6048	528	0	280	1	5	168	152	984	1136	java.lang.invoke.ResolvedMethodName
25	28	5856	648	120	5120	28	840	6920	4248	9040	13288	java.lang.invoke.LambdaForm\$NamedFunction
26	28	5488	536	408	4952	29	1609	12320	6344	12880	19224	java.lang.invoke.MethodTypeForm
27	-1	5400	504	0	0	0	0	0	24	616	640	[Ljava.lang.invoke.LambdaForm\$Name;
28	-1	5072	520	0	1272	14	109	3424	1528	3920	5448	java.lang.Object
29	28	4960	608	0	1512	8	240	2112	1328	3232	4560	java.util.Hashtable\$Entry
30	920	4864	1464	0	17368	95	9759	48824	27784	42016	69800	java.util.concurrent.ConcurrentHashMap
31	28	4704	552	0	2168	11	337	2392	1816	3632	5448	java.lang.module.ModuleDescriptor\$Exports
32	-1	4320	504	0	0	0	0	0	24	616	640	[Ljava.lang.invoke.MethodHandle;
33	28	4320	792	344	16400	88	6186	28344	19232	28000	47232	java.lang.invoke.LambdaForm
34	689	4096	592	0	2464	26	326	4944	3272	5280	8552	java.lang.Byte
35	46	4080	912	0	936	4	54	800	616	2240	2856	java.lang.invoke.DirectMethodHandle\$Accessor
36	609	3528	760	0	3352	12	687	3072	2416	5040	7456	java.beans.MethodDescriptor
37	920	3504	1024	0	7804	51	4065	24888	12672	22128	34800	java.util.HashMap
38	28	3384	728	344	9016	73	1696	14976	10336	15952	26288	java.lang.Thread
39	829	2960	1192	0	6328	44	853	8680	5856	11056	16912	java.lang.reflect.Constructor
40	28	2904	528	0	1416	5	229	1128	888	2392	3280	java.beans.MethodRef
41	28	2752	528	0	728	2	21	352	456	1488	1944	java.lang.Class\$ReflectionData
42	22	2736	576	0	528	2	17	392	336	1344	1680	sun.util.locale.LocaleObjectCache\$CacheEntry
43	-1	2616	504	0	0	0	0	0	48	616	664	[Ljava.lang.reflect.Method;
44	-1	2240	504	0	0	0	0	0	32	616	648	[Ljava.util.Hashtable\$Entry;
45	919	2184	1440	0	7000	64	2681	19064	11232	17264	28496	java.util.ArrayList
46	796	2176	880	328	14616	42	4382	14224	12296	18808	31104	java.lang.invoke.DirectMethodHandle
47	665	2112	576	0	7088	5	1594	2512	3816	7704	11520	java.lang.invoke.LambdaForm\$Kind
48	609	2088	896	0	8128	31	2512	9024	7352	11224	16776	java.beans.PropertyDescriptor
49	22	1848	568	0	3048	23	975	5360	4096	5544	9640	java.lang.invoke.LambdaForm\$Editor\$Transform

下面还有很多不截取了，可以注意到第四行显示了我申请了多少个运动员对象。

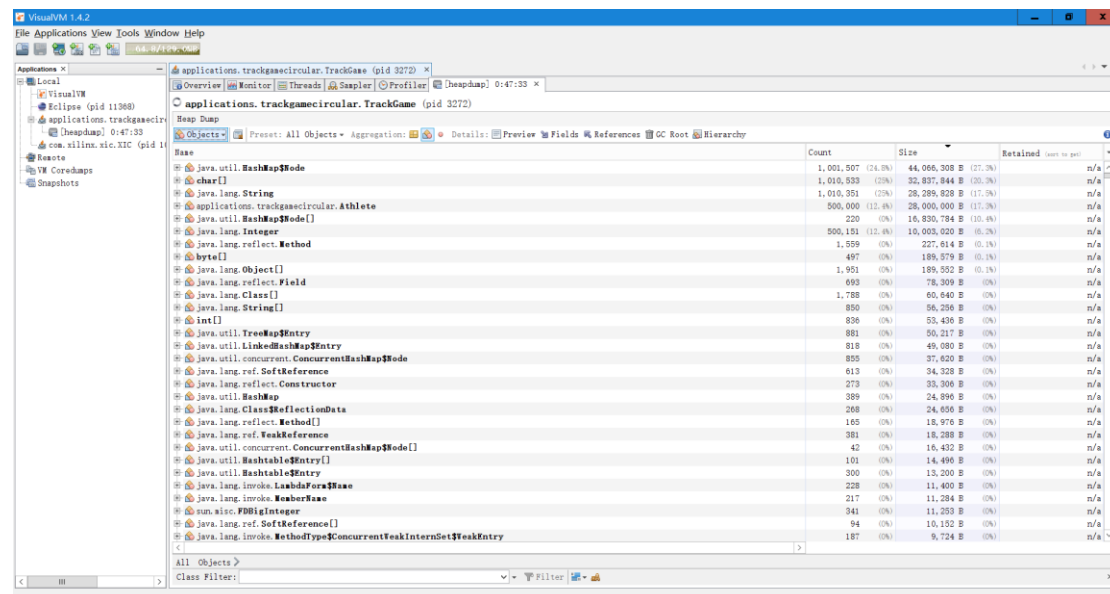
3.3.5 使用 jmap -permstat 命令行工具

-permstat 就是 java8 之前的-clstats 参数，目录有误。

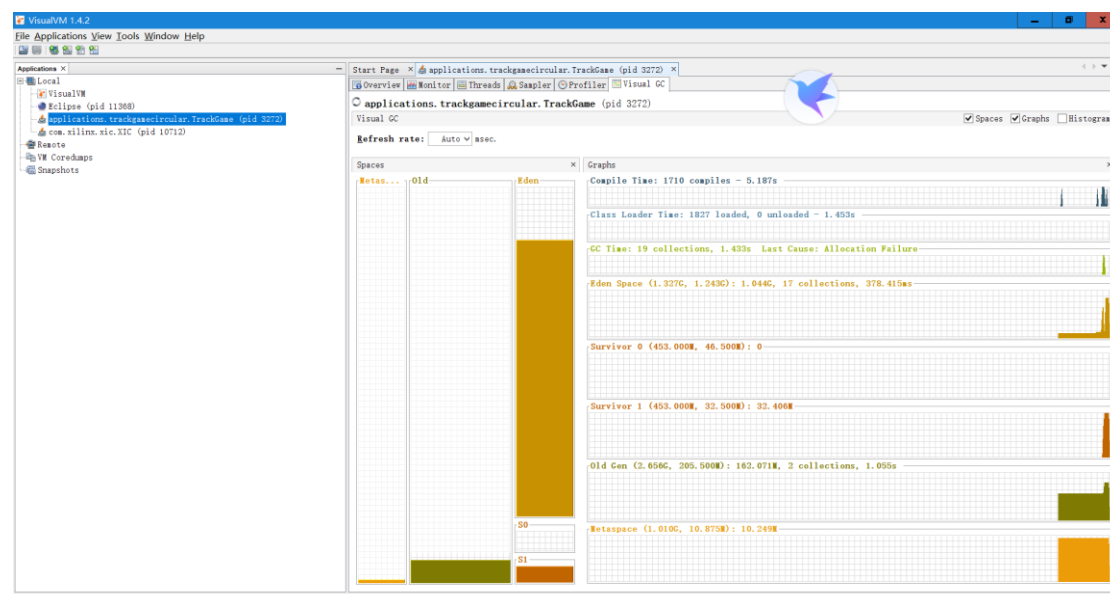
3.3.6 使用 JMC/JFR、jconsole 或 VisualVM 工具



heap 的分配和使用：



使用 visualGC 插件查看



3.3.7 分析垃圾回收过程

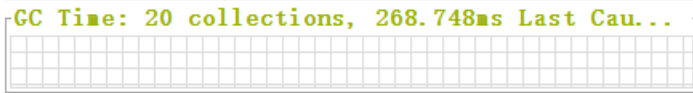
垃圾回收过程：

1. 当新的对象申请之后，会被储存在新生代的 Eden 区中，从它的名字也很好理解，Eden，就是诞生的地方的意思。
2. 当 eden 区放满了之后，进行一次 minor GC，对 eden 区的对象进行筛选，第一次是将存活的对象复制到 S0 区，之后将 eden 区域清空，下一次的 minor GC，S0 和 S1 的角色互换。
3. 存活过一定次数的 minor GC 之后的对象就会被放置到老年代之中。
4. 在程序进行了很多次 minor GC 之后，就有可能老年代的空间已经不够用了，此时会发生一次 full GC，相比于 minor GC，full GC 的时间开销比较大，因为 full GC 针对老年代，年轻代和永久代。

3.3.8 配置 JVM 参数并发现优化的参数配置

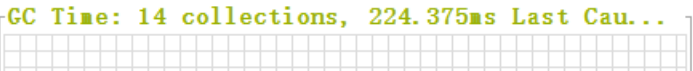
最开始设置的是-Xmx32m -Xms4m 马上发现根本不够用的，最大 heapsize 扩大到 1g 时的效果如下：（只是读取文件）：

```
GC Time: 20 collections, 268.748ms Last Cau...
```



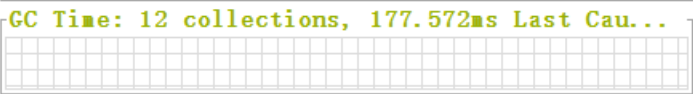
2g 时：

```
GC Time: 14 collections, 224.375ms Last Cau...
```



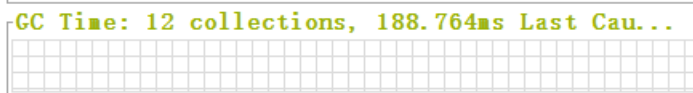
根据控制变量的思想，发现扩大最大堆大小似乎有利于加快 gc 速度，于是改为 4g 最大堆大小

```
GC Time: 12 collections, 177.572ms Last Cau...
```



再改为 8g：

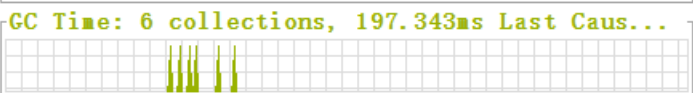
```
GC Time: 12 collections, 188.764ms Last Cau...
```



可以发现，此时过大的空间对 gc 速度的损害已经暴露出来了，特别是巨大的 eden 空间，这样我们就确定最大的堆大小。

接下来我们设置最小堆大小为 2g

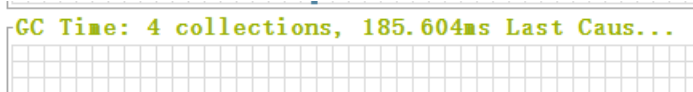
```
GC Time: 6 collections, 197.343ms Last Caus...
```



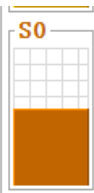
可以发现最小堆大小扩大可以有效得减少 minorGC 得次数

观察发现，eden space 的大小会一直扩大到 1.2g 左右，于是我们通过指定新生代的大小：使用-Xmn 1536m 来指定大小：

```
GC Time: 4 collections, 185.604ms Last Caus...
```



虽然时间差异不大但是也减小了回收次数。



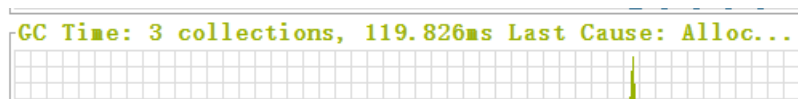
发现 S0 和 S1 在过程中始终处于放不满的状态，于是-XX:SurvivorRatio=8 调节一下

我们还发现 metaspace 的空间一直没有什么使用，于是使用：-XX:MetaspaceSize=30m 和-XX:MaxMetaspaceSize=30m

所以最后使用的参数如下

```
-verbose:gc
-Xmx4g
-Xmn1536m
-XX:SurvivorRatio=8
-XX:MetaspaceSize=30m
-XX:MaxMetaspaceSize=30m
-Dfile.encoding=UTF-8
```

相比 default 设置也是快了不少：



3.4 Dynamic Program Profiling

3.4.1 使用 JMC 或 VisualVM 进行 CPU Profiling

main	4,491 ms (100%)	4,223 ms (100%)	2
applications.trackgamecircular.TrackCircularOrbitBuilder	4,491 ms (100%)	4,223 ms (100%)	1
Self time	2,753 ms (61.3%)	2,517 ms (59.6%)	1
applications.trackgamecircular.Athlete.getInstance (String, Integer, String, Integer, double)	500,000		
applications.trackgamecircular.Athlete.checkRep	1,015 ms (22.6%)	938 ms (22.2%)	500,000
Self time	450 ms (10%)	298 ms (7.1%)	500,000
iostrategy.BufferedInputStrategy.readLine ()	270 ms (6%)	469 ms (11.1%)	500,042
applications.trackgamecircular.TrackCircularOrbitBuilder	1.3 ms (0%)	0.0 ms (0%)	39
applications.trackgamecircular.Athlete.<clinit>	0.063 ms (0%)	0.0 ms (0%)	1
iostrategy.BufferedInputStrategy.close ()	0.038 ms (0%)	0.0 ms (0%)	1
applications.trackgamecircular.TrackGame.gameMenu	0.337 ms (0%)	0.0 ms (0%)	1

先看文件读取环节，主要是构建函数的时间开销大，主要开销为：正则表达式匹配，和文件读取的 readline 函数，构造 person 实例时也可以发现，主要是 checkrep 占据了时间，是因为我在 checkrep 中检查了各个运动员参数是否合法，同样也是基本正则匹配的开销。

再看安排比赛的开销：

applications.trackgamecircular.TrackGame.arrangeOrbit	33,090 ms (88%)	32,700 ms (88.4%)	1
applications.trackgamecircular.strategy.RandomStrategy	21,702 ms (57.7%)	21,390 ms (57.9%)	1
Self time	10,607 ms (28.2%)	10,637 ms (28.8%)	1
circularorbit.CircularOrbitBuilder.bulidPhysical	403 ms (1.1%)	276 ms (0.7%)	50,001
circularorbit.CircularOrbitBuilder.bulidTracks	313 ms (0.8%)	370 ms (1%)	50,001
applications.trackgamecircular.TrackCircularOrbitBuilder	63.7 ms (0.2%)	25.0 ms (0.1%)	50,001
track.Track.<clinit> ()	0.032 ms (0%)	0.0 ms (0%)	1
track.Track.checkRep ()	0.031 ms (0%)	0.0 ms (0%)	10

和我预想的一样，时间开销还是主要花在了 strategy 的执行上，所以，开销都是合理的，只是有点慢。

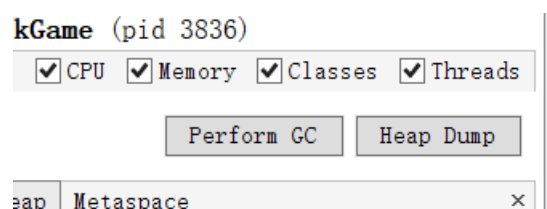
3.4.2 使用 VisualVM 进行 Memory profiling

name	count	size	count	size
int[]	228,414,880 B (27.5%)	1,479,270 (8.6%)		
char[]	70,698,448 B (8.5%)	1,617,494 (9.4%)		
java.util.HashMap\$Node	64,866,240 B (7.8%)	2,027,070 (11.7%)		
java.lang.Object[]	61,134,840 B (7.4%)	1,481,984 (8.6%)		
boolean[]	48,452,592 B (5.8%)	178,150 (1%)		
java.lang.String	32,849,856 B (4%)	1,368,744 (7.9%)		
byte[]	27,372,056 B (3.3%)	126,917 (0.7%)		
applications.trackgamecircular.Athlete	20,000,000 B (2.4%)	500,000 (2.9%)		
java.util.regex.Pattern\$Curly	18,241,152 B (2.2%)	570,036 (3.3%)		
java.util.HashMap\$Node[]	17,547,632 B (2.1%)	120,068 (0.7%)		
java.util.LinkedList	16,000,512 B (1.9%)	500,016 (2.9%)		
java.util.regex.Pattern	15,391,008 B (1.9%)	213,764 (1.2%)		
java.io.ObjectStreamClass\$WeakClassKey	15,324,064 B (1.8%)	478,877 (2.8%)		
java.util.LinkedList\$Node	14,400,288 B (1.7%)	600,012 (3.5%)		
java.util.TreeMap\$Entry	13,928,480 B (1.7%)	348,212 (2%)		
java.util.regex.Matcher	13,680,768 B (1.6%)	213,762 (1.2%)		
java.util.ArrayList	12,125,616 B (1.5%)	505,234 (2.9%)		
java.util.regex.Pattern\$GroupHead[]	11,970,560 B (1.4%)	213,760 (1.2%)		
java.util.HashMap	11,874,528 B (1.4%)	247,386 (1.4%)		
java.util.regex.Pattern\$Ctype	10,260,792 B (1.2%)	427,533 (2.5%)		
java.util.regex.Pattern\$Slice	8,550,720 B (1%)	356,280 (2.1%)		
java.lang.Integer	8,490,272 B (1%)	530,642 (3.1%)		
java.util.regex.Pattern\$GroupTail	7,695,504 B (0.9%)	320,646 (1.9%)		
java.util.regex.Pattern\$GroupHead	7,695,504 B (0.9%)	320,646 (1.9%)		
java.util.regex.Pattern\$Single	6,840,528 B (0.8%)	285,022 (1.6%)		
java.util.regex.Pattern\$1	5,985,288 B (0.7%)	249,387 (1.4%)		

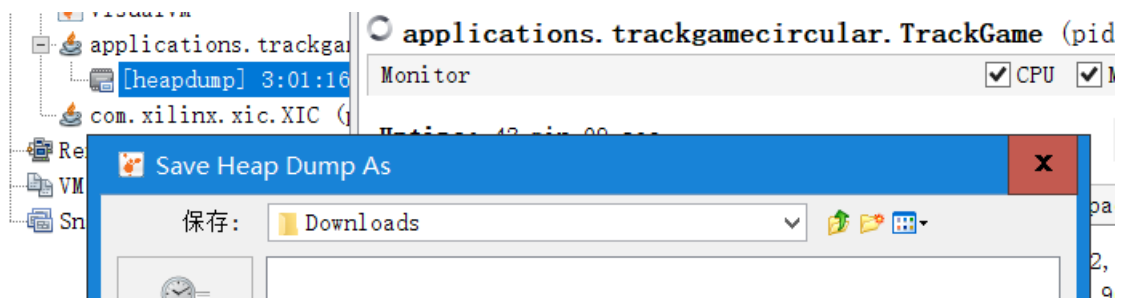
总体来说正常，除了 int 和 char 数组，最高的就是 hashmap 的 node 了，这是为了快速查找时申请的 hashmap，之后是 pattern，因为多次匹配字符串的原因，这一类对象也特别多，运动员的人数正好是 500000。总体合理。

3.5 Memory Dump Analysis and Performance Optimization

3.5.1 内存导出

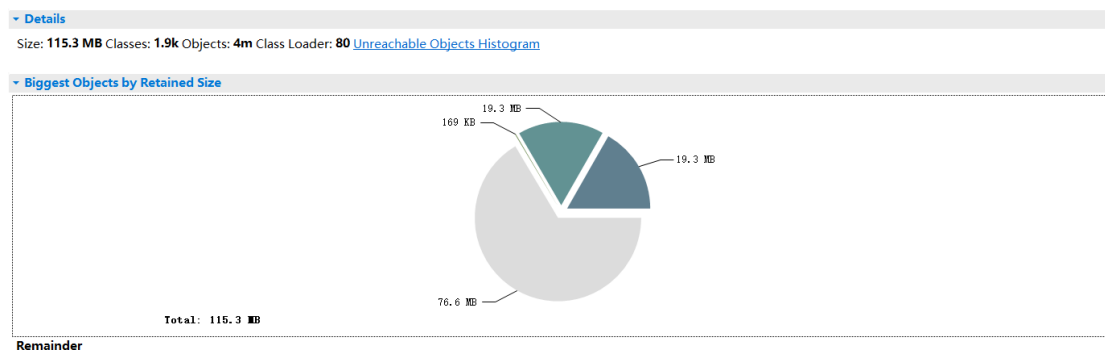


点击 heap dump 之后
右键就可以另存为了。



3.5.2 使用 MAT 分析内存导出文件

Overview:



Histogram:

Class Name	Objects	Shallow Heap	Retained
<Regex>	<Numeric>	<Numeric>	<Numeri>
java.util.HashMap\$Node	1,001,500	32,048,000	
char[]	1,008,125	27,366,576	
java.lang.String	1,007,943	24,190,632	
applications.trackgamecircular.Athlete	500,000	16,000,000	
java.lang.Double	500,001	12,000,024	
java.util.HashMap\$Node[]	206	8,415,688	
java.lang.Integer	500,151	8,002,416	
byte[]	454	185,560	
java.lang.reflect.Method	621	54,648	
int[]	510	42,064	
java.lang.Object[]	930	35,896	
java.util.TreeMap\$Entry	881	35,240	
java.lang.String[]	850	32,576	
java.util.LinkedHashMap\$Entry	810	32,400	
java.util.concurrent.ConcurrentHashMap\$Node	855	27,360	
java.lang.Class	1,924	18,568	
java.util.HashMap	336	16,128	
java.lang.ref.SoftReference	346	13,840	
java.lang.Class[]	556	13,184	
java.lang.ref.WeakReference	382	12,224	
sun.misc.FDBigInteger	341	10,912	
java.util.Hashtable\$Entry	301	9,632	
java.util.concurrent.ConcurrentHashMap\$Node[]	42	8,384	
java.lang.Object	484	7,744	
java.lang.Long	322	7,728	
java.lang.invoke.LambdaForm\$Name	228	7,296	
java.lang.reflect.Field	94	6,768	
java.lang.invoke.MemberName	209	6,688	
java.util.Hashtable\$Entry[]	78	6,568	
	155	6,320	



java.lang.ref.SoftReference[]	94	5,640
java.lang.invoke.MethodType	137	5,480
java.net.URL	79	5,056
java.util.WeakHashMap\$Entry	124	4,960
javax.management.ImmutableDescriptor	189	4,536
java.io.ObjectStreamClass	42	4,368
java.lang.Byte	256	4,096
com.sun.jmx.mbeanserver.ConvertingMethod	128	4,096
java.lang.Short	256	4,096
Total: 40 of 1,915 entries; 1,875 more	4,536,740	128,871,328

Top consumer:

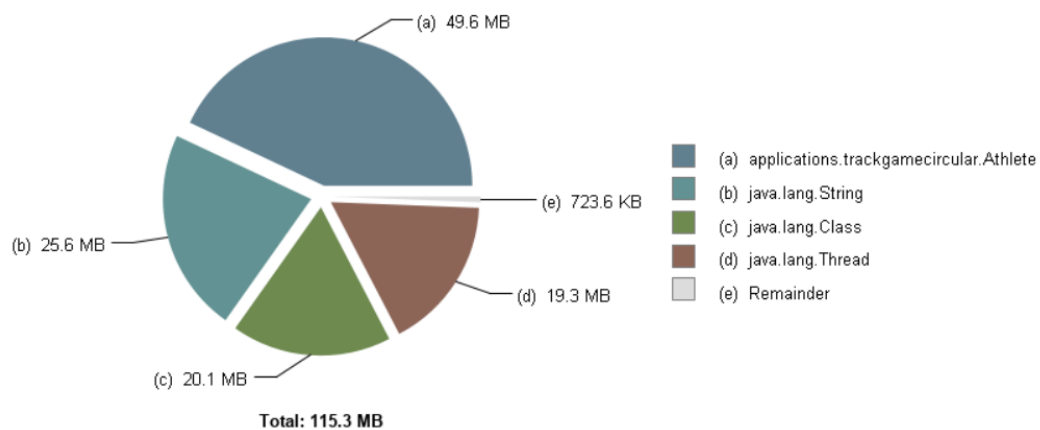
Top Consumers

▼ **Biggest Objects** 🗑️

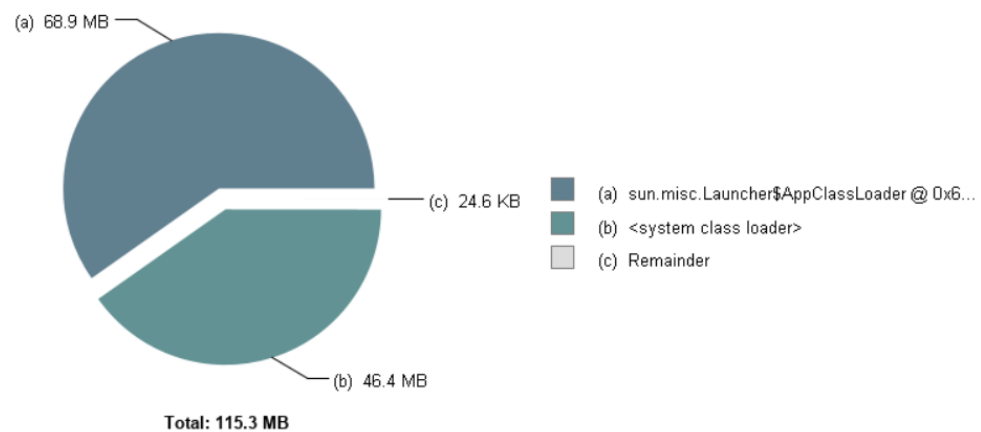
hide / unhide

Class Name	Shallow Heap	Retained Heap
 java.lang.Thread @ 0x6fea36348 main Thread »	120	20,220,672
 class applications.trackgamecircular.Athlete @ 0x6fee65190 »	8	20,194,376
Σ Total: 2 entries		

▼ **Biggest Top-Level Dominator Classes (Overview)**



▼ Biggest Top-Level Dominator Class Loaders (Overview)



▼ Biggest Top-Level Dominator Packages

Package	Retained Heap	Retained Heap, %	# Top Dominators
<all> First 10 of 1,004,175 objects	120,901,496	100.00%	1,004,175
applications First 10 of 500,003 objects	72,192,336	59.71%	500,003
trackgamecircular First 10 of 500,003 objects	72,192,336	59.71%	500,003
Athlete First 10 of 500,001 objects	72,192,328	59.71%	500,001
java First 10 of 502,971 objects	47,783,688	39.52%	502,971
lang First 10 of 502,187 objects	47,254,680	39.09%	502,187
String First 10 of 501,129 objects	26,813,808	22.18%	501,129
Thread All 9 objects	20,259,520	16.76%	9
Σ Total: 2 entries	47,073,328		501,138
Σ Total: 2 entries	119,976,024		1,002,974

Dominator tree:

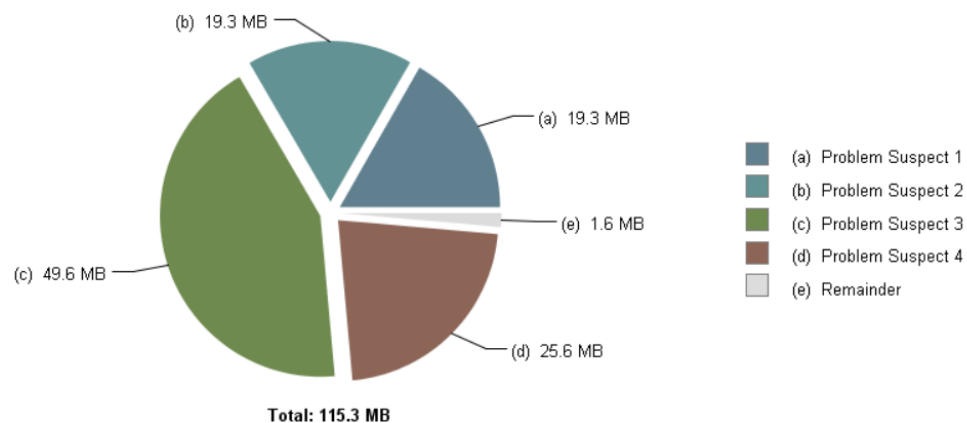
i Overview dominator_tree			
Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
> java.lang.Thread @ 0x6fea36348 main Thread	120	20,220,672	16.72%
> class applications.trackgamecircular.Athlete @ 0x6fee65190	8	20,194,376	16.70%
> class java.util.ResourceBundle @ 0x7f552ad08 System Class	24	173,048	0.14%
> class sun.util.calendar.ZoneInfoFile @ 0x7f552cd78 System Cla	120	159,360	0.13%
> sun.nio.cs.ext.ExtendedCharsets @ 0x7f556ae68	40	79,704	0.07%
> class java.io.ObjectStreamClass\$Caches @ 0x7f50a68f0 System	16	70,288	0.06%
> class java.beans.ThreadGroupContext @ 0x7f552fae8 System C	8	67,624	0.06%
> char[28672] @ 0x7f50d3cf0 \ufffd\ufffd\ufffd\ufffd\ufffd\	57,360	57,360	0.05%
> class sun.nio.cs.ext.GBK @ 0x7f5540a78 System Class	32	55,048	0.05%
> char[256][] @ 0x6fee7fb98	1,040	51,440	0.04%
> sun.security.provider.Sun @ 0x7f58e1670	96	40,768	0.03%
> class sun.misc.FDBigInteger @ 0x7f5508fb8 System Class	32	37,424	0.03%
> sun.misc.Launcher\$AppClassLoader @ 0x6fea0e880	88	33,408	0.03%
> sun.management.DiagnosticCommandImpl @ 0x7f508cfb8	40	28,760	0.02%
> java.io.PrintStream @ 0x6fea19530	32	25,048	0.02%
> java.io.PrintStream @ 0x6fea1b880	32	25,048	0.02%
> sun.misc.Launcher\$ExtClassLoader @ 0x6fea0e8e0	80	24,656	0.02%
> class java.nio.charset.Charset @ 0x6fea05320 System Class	24	20,424	0.02%
> java.lang.Thread @ 0x6fea2ba58 RMI TCP Connection(1)-192.1	120	18,408	0.02%
> java.lang.Thread @ 0x6fea1ba90 RMI TCP Connection(2)-192.1	120	18,192	0.02%
> class sun.util.locale.provider.LocaleProviderAdapter @ 0x7f552	40	16,136	0.01%
> com.sun.jmx.mbeanserver.PerInterface @ 0x7f5536fd8	40	15,496	0.01%
> class java.lang.Package @ 0x7f553a8f0 System Class	16	14,336	0.01%
> com.sun.jmx.mbeanserver.DefaultMXBeanMappingFactory\$Co	48	14,024	0.01%
> org.apache.log4j.DailyRollingFileAppender @ 0x6fee5d820	96	13,440	0.01%
> org.apache.log4j.DailyRollingFileAppender @ 0x6fee69700	96	13,440	0.01%
> class java.security.Security @ 0x7f55310c8 System Class	16	13,408	0.01%
> class java.lang.invoke.MethodType @ 0x7f50c3f00 System Clas	56	13,264	0.01%
> com.sun.jmx.mbeanserver.JmxMBeanServer @ 0x7f508be50	40	12,616	0.01%
> java.util.logging.LogManager @ 0x6fea04a70	56	12,056	0.01%
> java.util.HashSet @ 0x7f5729b30	16	11,536	0.01%
> java.util.logging.LogManager @ 0x6fea04a70	56	12,056	0.01%
> java.util.HashSet @ 0x7f5729b30	16	11,536	0.01%
> org.apache.log4j.ConsoleAppender @ 0x6fee6cc20	64	11,448	0.01%
> class java.io.File @ 0x7f553dc40 System Class	48	10,792	0.01%
> java.util.jar.JarFile @ 0x6fee72740	64	9,704	0.01%
> class sun.rmi.runtime.Log\$LoggerLog @ 0x7f50c8d10 System	8	8,528	0.01%
Total: 35 of 1,004,175 entries; 1,004,140 more			

Leak Suspects:

System Overview

Leaks

Overview



Problem Suspect 1

The thread **java.lang.Thread @ 0x6fea36348 main** keeps local variables with total size **20,220,672 (16.72%)** bytes.

The memory is accumulated in one instance of **"java.util.HashMap\$Node[]"** loaded by **"<system class loader>"**.

The stacktrace of this Thread is available. [See stacktrace](#).

Keywords

java.util.HashMap\$Node[]

[Details »](#)

Problem Suspect 2

The class **"applications.trackgamecircular.Athlete"**, loaded by **"sun.misc.Launcher\$AppClassLoader @ 0x6fea0e880"**, occupies **20,194,376 (16.70%)** bytes. The memory is accumulated in one instance of **"java.util.HashMap\$Node[]"** loaded by **"<system class loader>"**.

Keywords

sun.misc.Launcher\$AppClassLoader @ 0x6fea0e880

applications.trackgamecircular.Athlete

java.util.HashMap\$Node[]

[Details »](#)

▼ Problem Suspect 3

500,000 instances of "**applications.trackgamecircular.Athlete**", loaded by "**sun.misc.Launcher\$AppClassLoader @ 0x6fea0e880**" occupy **51,997,952 (43.01%)** bytes. These instances are referenced from one instance of "**java.util.HashMap\$Node[]**", loaded by "<system class loader>"

Keywords

sun.misc.Launcher\$AppClassLoader @ 0x6fea0e880
applications.trackgamecircular.Athlete
java.util.HashMap\$Node[]

[Details »](#)

▼ Problem Suspect 4

501,128 instances of "**java.lang.String**", loaded by "<system class loader>" occupy **26,813,760 (22.18%)** bytes. These instances are referenced from one instance of "**java.util.HashMap\$Node[]**", loaded by "<system class loader>"

Keywords

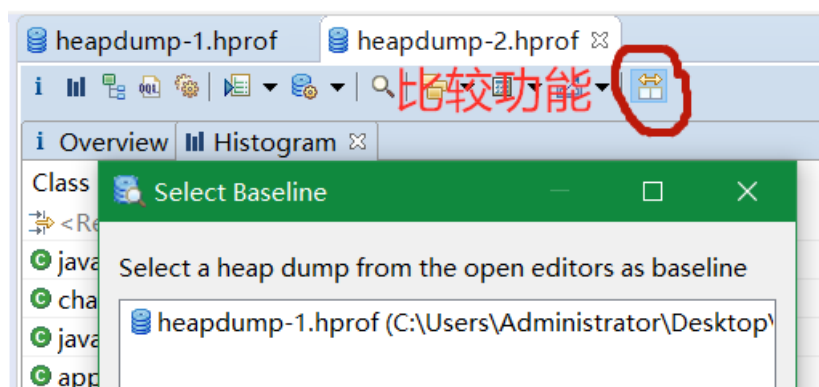
java.util.HashMap\$Node[]
java.lang.String

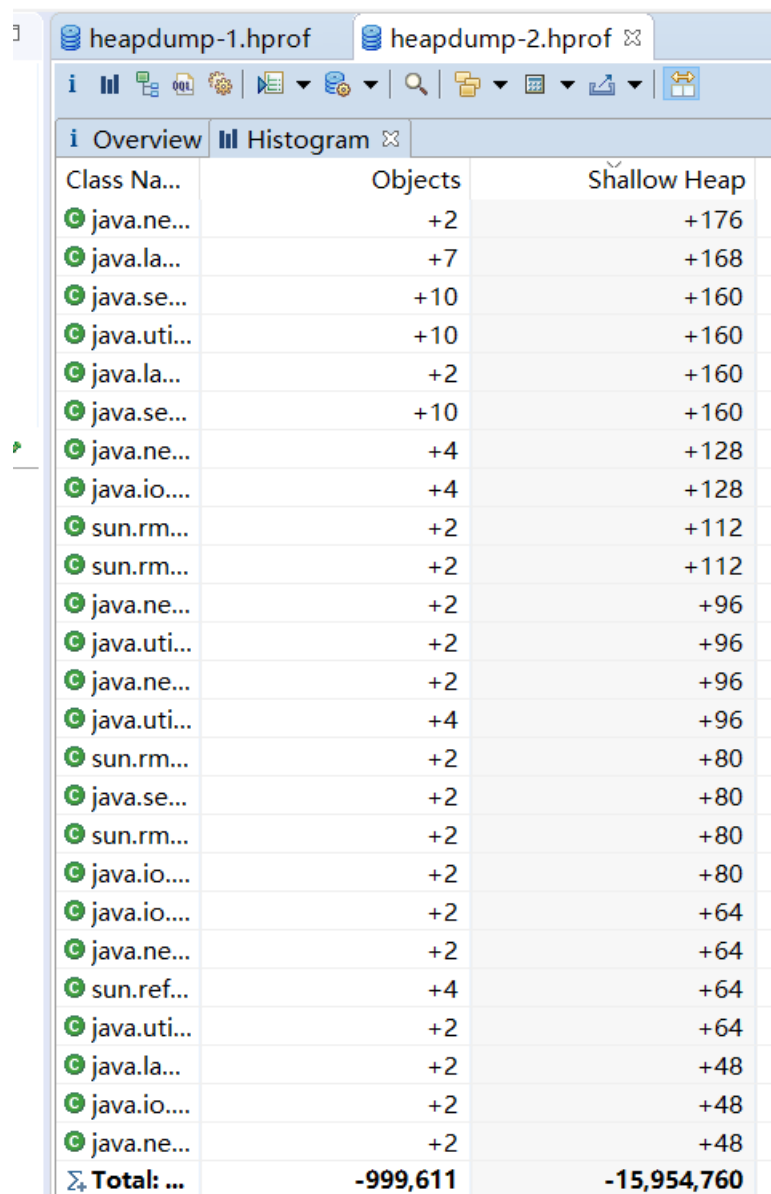
[Details »](#)

3.5.3 发现热点/瓶颈并改进、改进前后的性能对比分析

applications.trackgamecircular.Athlete	500,000	10,000,000
java.lang.Double	500,001	12,000,024
java.util.HashMap\$Node[]	206	8,415,688
java.lang.Integer	500,151	8,002,416

通过观察 Histogram，我发现我申请了 500000 个 Double 和 Integer 对象，但是这个其实是可以通过使用 double 和 int 直接避免的，于是修改了构造函数和传递的参数。
重新生成 heapdump 之后使用 MAT 的比较功能





Class Na...	Objects	Shallow Heap
java.ne...	+2	+176
java.la...	+7	+168
java.se...	+10	+160
java.uti...	+10	+160
java.la...	+2	+160
java.se...	+10	+160
java.ne...	+4	+128
java.io....	+4	+128
sun.rm...	+2	+112
sun.rm...	+2	+112
java.ne...	+2	+96
java.uti...	+2	+96
java.ne...	+2	+96
java.uti...	+4	+96
sun.rm...	+2	+80
java.se...	+2	+80
sun.rm...	+2	+80
java.io....	+2	+80
java.io....	+2	+64
java.ne...	+2	+64
sun.ref...	+4	+64
java.uti...	+2	+64
java.la...	+2	+48
java.io....	+2	+48
java.ne...	+2	+48
Total: ...	-999,611	-15,954,760

可以发现减少了很多对象。

3.5.4 在 MAT 内使用 OQL 查询内存导出

查询 TrackCircularOrbit:

i Overview Histogram OQL			
select * from instanceof circularorbit.ConcreteCircularOrbit			
Class Name	Shallow Heap	Retained Heap	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23dc1	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23db8	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23d47	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23d40	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23d3a	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23cd9	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23cd4	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23ccc	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23cc7	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23cbc	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23cb7	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23c5c	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23c56	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23c45	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23c3f	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23c37	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23bfc	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23bf6	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23bea	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23bcc	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23bc3	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23bbc	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23bb8	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23b84	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23b6e	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23b68	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23b63	32	1,392	
> applications.trackgamecircular.TrackCircularOrbit @ 0x7f23b5c	32	1,392	
Total: 28 of 50,001 entries; 49,973 more			




































查询长度大于 50 的字符串对象：

i Overview Histogram OQL			
select objects s.value from java.lang.String s where s.value.@length>=50			
Class Name	Shallow Heap	Retained Heap	
> char[161] @ 0x7ed8e1840 defaultValueljava/lang/Object;legal	344	344	
> char[82] @ 0x7ed8e1310 classNameLjava/lang/String;descripti	184	184	
> char[84] @ 0x7ed8e0ff8 compositeTypeLjavax/management/c	184	184	
> char[258] @ 0x7ed8e0d58 attributes[Ljavax/management/MB	536	536	
> char[50] @ 0x7ed8dcd38 javax.management.remote.rmi.RMIC	120	120	
> char[60] @ 0x7ed8dcb78 org.omg.stub.javax.management.rer	136	136	
> char[69] @ 0x7ed8dbc4e0 void clean(java.rmi.server.ObjID[], lor	160	160	
> char[75] @ 0x7ed8bc410 java.rmi.dgc.Lease dirty(java.rmi.serv	168	168	
> char[166] @ 0x7ed8bb350 (Ljava/lang/Class;Ljava/lang/String;l	352	352	
> char[51] @ 0x7ed8b97e0 (Ljava/lang/Object;ILjava/lang/invoke	120	120	
> char[51] @ 0x7ed8b8fb0 (ILjava/lang/Object;Ljava/lang/invoke	120	120	
> char[57] @ 0x7ed8b8a90 (ILjava/lang/Object;Ljava/lang/Object	136	136	
> char[86] @ 0x7ed8b8848 (ILjava/lang/Object;Ljava/lang/Objec	192	192	
> char[93] @ 0x7ed8b8300 (Ljava/lang/Object;Ljava/lang/Object	208	208	
> char[104] @ 0x7ed8b7fc8 (Ljava/lang/Object;ILjava/lang/Objec	224	224	
> char[75] @ 0x7ed8b7c48 (Ljava/lang/Object;ILjava/lang/Objec	168	168	
> char[55] @ 0x7ed8b7178 (Ljava/lang/Class;Ljava/lang/Object;l	128	128	
> char[50] @ 0x7ed890778 javax.security.auth.login.Configuratic	120	120	
> char[247] @ 0x7ed870ef8 SUN (DSA key/parameter generatio	512	512	
> char[52] @ 0x7ed8708f8 sun.security.provider.certpath.PKIXCe	120	120	
> char[50] @ 0x7ed86ff30 sun.security.provider.certpath.Collect	120	120	
> char[51] @ 0x7ed86ef78 Alg.Alias.AlgorithmParameters.OID.1.	120	120	
> char[50] @ 0x7ed86ea28 Alg.Alias.MessageDigest.OID.2.16.84	120	120	
> char[50] @ 0x7ed86e940 Alg.Alias.MessageDigest.OID.2.16.84	120	120	
> char[50] @ 0x7ed86e858 Alg.Alias.MessageDigest.OID.2.16.84	120	120	
> char[50] @ 0x7ed86e620 Alg.Alias.MessageDigest.OID.2.16.84	120	120	
> char[57] @ 0x7ed86dc38 sun.security.provider.certpath.Index	136	136	
> char[76] @ 0x7ed86dad8 java.security.interfaces.DSAPublicKey	168	168	
Total: 36 of 534 entries; 498 more			

大于特定大小 50 的任意对象：

i Overview Histogram OQL			
select * from instanceof java.lang.Object o where o.@usedHeapSize>=50			
Class Name	Shallow Heap	Retained Heap	
<Regex>	<Numeric>	<Numeric>	
> java.lang.Thread @ 0x7ecba8110 Signal Dispatcher Thread	120	256	
> java.lang.Thread @ 0x7ecb9b2e0 main Thread	120	91,022,344	
> java.lang.Thread @ 0x7ecb4a958 RMI TCP Accept-0 Thread	120	304	
> java.lang.Thread @ 0x7ecb2d010 Attach Listener Thread	120	960	
> java.lang.Thread @ 0x7eca753f8 JMX server connection timeo	120	296	
> java.lang.Thread @ 0x7eca75180 RMI TCP Connection(1)-192.1	120	18,408	
> java.lang.Thread @ 0x7eca72b70 RMI TCP Connection(2)-192.	120	18,192	
> java.lang.Thread @ 0x7ec9aab38 RMI Scheduler(0) Thread	120	248	
> java.lang.Object[10] @ 0x7f23dc6e8	56	56	
> java.lang.Object[10] @ 0x7f23dc678	56	56	
> java.lang.Object[10] @ 0x7f23dc608	56	56	
> java.lang.Object[10] @ 0x7f23dc4e8	56	56	
> java.lang.Object[10] @ 0x7f23dc478	56	56	
> java.lang.Object[10] @ 0x7f23dc408	56	56	
> java.lang.Object[10] @ 0x7f23dc398	56	56	
> java.lang.Object[10] @ 0x7f23dc328	56	56	
> java.lang.Object[10] @ 0x7f23dc2b8	56	56	
> java.lang.Object[10] @ 0x7f23dc248	56	56	
> java.lang.Object[10] @ 0x7f23dbd70	56	56	
> java.lang.Object[10] @ 0x7f23dbd00	56	56	
> java.lang.Object[10] @ 0x7f23dbc90	56	56	
> java.lang.Object[10] @ 0x7f23dbc20	56	56	
> java.lang.Object[10] @ 0x7f23dbbb0	56	56	
> java.lang.Object[10] @ 0x7f23dbb40	56	56	
> java.lang.Object[10] @ 0x7f23dbad0	56	56	
> java.lang.Object[10] @ 0x7f23dba60	56	56	
> java.lang.Object[10] @ 0x7f23db9f0	56	56	
> java.lang.Object[10] @ 0x7f23db980	56	56	
> java.lang.Object[10] @ 0x7f23d4c88	56	56	
> java.lang.Object[10] @ 0x7f23d4a20	56	56	
> java.lang.Object[10] @ 0x7f23d49b0	56	56	
> java.lang.Object[10] @ 0x7f23d4940	56	56	
> java.lang.Object[10] @ 0x7f23d48d0	56	56	
Total: 36 of 555,967 entries; 555,931 more			

PhysicalObject 的所有子类：

i Overview Histogram OQL			
select * from instanceof phsicalobject.PhysicalObject			
Class Name	Shallow Heap	Retained Heap	
 <Regex>	<Numeric>	<Numeric>	
>  applications.trackgamecircular.Athlete @ 0x7f23dc520	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23d4cc0	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23d4b38	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23d4500	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23d3f38	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23d3c60	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23cdb0	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23cd318	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23cd038	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23cceb0	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23c4810	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23c41d8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23c34f8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23c0020	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23bee50	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23bc968	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23bc848	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23bc6b8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23bb4c0	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23b5608	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23b3748	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23ae288	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23adf38	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23ad9e8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23ad858	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23ad3e8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23ac3e0	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23ac258	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23ab760	32	120	
...			
>  applications.trackgamecircular.Athlete @ 0x7f23a65c8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23a64a8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23a54c0	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23a53a8	32	120	
>  applications.trackgamecircular.Athlete @ 0x7f23a50c8	32	120	
Σ Total: 36 of 500,000 entries; 499,964 more			

总占用内存：

i Overview Histogram OQL Histogram of oql "select * from instanceof phsicalobject.PhysicalObject"			
Class Name	Objects	Shallow Heap	
<Regex>	<Numeric>	<Numeric>	
applications.trackgamecircular.Athlete	500,000	16,000,000	

查询下大于 100 个物体的所有 collection

因为没法一次性查所有 collection，所以就查了几个

HashMap:

i Overview Histogram OQL Histogram of oql "select * from instanceof java.util.HashMap m where m.size>=100"		
Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
> java.util.LinkedHashMap @ 0x7ed2dd978	56	116,072
> java.util.HashMap @ 0x7ed386568	48	30,888
> java.util.HashMap @ 0x7ed2aeae0	48	11,520
> java.util.HashMap @ 0x7ed2838e0	48	12,808
> java.util.HashMap @ 0x7ed27edb8	48	9,376
> java.util.HashMap @ 0x7ecbd8df0	48	20,194,576
> java.util.HashMap @ 0x6fea021d8	48	20,194,368
Σ Total: 7 entries		

ArrayList: 结果没有

i Overview Histogram OQL Histogram of oql "select * from instanceof java.util.ArrayList l where l.size>=100"		
Your Query did not yield any result.		
SELECT * FROM INSTANCEOF java.util.ArrayList l WHERE (l.size >= 100)		

LinkedList:

i Overview Histogram OQL Histogram of oql "select * from instanceof java.util.LinkedList l where l.size>=100"		
Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
> java.util.LinkedList @ 0x7ed1a6268	32	70,800,056

3.5.5 观察 jstack/jcmd 导出程序运行时的调用栈

读取文件时:

```
"main" #1 prio=5 os_prio=0 cpu=2843.75ms elapsed=25.74s tid=0x000001f990fb5000 nid=0x3bc4 runnable [0x000000376e8fe000]
java.lang.Thread.State: RUNNABLE
    at java.io.FileInputStream.readBytes(java.base@11.0.3/Native Method)
    at java.io.FileInputStream.read(java.base@11.0.3/FileInputStream.java:279)
    at java.io.BufferedReader.read1(java.base@11.0.3/BufferedReader.java:290)
    at java.io.BufferedReader.read(java.base@11.0.3/BufferedReader.java:351)
    - locked <0x00000000c0243bd8> (a java.io.BufferedReader)
    at sun.nio.cs.StreamDecoder.readBytes(java.base@11.0.3/StreamDecoder.java:284)
    at sun.nio.cs.StreamDecoder.implRead(java.base@11.0.3/StreamDecoder.java:326)
    at sun.nio.cs.StreamDecoder.read(java.base@11.0.3/StreamDecoder.java:178)
    - locked <0x00000000c033e760> (a java.io.InputStreamReader)
    at java.io.InputStreamReader.read(java.base@11.0.3/InputStreamReader.java:185)
    at java.io.BufferedReader.fill(java.base@11.0.3/BufferedReader.java:161)
    at java.io.BufferedReader.readLine(java.base@11.0.3/BufferedReader.java:326)
    - locked <0x00000000c033e760> (a java.io.InputStreamReader)
    at java.io.BufferedReader.readLine(java.base@11.0.3/BufferedReader.java:392)
    at applications.trackgamecircular.TrackGame.gameMain(TrackGame.java:90)
    at applications.trackgamecircular.TrackGame.main(TrackGame.java:430)
```

随机分配赛道:


```

"main" #1 prio=5 os_prio=0 cpu=35078.13ms elapsed=109.96s tid=0x00000187b3387800 nid=0x396c runnable [0x0000007a52afe000]
java.lang.Thread.State: RUNNABLE
    at java.util.Random.next(java.base@11.0.3/Random.java:204)
    at java.util.Random.nextInt(java.base@11.0.3/Random.java:390)
    at applications.trackgamecircular.strategy.RandomStrategy.arrange(RandomStrategy.java:39)
    at applications.trackgamecircular.TrackGame.arrangeOrbit(TrackGame.java:395)
    at applications.trackgamecircular.TrackGame.gameMain(TrackGame.java:137)
    at applications.trackgamecircular.TrackGame.main(TrackGame.java:430)

```

有序分配赛道：

```

"main" #1 prio=5 os_prio=0 cpu=66015.63ms elapsed=143.87s tid=0x00000187b3387800 nid=0x396c runnable [0x0000007a52afe000]
java.lang.Thread.State: RUNNABLE
    at applications.trackgamecircular.strategy.RecordStrategy.arrange(RecordStrategy.java:48)
    at applications.trackgamecircular.TrackGame.arrangeOrbit(TrackGame.java:395)
    at applications.trackgamecircular.TrackGame.gameMain(TrackGame.java:144)
    at applications.trackgamecircular.TrackGame.main(TrackGame.java:430)

```

3.5.6 使用设计模式进行代码性能优化

3.5.6.1.1 flyweight 设计模式

```

public class ParticleFactroy {
    private Map<String, Particle> particleMap = new HashMap<String, Particle>();

    public void saveParticle(String name, Particle particle) {
        particleMap.put(name, particle);
    }

    public Particle getParticle(String name) {
        return particleMap.get(name);
    }
}

```

我的电子是基于 Particle 类实现的，现在实现一个 ParticleFactroy，能根据传入不同的名字返回不同的微粒对象（扩展到中子，质子也很好用）。

在使用之前

```

Particle particle=Particle.getElectron();
particleFactroy.saveParticle("Electon", particle);

```

先 save 一个 Electon 到电子的对应。

之后每次需要一个电子对象就用 particleFactroy.getParticle 方法。传入"Electon"。

```
for (int j = 0; j < objNum; j++) {
```

```
    Particle p = particleFactroy.getParticle("Electon");
```

```
    currentList.add(p);
```

3.5.6.1.2 prototype 模式

因为在安排比赛的过程中会重复声明很多有一样的数目的轨道的 Orbit，所以使用 prototype 设计模式

建立一个新的类管理所有的 prototype

```
public class TrackOrbitPrototypeManager {
    private Map<Integer, TrackCircularOrbit> prototypeMap =
        new HashMap<Integer, TrackCircularOrbit>();

    public void addPrototype(Integer tracknum, TrackCircularOrbit orbit) {
        prototypeMap.put(tracknum, orbit);
    }

    public TrackCircularOrbit getPrototype(Integer tracknum) {
        return prototypeMap.get(tracknum).clone();
    }

    public boolean containsKey(Integer tracknum) {
        return prototypeMap.containsKey(tracknum);
    }
}
```

之后每次建立具有一样的轨道数目的 Orbit，先检查下有没有这样的 prototype，有就直接 clone 一个，没有就构建好之后设置为 prototype

```
public void bulidTracks(List<Track> trackList) {
    Integer size = trackList.size();
    if (prototypeManager.containsKey(size)) { //如果有这样的prototype
        concreteCircularOrbit = prototypeManager.getPrototype(size); //直接clone
    } else { //如果没有这样的prototype
        for (Track t : trackList) {
            concreteCircularOrbit.addTrack(t);
        }
        prototypeManager.addPrototype(size, (TrackCircularOrbit) concreteCircularOrbit); //就设置为新的prototype
    }
}
```

3.5.6.1.3 大量使用 hashmap

相比于其他的 collection 查找，hashmap 快得不是一星半点，
以前我把 person 保存在一个 list 里，每次读取名字都要遍历整个 list 查找

```
// while (iterator.hasNext()) {  
// Person person = iterator.next();  
// if (person.getName().equals(keeper.getFromString())) {  
// p1 = person;  
// }
```

之后的改进：

建立一个名字->person 对象的 map，每次只需要 get 一下，快了很多。

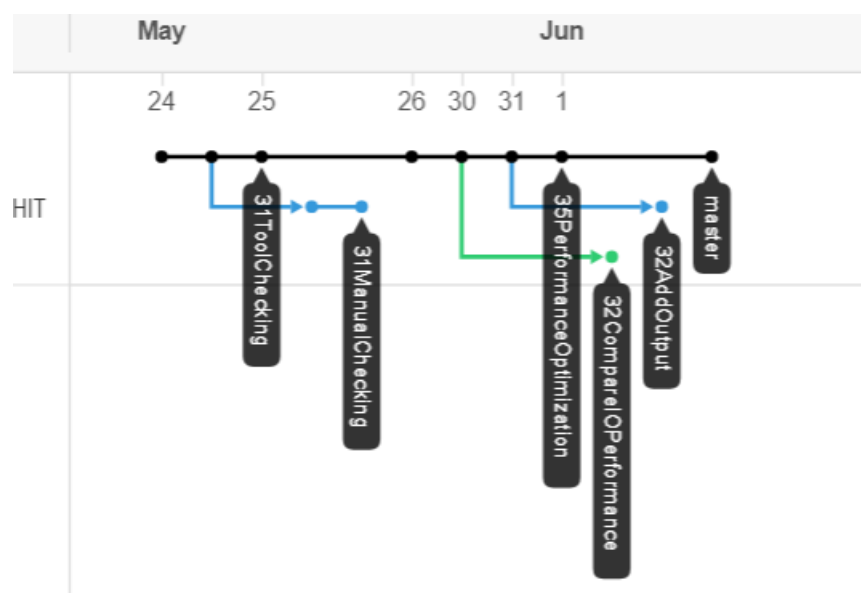
```
Map<String, Person> personNameSet = new HashMap<>();  
for (Person p : personSet) {  
    personNameSet.put(p.getName(), p);  
}
```

3.5.6.1.4 其他的优化细节

包括使用 stringbuilder 代替“+”，用 int 替换一些 Integer，用 double 替换 Double 等等。

3.6 Git 仓库结构

注：一开始写 io 的时候就是按照 32CompareIOPerformance 的要求来写的所以忘记了 32AddOutput，所以这个分支会出现在 32CompareIOPerformance 的后面是后来补的。但是关系不大。



4 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
5.24	19:00-23:00	3.1	完成
5.25	8:00-23:00	3.2	完成
5.26	8:00-23:00	3.3	未完成，visualvm 无法启动
5.27	19:00-23:00	3.3	完成
5.31	15:00-23:00	3.4	未完成
6.1	8:00-23:00	3.4+3.5	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
Visualvm 用不了了	重装 jdk

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

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？

综合考虑。看情况。

- (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？

会

- (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？

暂时还不能，复习的时候再体会一下

- (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？

计算机系统好像没有分区策略。

- (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？

手动麻烦但是上限高，自动简单但是上限低。

- (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？

有，垃圾回收时间从 220+ms 降到了 110+ms.

- (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？

有的工具有些老旧甚至不用了希望更新一下，一直在跟进的工具还是很好的。

- (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？

希望我有。

- (9) 关于本实验的工作量、难度、deadline。

比较麻烦，很多工具没用过需要自学，而且有的工具不知道出什么 bug（visulavm），也查不到解决方法。

- (10) 到目前为止，你对《软件构造》课程的意见与建议。

很花时间。收获不少。