

**2018年春季学期  
计算机学院大二软件构造课程**

**Lab 5实验报告**

|  |  |
| --- | --- |
| 姓名 | 郭琳 |
| 学号 | 1160300730 |
| 班号 | 1603007 |
| 电子邮件 | 1915391067@qq.com |
| 手机号码 | 18845150271 |

**目录**

[1 实验目标概述 1](#_Toc513573923)

[2 实验环境配置 1](#_Toc513573924)

[3 实验过程 1](#_Toc513573925)

[3.1 Static Program Analysis 1](#_Toc513573926)

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

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

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

[3.2.1 多种I/O实现方式 1](#_Toc513573930)

[3.2.2 多种I/O实现方式的效率对比分析 2](#_Toc513573931)

[3.3 Java Memory Management and Garbage Collection (GC) 3](#_Toc513573932)

[3.3.1 使用-verbose:gc参数 3](#_Toc513573933)

[3.3.2 用jstat命令行工具的-gc和-gcutil参数 3](#_Toc513573934)

[3.3.3 使用jmap -heap命令行工具 3](#_Toc513573935)

[3.3.4 使用jmap -histo命令行工具（可选） 3](#_Toc513573936)

[3.3.5 使用jmap -permstat命令行工具（可选） 3](#_Toc513573937)

[3.3.6 使用jconsole或VisualVM工具 3](#_Toc513573938)

[3.3.7 分析垃圾回收过程是否正常、异常 3](#_Toc513573939)

[3.3.8 配置JVM参数并发现最优参数配置 3](#_Toc513573940)

[3.4 Dynamic Program Profiling 3](#_Toc513573941)

[3.4.1 使用Visual VM进行CPU Profiling 3](#_Toc513573942)

[3.4.2 使用Visual VM进行Memory profiling 3](#_Toc513573943)

[3.5 Memory Dump Analysis and Performance Optimization 3](#_Toc513573944)

[3.5.1 内存导出(memory dump) 3](#_Toc513573945)

[3.5.2 使用MAT分析内存导出文件 3](#_Toc513573946)

[3.5.3 发现热点/瓶颈并改进、改进前后的性能对比分析 3](#_Toc513573947)

[3.5.4 jhat和OQL查询内存导出（可选） 4](#_Toc513573948)

[3.5.5 jstack导出java程序运行时的调用栈（可选） 4](#_Toc513573949)

[4 实验进度记录 4](#_Toc513573950)

[5 实验过程中遇到的困难与解决途径 4](#_Toc513573951)

[6 实验过程中收获的经验、教训、感想 4](#_Toc513573952)

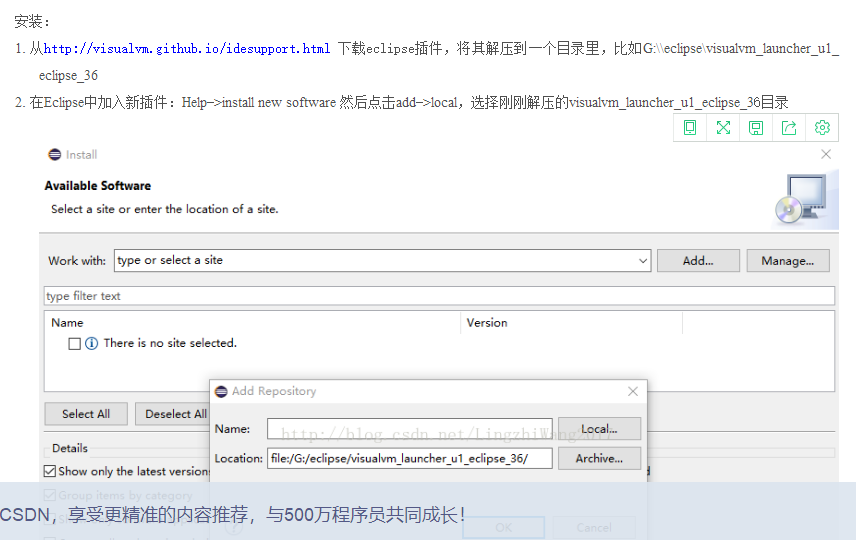
# 实验目标概述

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

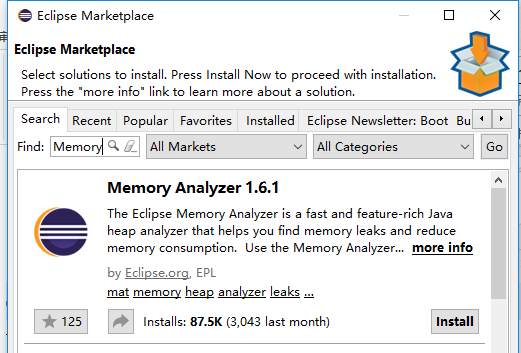
# 实验环境配置

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。



安装MAT ：



在这里给出你的GitHub Lab5仓库的URL地址（Lab5-学号）。

https://github.com/ComputerScienceHIT/Lab5-1160300730.git

# 实验过程

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

## Static Program Analysis

### 人工代码走查（walkthrough）

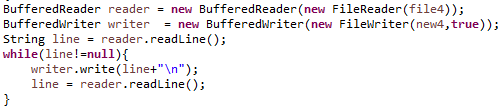
比如空行问题和缩进不正确的问题，可以直接改正。

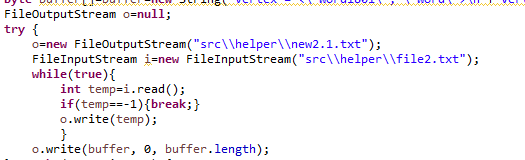
### 使用CheckStyle和FindBugs进行静态代码分析

## Java I/O

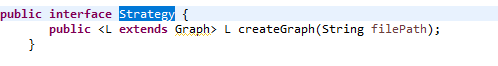
### 多种I/O实现方式

我用了策略1：Reader/Writer和策略2：Stream来进行对文件的读写。

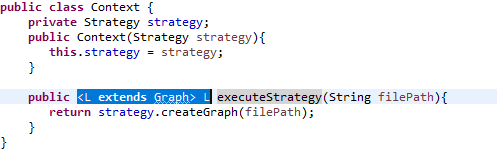




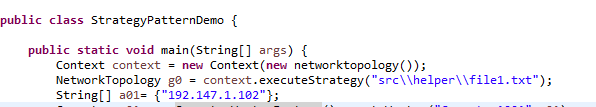
策略模式先定义一个接口名为Strategy



客户端可使用的context



在strategypatterdemo进行调用



### 多种I/O实现方式的效率对比分析

如何收集你的程序I/O语法文件的时间。

我是利用了test来测试每个方法的时间。

表格方式对比不同I/O的性能。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | File1 | File2 | File3 | File4 |
| IO策略1 | 读写文件 | 1828.680s | 4.304s | 3049.024s | 1257.180s |
| IO策略2 | 读写文件 | 4134.446s | 191.493s | 3441.009s | 2307.336s |
| IO策略3 | 读写文件 |  |  |  |  |

图形对比不同I/O的性能（可选）。

## Java Memory Management and Garbage Collection (GC)

### 使用-verbose:gc参数

[Full GC (Ergonomics) 55909K->54824K(149504K), 0.2188027 secs]

Full gc出现频率比较小

### 用jstat命令行工具的-gc和-gcutil参数

### 使用jmap -heap命令行工具

### 使用jmap -histo命令行工具（可选）

### 使用jmap -permstat命令行工具（可选）

### 使用jconsole或VisualVM工具

jvisualvm.exe

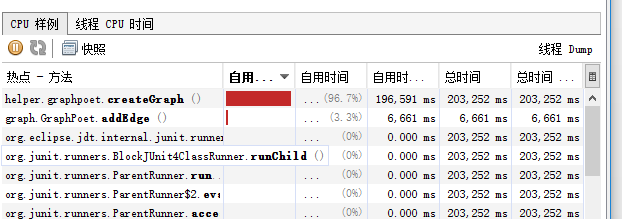
### 分析垃圾回收过程是否正常、异常

正常

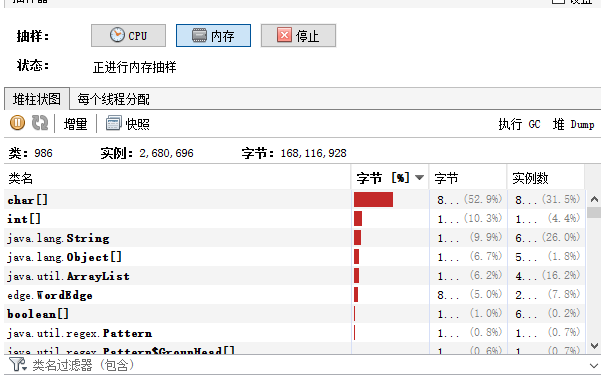
### 配置JVM参数并发现最优参数配置

## Dynamic Program Profiling

### 使用Visual VM进行CPU Profiling



### 使用Visual VM进行Memory profiling



## Memory Dump Analysis and Performance Optimization

### 内存导出(memory dump)

### 使用MAT分析内存导出文件

### 发现热点/瓶颈并改进、改进前后的性能对比分析

### jhat和OQL查询内存导出（可选）

### jstack导出java程序运行时的调用栈（可选）

# 实验进度记录

请尽可能详细的记录你的进度情况。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 6.4前 |  | 完成 |  |
|  |  |  |  |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

百度

# 实验过程中收获的经验、教训、感想

又接触到了几个新的软件工具