**哈尔滨工业大学计算机科学与技术学院**

**2016年秋季学期《软件工程》**

**Lab 4：代码评审与程序性能优化**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| 冯掌印 | 1140310418 | fzybcg@163.com/18846448814 |
| 马一丰 | 1140310405 | 993030067@qq.com/18245029647 |

**目 录**

[1 实验要求 1](#_Toc463000523)

[2 在Eclipse中配置代码审查与分析工具 1](#_Toc463000524)

[2.1 Checkstyle 1](#_Toc463000525)

[2.2 PMD 1](#_Toc463000526)

[2.3 FindBugs 1](#_Toc463000527)

[2.4 VisualVM 1](#_Toc463000528)

[3 本次实验所评审的代码 1](#_Toc463000529)

[4 代码review记录 1](#_Toc463000530)

[5 Checkstyle所发现的代码问题清单及原因分析 2](#_Toc463000531)

[6 PMD所发现的代码问题清单及原因分析 2](#_Toc463000532)

[7 FindBugs所发现的代码问题清单及原因分析 2](#_Toc463000533)

[8 VisualVM性能分析结果 2](#_Toc463000534)

[8.1 执行时间的统计结果与原因分析 2](#_Toc463000535)

[8.2 内存占用的统计结果与原因分析 2](#_Toc463000536)

[8.3 代码改进之后的执行时间统计结果 3](#_Toc463000537)

[8.4 代码改进之后的内存占用统计结果 3](#_Toc463000538)

[9 利用Git/GitHub进行协作的过程 3](#_Toc463000539)

[10 评述 3](#_Toc463000540)

[10.1 对代码规范方面的评述 3](#_Toc463000541)

[10.2 对代码性能方面的评述 4](#_Toc463000542)

[11 计划与实际进度 4](#_Toc463000543)

[12 小结 4](#_Toc463000544)

[文档全部完成之后，请更新上述区域]

# 实验要求

针对Lab1所完成的代码，进行代码评审(走查)和性能分析，从性能角度对代码进行优化；

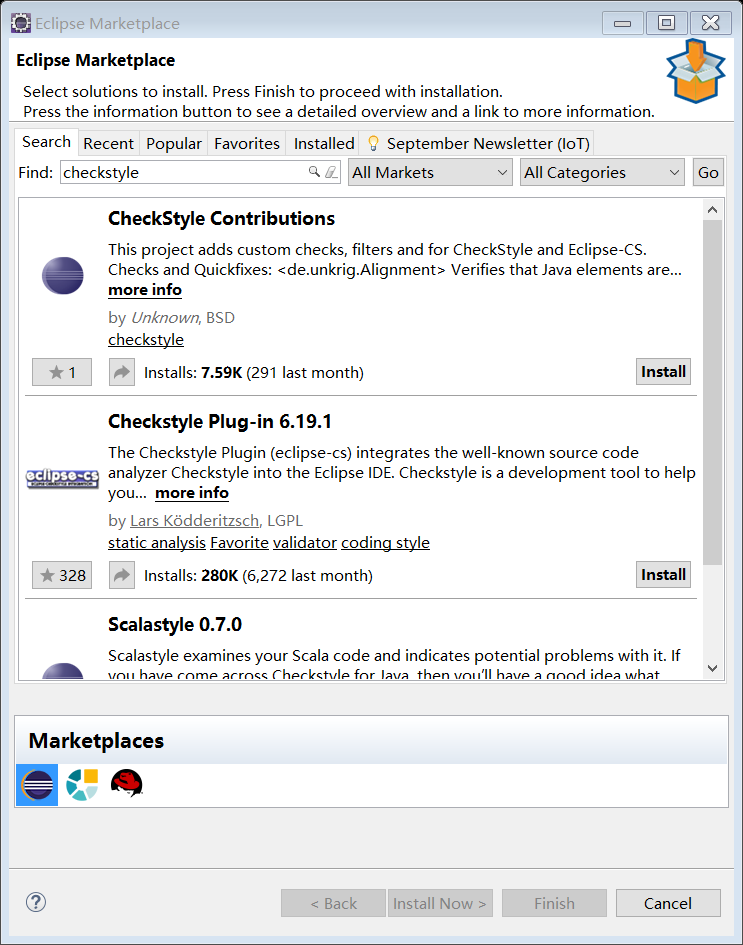
练习代码评审的两个方面：静态分析、动态分析(profiling)；

# 在Eclipse中配置代码审查与分析工具

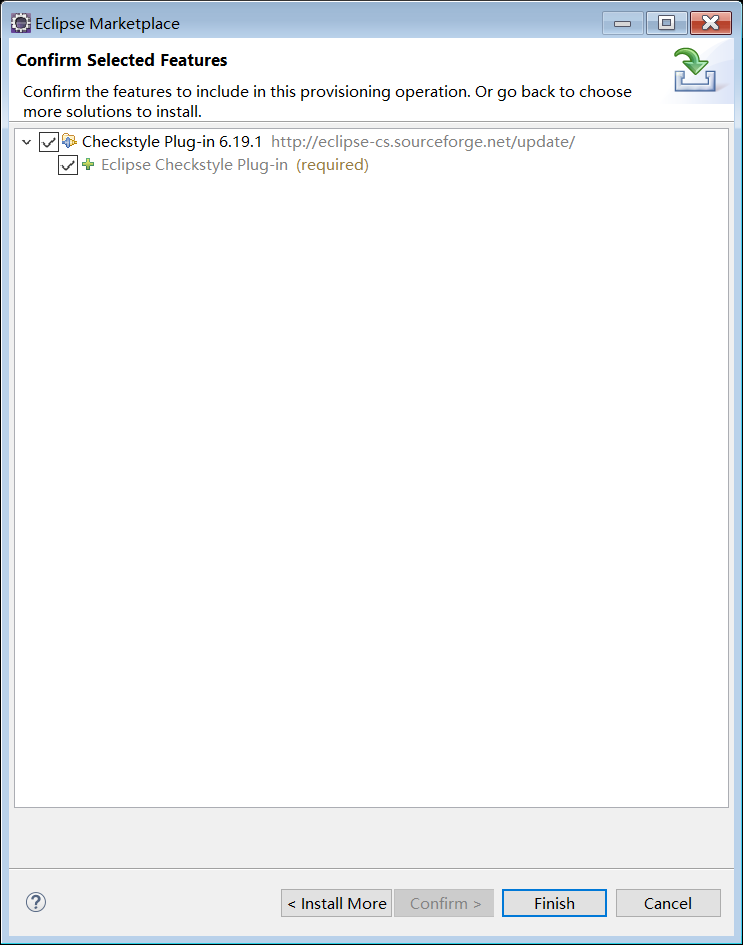
采用屏幕截图的方式给出在Eclipse中配置Checkstyle、PMD、Findbugs的过程、安装VisualVM的过程。

## Checkstyle

我使用了不同于实验手册中的安装方法，在Help/Eclipse Market 中搜索Checkstyle得到Checkstyle plug-in



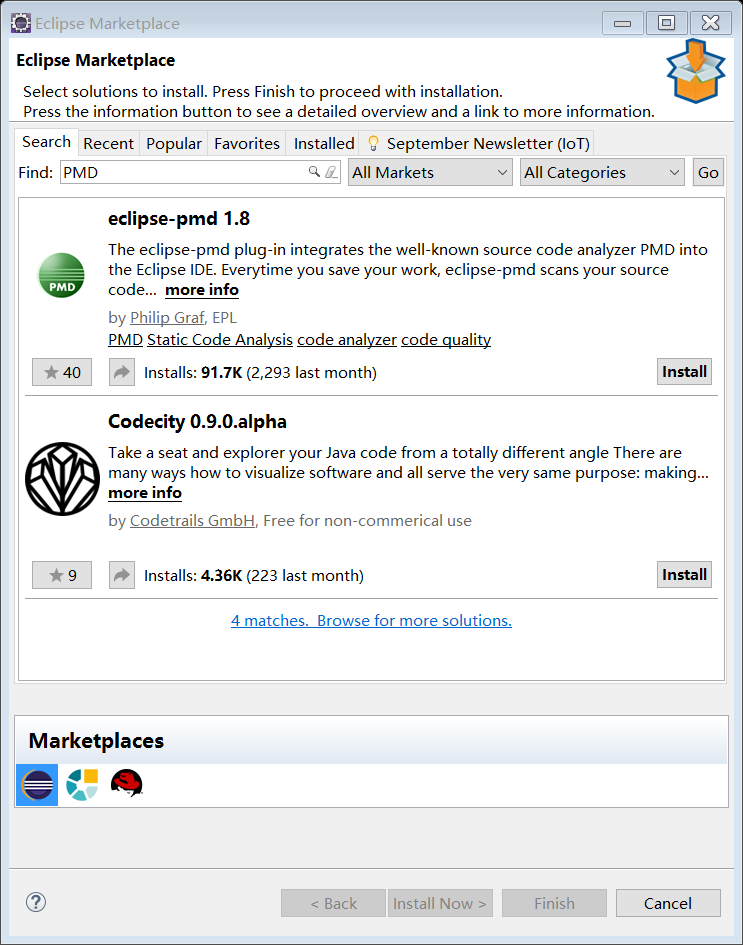
点击安装，出现如下窗口

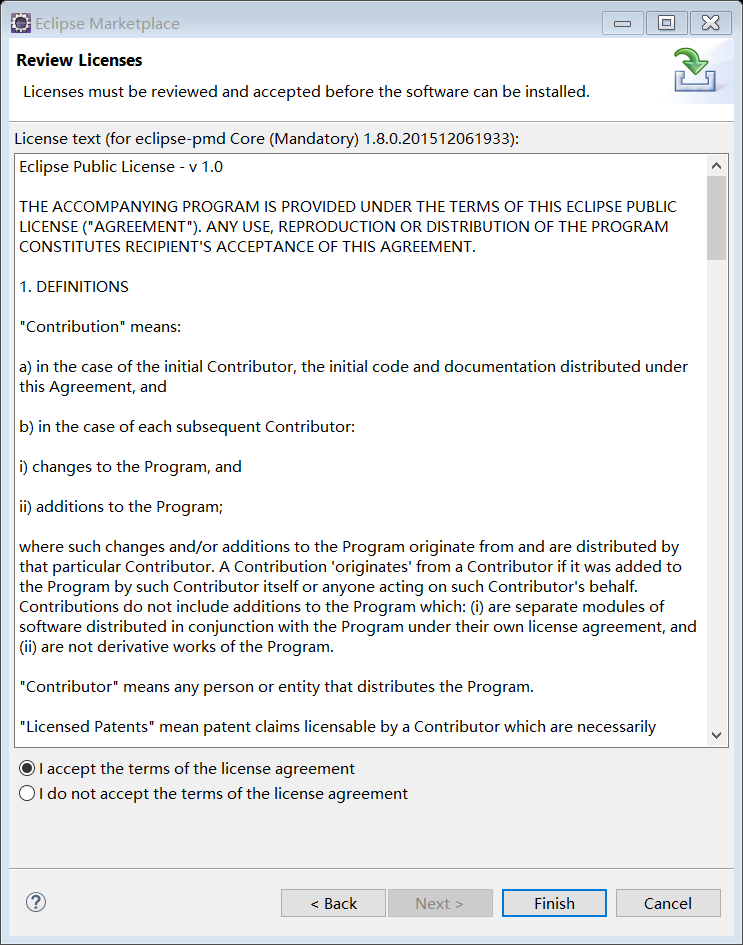


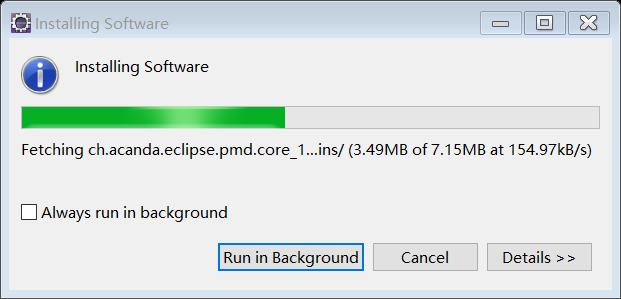
点击finish，即可自动安装。

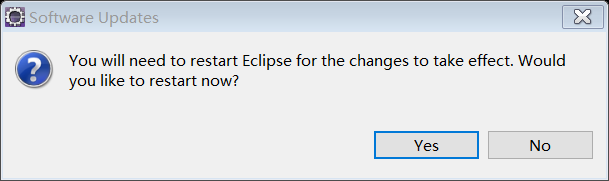
可以利用插件中包含的基本配置（Sun Checks），也可以自定义配置（可选）。

## PMD





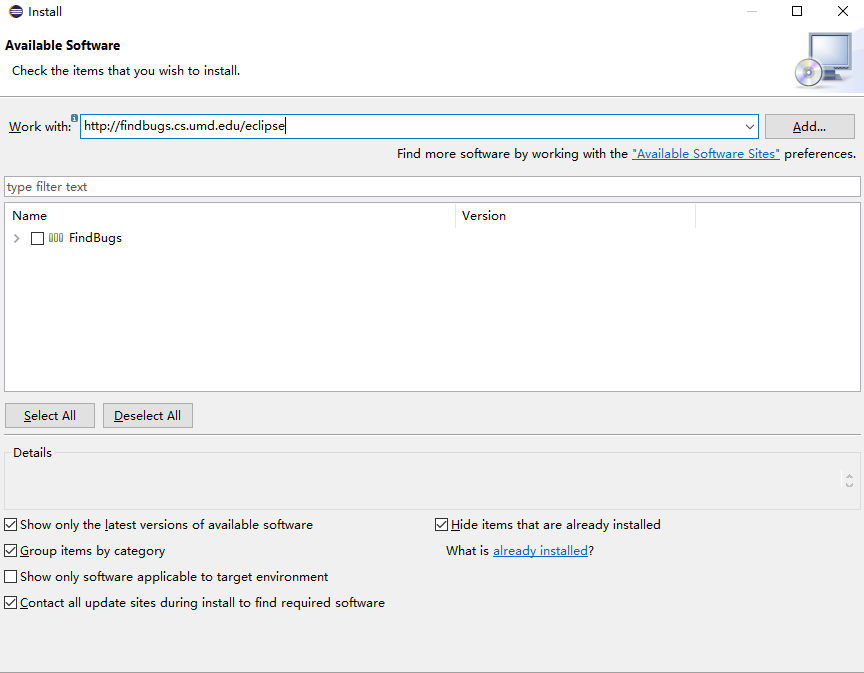




## FindBugs

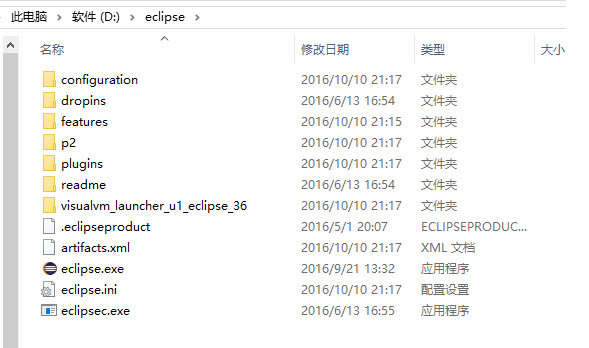
Eclipse中，选择Help->Install New Software，在work with中输入

<http://findbugs.cs.umd.edu/eclipse> 点击next安装

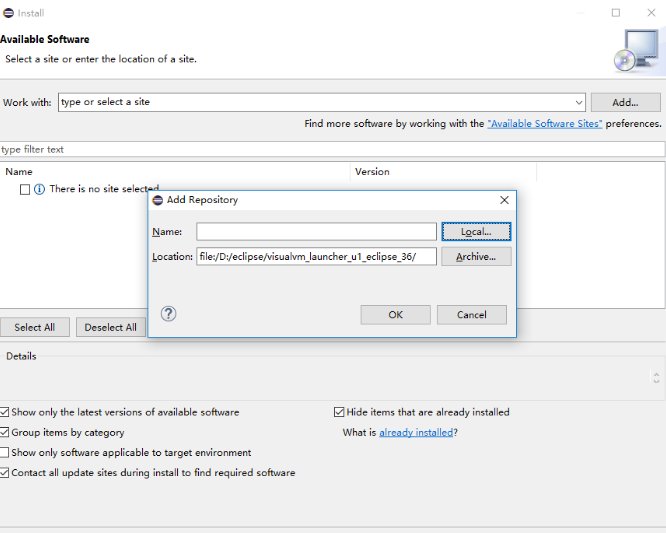


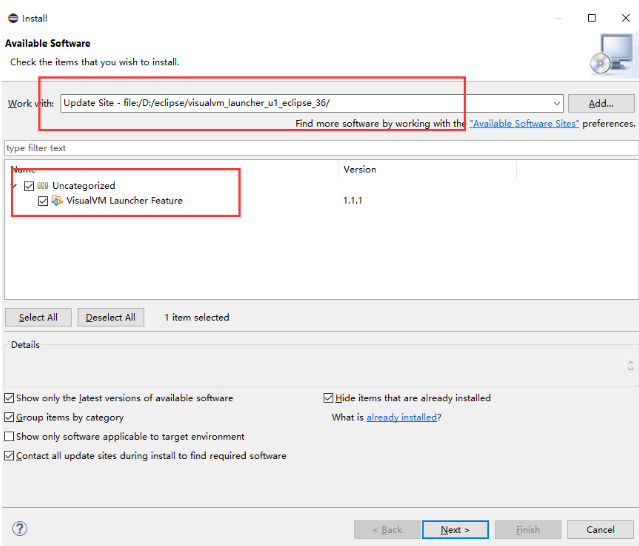
## VisualVM

1. 到官网下载VisualVM包
2. 将visualvm\_launcher\_u1\_eclipse\_36.zip 解压到Eclipse主目录，如图所示：

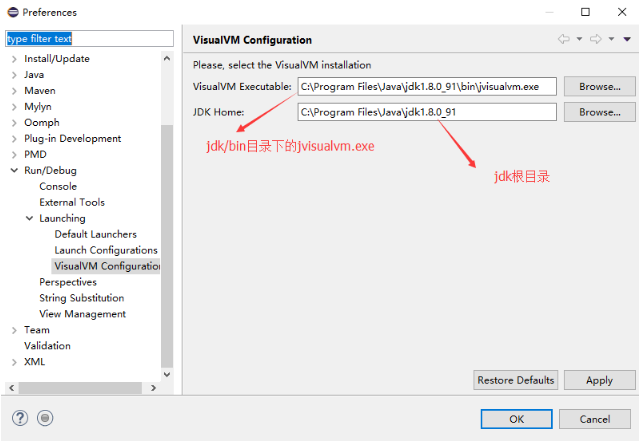


1. 在Eclipse中加入新插件：Help—>install new software 然后add—>local 刚刚解压的visualvm\_launcher\_u1\_eclipse\_36 目录,如图： ok之后显示下边的界面就代表成功了，直接next安装就行了





1. 配置： 在window的preferences中进行VisualVM的配置，需要配置它的启动器（jdk、bin目录下面的jvisualvm.exe）还有jdk目录,如图所示，点击apply，ok即可完成安装配置：

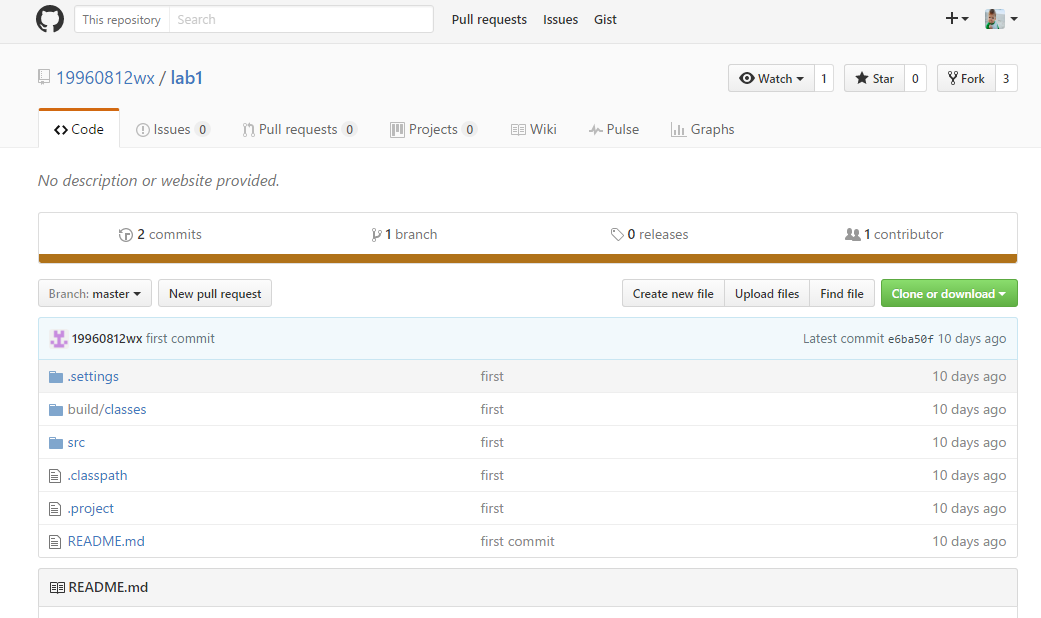


# 本次实验所评审的代码

写清楚原开发者的名字、学号、GitHub地址，给出GitHub上项目的文件清单截图。

汪道远（1140310117）/王雄（1140310105）

<https://github.com/19960812wx/lab1>



# 代码review记录

两个人对待评审代码进行review（不借助任何工具），在下表中记录所发现的问题，对代码进行修改。针对同种类型的问题，只需要列出一个典型代表即可。

|  |  |  |  |
| --- | --- | --- | --- |
| 问题描述 | 类型 | 所在代码行号 | 修改方式 |
| 引入无用包 | 内容规范 无用引用 | 4，5 | 删除无用引用 |
| 命名不合理，可读性差 | 形式规范 命名 | 6 | 修改命名 |
| 声明了未使用的局部变量 | 内容规范 | 10 | 删除声明 |
| 代码重复度高 | 内容规范 | 51-60，60-70 | 修改代码结构 |

# Checkstyle所发现的代码问题清单及原因分析

针对同种类型的问题，只需要列出一个典型代表即可。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 问题描述 | 类型 | 所在代码行号 | 修改策略 |
| 1 | 缺少文档注释 | 形式规范 | 82 | 添加文档注释 |
| 2 | 逗号之后缺少空格 | 形式规范 | 82 | 逗号之添加空格 |
| 3 | 运算符前后缺少空格 | 形式规范 | 88 | 减号前后添加空格 |
| 4 | 缺少或多余空行 | 形式规范 | 93 | 更改空行 |
| 5 | Resource leak: 's' is never closed | 内容规范 | 12 | 增加关闭s的语句 |
| 6 | The value of the field shiyan6.str is not used | 内容规范 | 7 | 删除未使用的变量 |
| 7 | 'if' construct must use '{}'s. | 形式规范 | 145 | 添加”{}” |
| 8 | Array brackets at illegal position. | 形式规范 | 159 | 更改[]位置 |
| 9 | 参数： str 应定义为 final 的。 | 内容规范 | 134 | 定义为final的 |
| 10 | 本行字符数超过80个。 | 形式规范 | 121 | 通过分行的形式减少该行字符 |
| 11 | '3' 是一个魔术数字（直接常数）。 | 内容规范 | 140 | 将常数定义为常量 |
| 12 | 命名不规范 | 内容规范 | 118 | 修改命名 |

# PMD所发现的代码问题清单及原因分析

针对同种类型的问题，只需要列出一个典型代表即可。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 优先级 | 问题描述 | 违反的规则集 | 所在代码行号 | 修改策略 |
|  | 疑似的复制，粘贴 |  | 54-62，64-72 | 调整代码结构 |
| 1 blocker | 变量应以小写字母开头 | VariableNamingConventions | 119 | 修改变量名 |
| 2 critical | 推荐不使用system.out | SystemPrintln | 82 | 实验情况特殊，应忽略这条问题 |
| 3 | 局部变量可以声明为final | LocalVariableCouldBeFinal | 182 | 声明为final |
| 3 | 违反the law of Demeter | LawOfDemeter | 19 | 等学习面向对象后再解决这个问题吧，现在不是很了解the law of Demeter |
| 2 | 避免实例化字符串 | StringInstantiation | 194 | 本程序中需实例化字符串，故忽略这条提示 |
| 3 | 避免使用String.indexOf(String)而使用String.indexOf(char) | UseIndexOfChar | 236 | 修改代码 |
| 3 | 避免在if条件判断中使用文字 | AvoidLiteralsInIfCondition | 29 | 本程序中必须如此，忽略这条提示 |
| 3 | 推荐使用 StringBuffer 的 += 来连接字符串 | UseStringBufferForStringAppends | 268 | 本程序特殊需要，因为之前要用到split方法，故忽略这条提示 |
| 3 | 避免变量名过短 | ShortVariable | 149 | 有些迭代变量命名为短变量名较为合适，故忽略这条提示 |
| 3 | 变量名过长 | LongVariable | 193 | 修改变量名 |
| 4 | 参数可以声明为final的 | MethodArgumentCouldBeFinal | 15 | 对可修改的进行修改，对不宜修改的忽略该提示 |
| 3 | 代码复杂度过高 | StdCyclomaticComplexity | 10 | 修改代码 |
| 3 | 所有方法都是静态的，推荐使用utility class | UseUtilityClass | 10 | 修改代码 |
| 3 | 过早的定义了变量 | PrematureDeclaration | 66 | 修改 |
| 3 | A method should have only one exit point, and that should be the last statement in the method | OnlyOneReturn | 97 | 程序需要，可以忽略 |
| 3 | 需要注释 | CommentRequired | 23 | 需要注释的添加注释，有些不需要的忽略提示 |
| 5 | 重复定义变量 | DataflowAnomalyAnalysis | 77 | 修改代码 |
| 5 | 定义的变量超出作用域变成未定义的 | DataflowAnomalyAnalysis | 90 | 修改代码 |

# FindBugs所发现的代码问题清单及原因分析

针对同种类型的问题，只需要列出一个典型代表即可。

|  |  |  |  |
| --- | --- | --- | --- |
| 问题描述 | 类型 | 所在代码行号 | 修改策略 |
| 使用非短路逻辑符存在潜在风险 | Dodgy code | 101 | 代码中使用了&符，将其改为&&即可 |

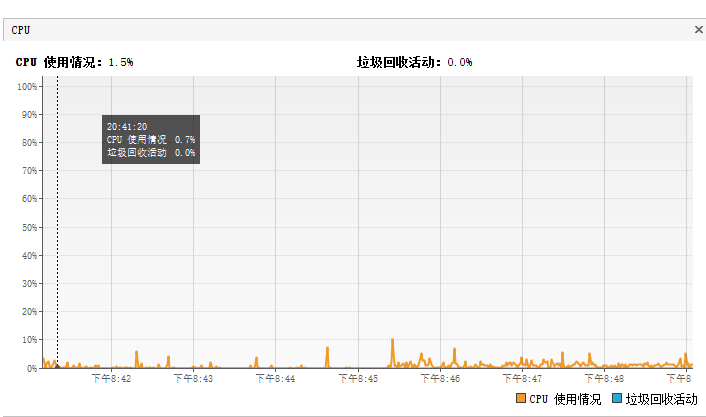
# VisualVM性能分析结果

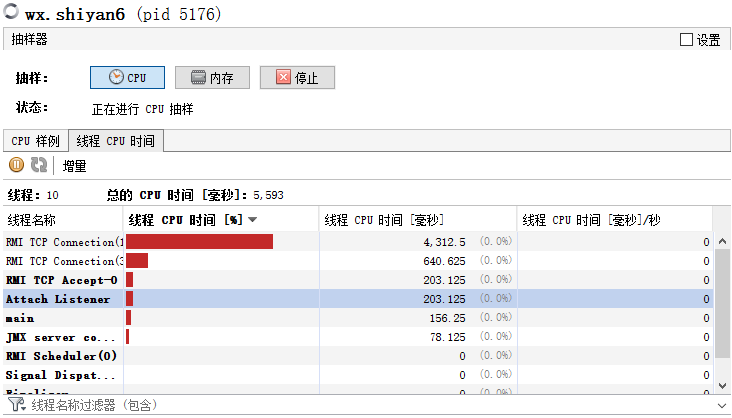
## 执行时间的统计结果与原因分析

给出图示

分析某些函数被调用次数和执行时间较高的原因

阐述如何改进





main()函数执行时间较长是因为该函数调用了其余所有函数。

expression函数执行时间稍长是因为该函数包含过多相似循环代码，可通过修改代码结构，整合循环来提高运行效率。

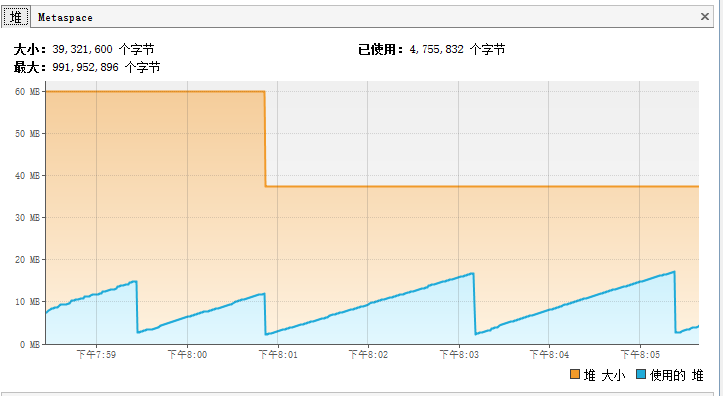
derivative和simplify函数执行时间较长是因为该函数进行了循环的嵌套和多次循环，可通过修改代码结构，尽可能减少嵌套和循环次数，提高程序运行效率。

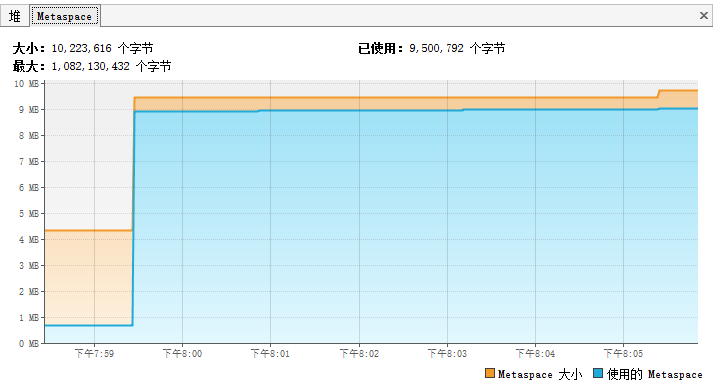
## 内存占用的统计结果与原因分

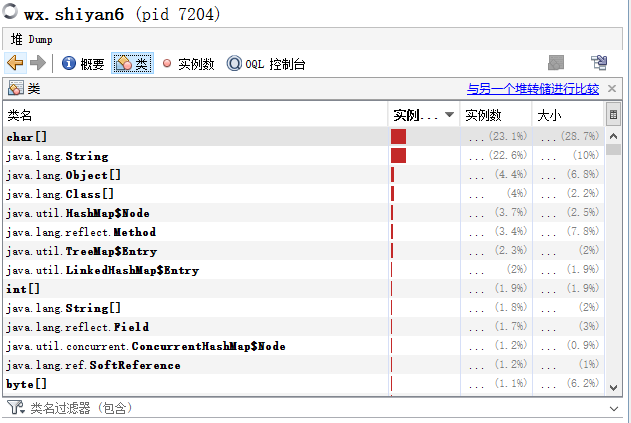
给出图示

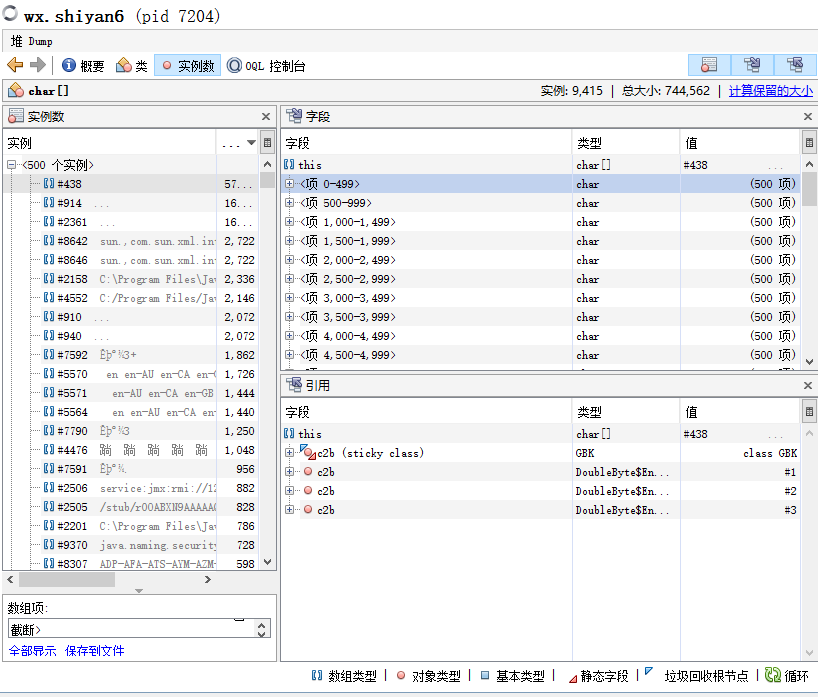
分析某些函数所需内存较高的原因

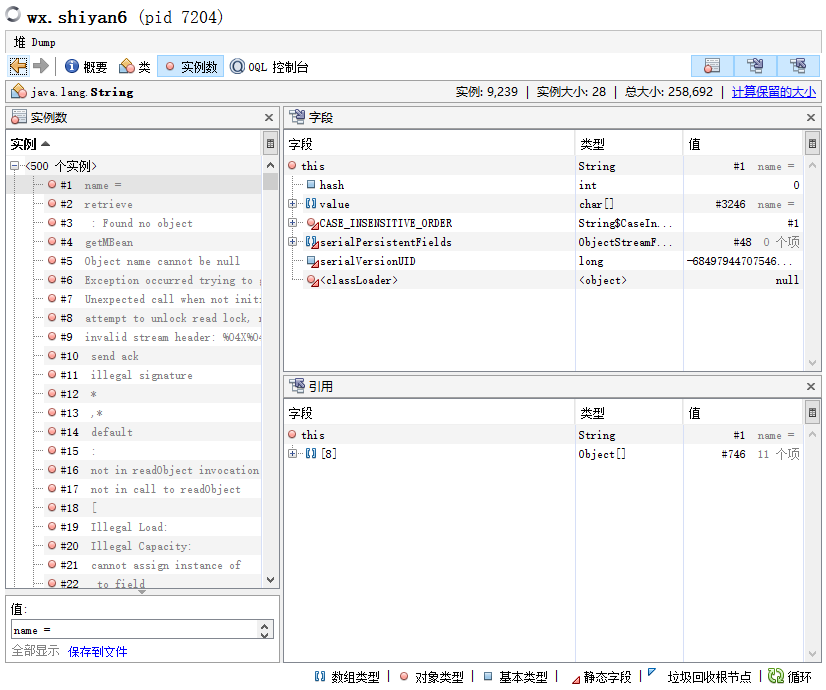
简要分析如何改进









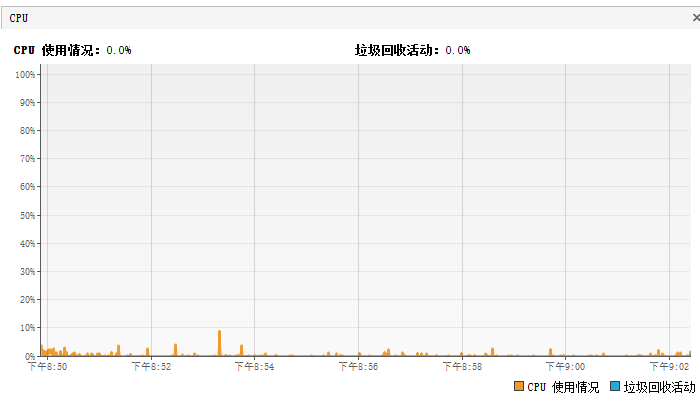


expression函数占用内存较多是因为该函数声明了int[]数组和几个String，导致占用交大内存。

## 代码改进之后的执行时间统计结果

改进代码之后，重新运行VisualVM获取结果，给出图示

分析改进的幅度、是否还可能继续改进



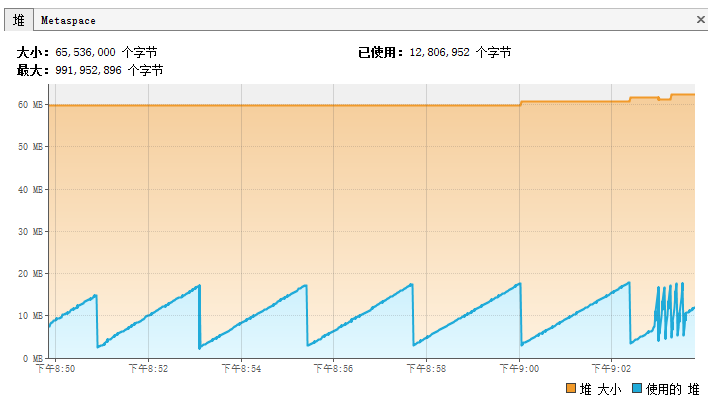


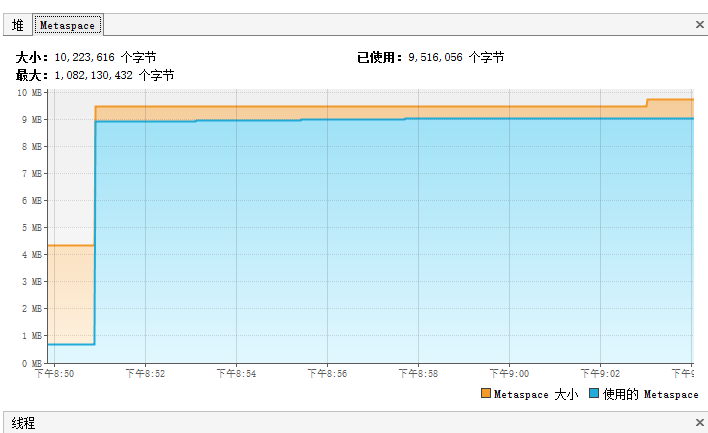
由改进之后的运行效果图与改进之前的运行效果图对比可知，改进之后运行效率更高，提高幅度较大，此时CPU使用已经很低，不过仍有改进的可能，但是改进幅度不会很大。

## 代码改进之后的内存占用统计结果

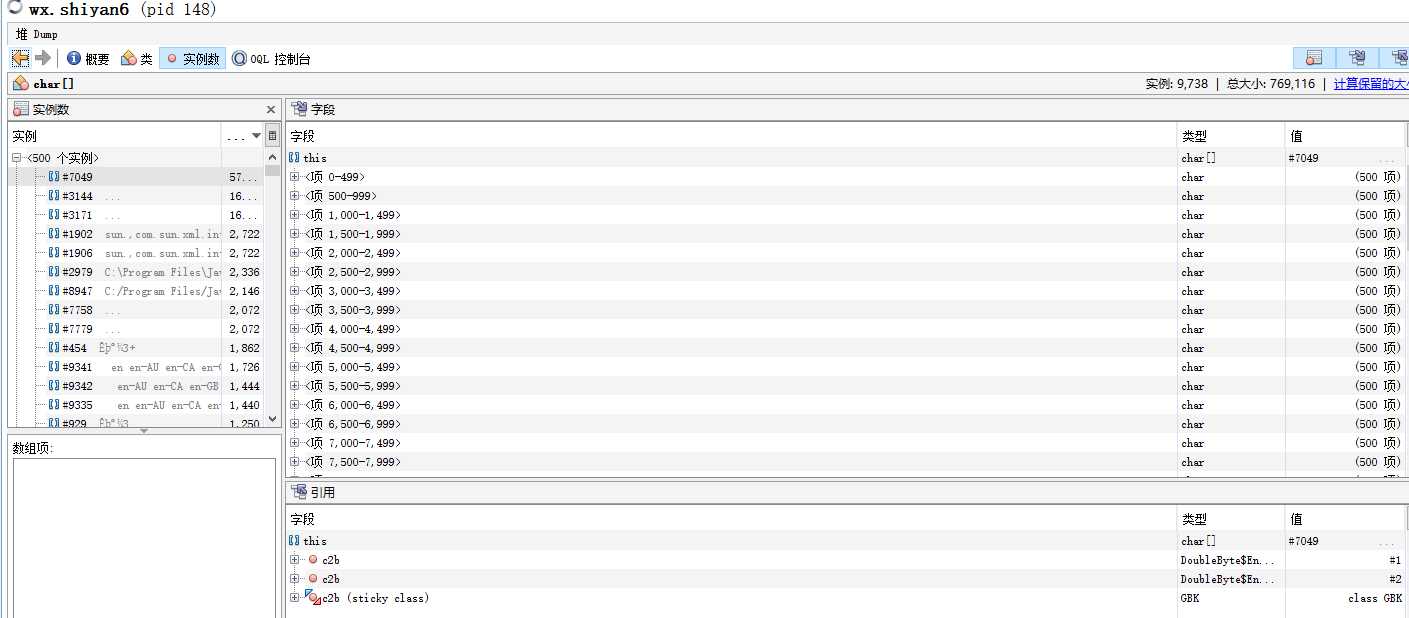
改进代码之后，重新运行VisualVM获取结果，给出图示

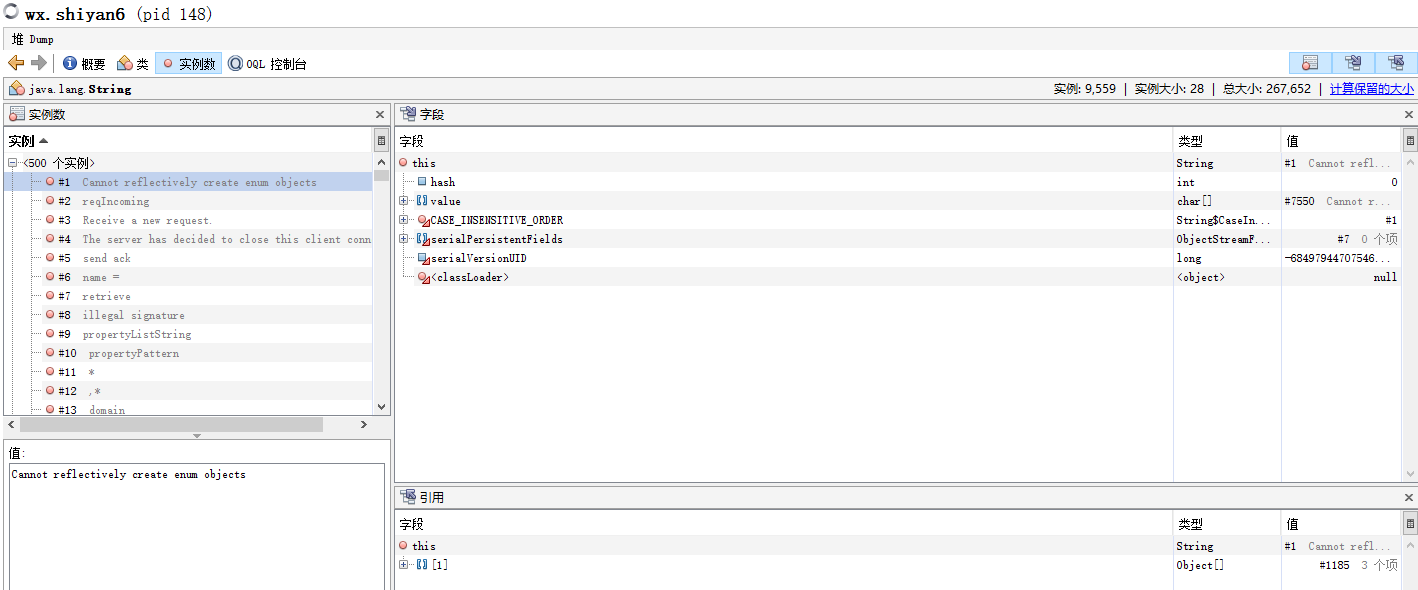
分析改进的幅度、是否还可能继续改进











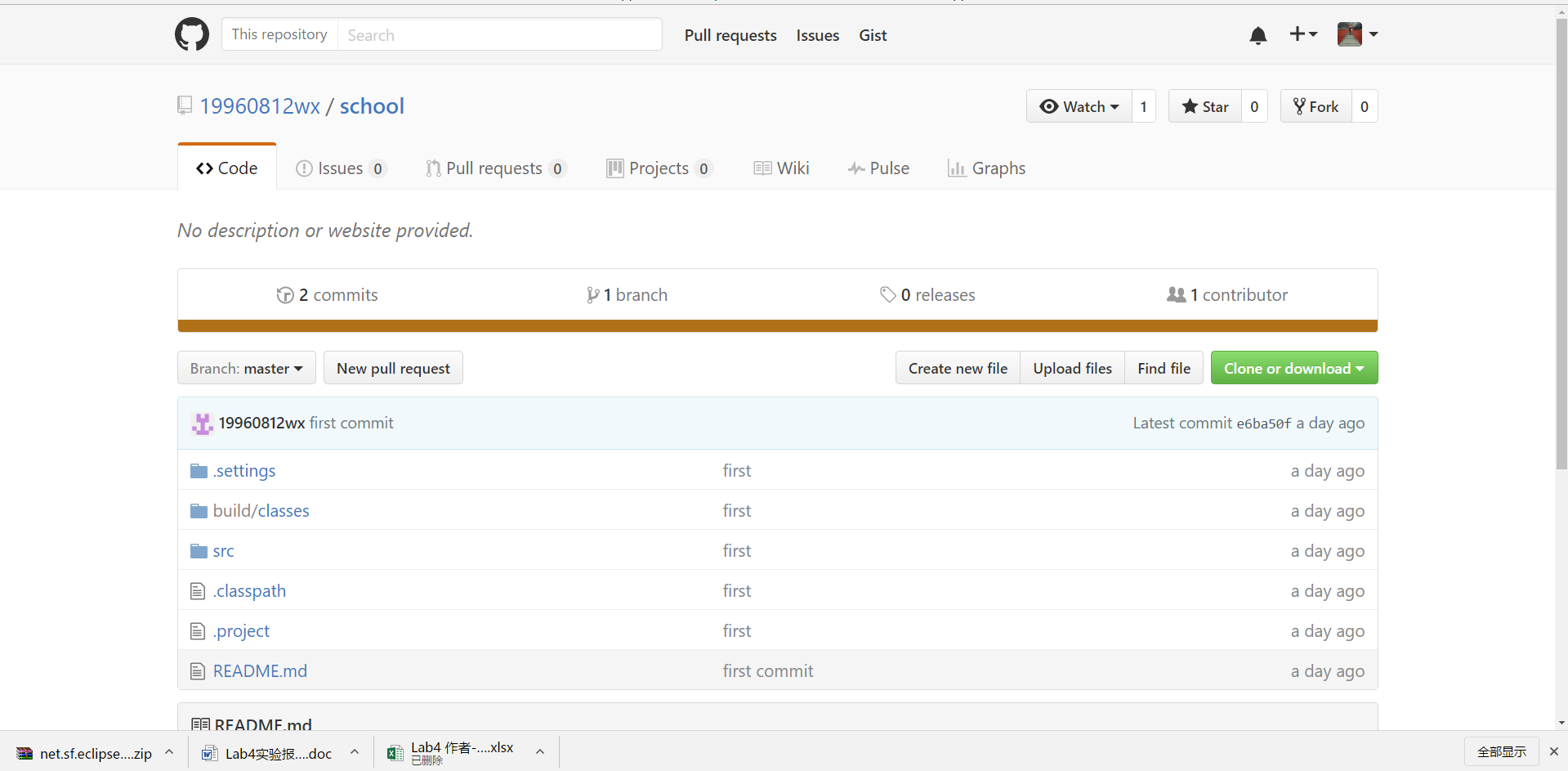
由于程序本身复杂性的限制导致内存占用提高不是很高，仅有少许提高，但是改进之后的代码较改进之前进行了功能的完善与鲁棒性的提高，虽然程序内存占用仍有提高可能，但是提高幅度不会很大。

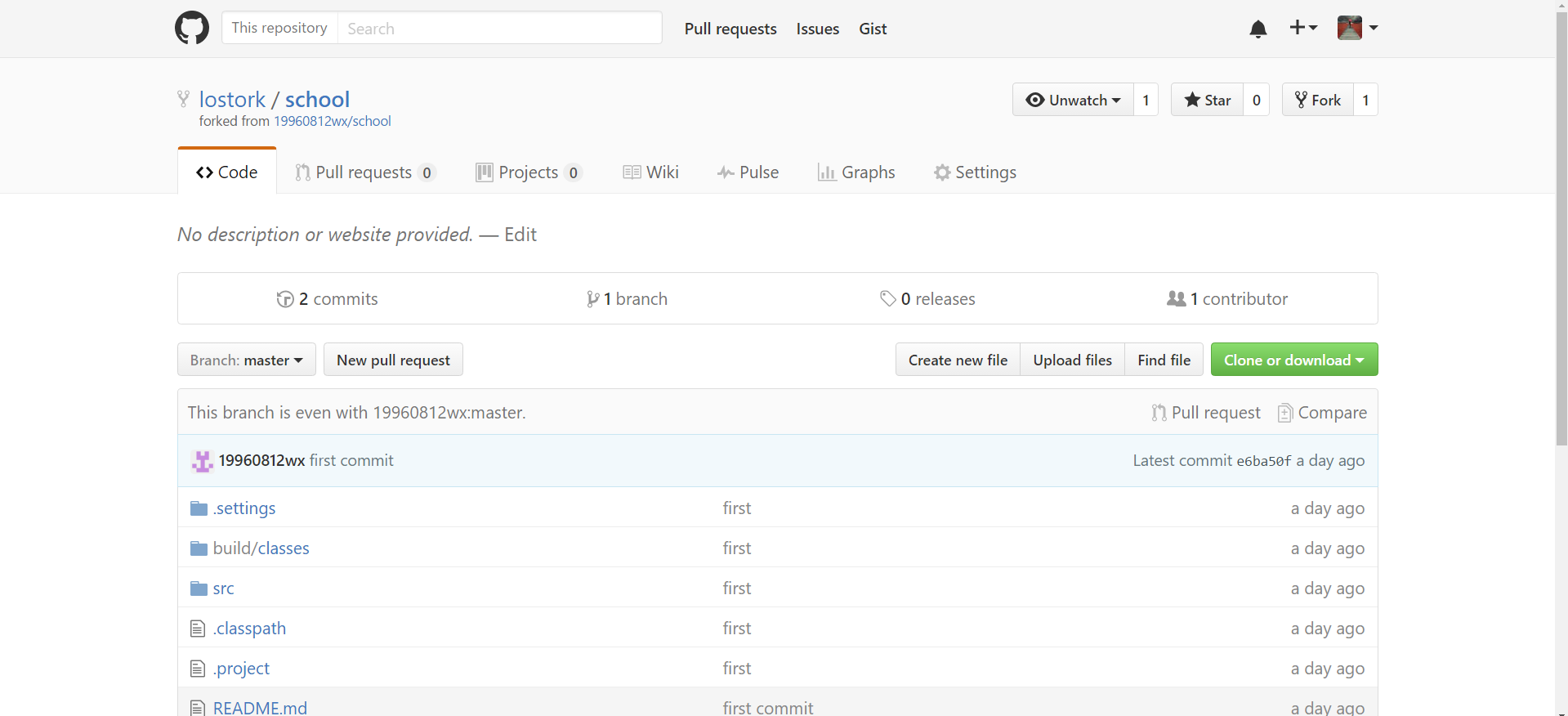
# 利用Git/GitHub进行协作的过程

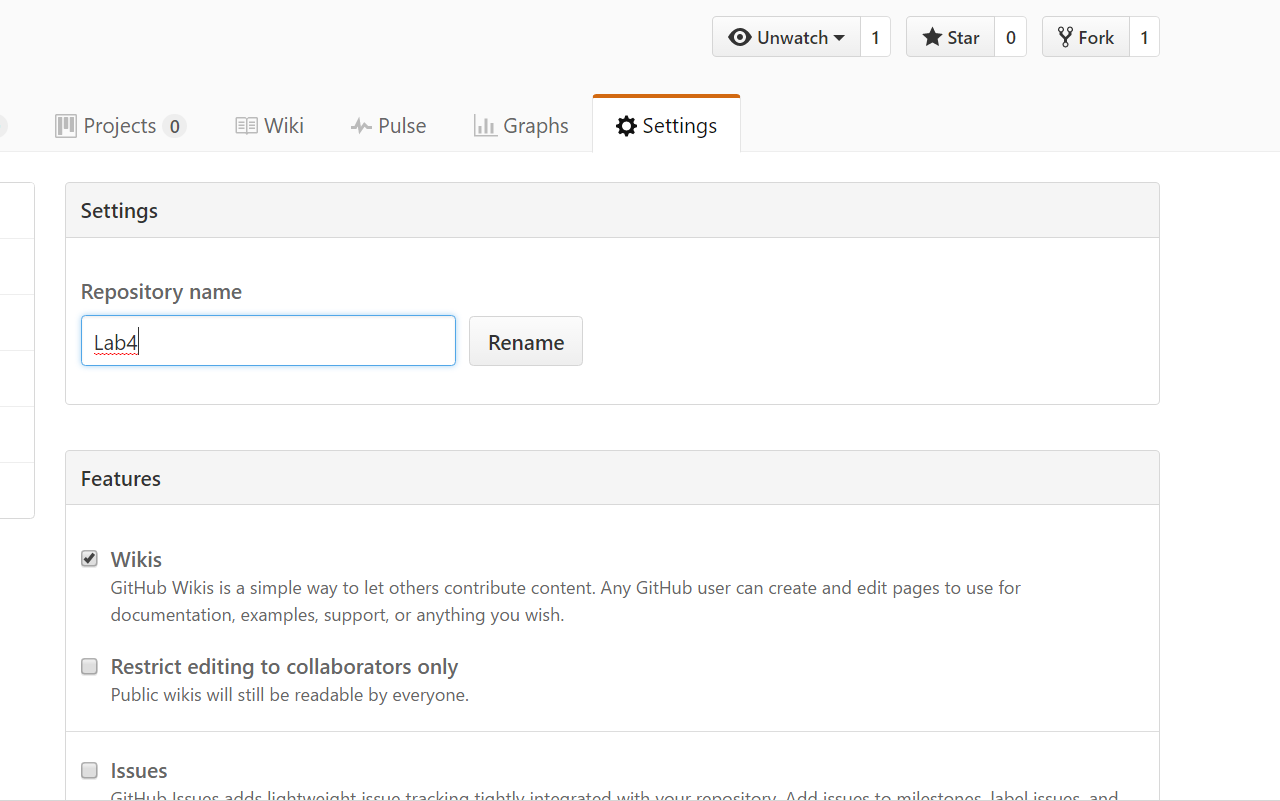
给出利用Git/GitHub完成以下步骤的截图：

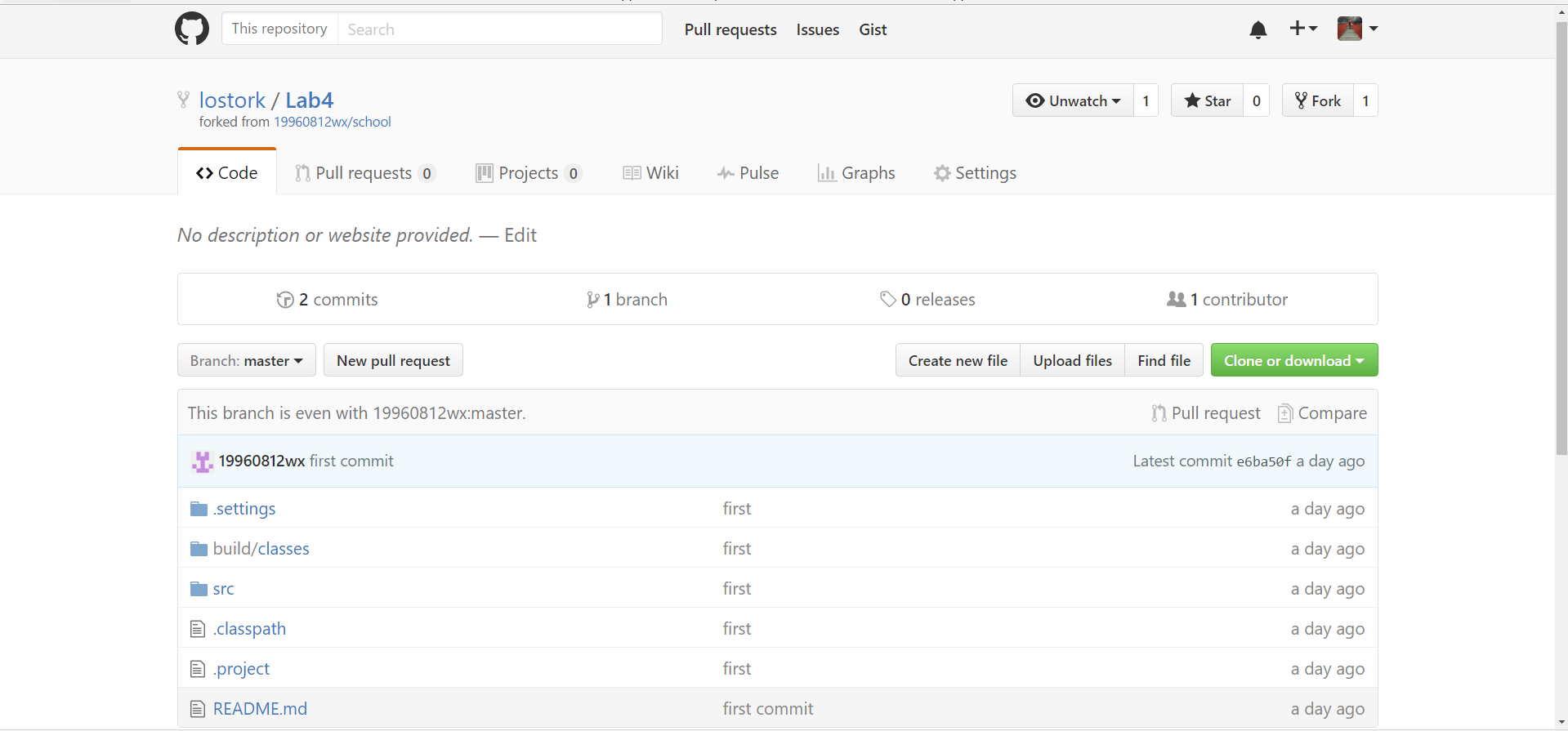
Part 1：

* 在GitHub上找到待评审代码，fork至本组GitHub仓库内并重命名为Lab4；

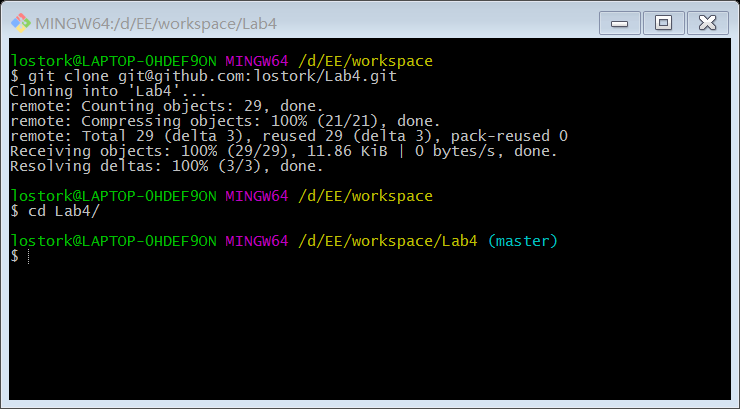








* 将fork得到的仓库Lab4 clone至本组的本地仓库Lab4；



* 人工review和工具review之后将修改提交至本地仓库；
* 将所有修改历史push到GitHub上本组仓库Lab4；
* 在Github上将最新的提交Pull Request到原作者的Lab1仓库；

Part 2：

* 原作者在自己的本地Lab1仓库建立新分支Lab4；
* 原作者在GitHub上查看评审组的pull request并合并至Lab4，将评审组pull request过来的代码合并进去，将Lab4分支fetch到本地Lab1仓库
* 原作者列出评审组所做的所有修改；
* 原作者在Lab4分支上对不认可的修改进行调整，并在Lab4分支上加以提交；
* 原作者将Lab4分支与原来的主分支merge起来，推送至本组GitHub的Lab1仓库。

# 评述

## 对代码规范方面的评述

对你们组所拿到的其他同学的程序在遵循代码风格规范方面的水平做简要评论。

若代码风格较好，给你们本次实验的工作带来哪些方面的好处？

若代码风格较差，给你们本次实验的工作带来哪些方面的坏处？

该组同学的代码风格应该算是一般，整个文件风格能保持统一，但是风格又不是十分规范，

诸如变量命名不规范，缺少必要注释，会定义一些变量但是不使用，代码还具有一定重复度；但是整体上缩进，括号的使用还是比较好的。由于问题较多，导致本次实验需要修改跟多地方，做大量重复的琐碎工作。

## 对代码性能方面的评述

对你们组所拿到的其他同学的程序在代码性能方面的水平做简要评论。

本组同学实现的功能比较简单，没有实现所有功能，不过实现部分代码性能较好，仅需做出较小修改即可，有助于本次实验修改。

# 计划与实际进度

|  |  |  |  |
| --- | --- | --- | --- |
| 任务名称 | 计划时间长度（分钟） | 实际耗费时间（分钟） | 提前或延期的原因分析 |
| 工具配置 | 60 | 40 | 配置比较简单，没有想象中复杂 |
| 代码获取 | 15 | 30 | 该组同学未及时上传代码 |
| 人工review | 30 | 45 | 问题较多，人工较想象中慢 |
| 工具检查 | 300 | 450 | 工具使用不够熟悉，花费一定时间 |
| github操作 | 100 | 120 | git命令不够熟悉，部分指令需要查阅 |

# 小结

回答几个问题：

* FindBugs、PMD、CheckStyles三者都是代码规范静态检查工具，它们之间有何异同？从分析结果看，它们有什么优劣？

**FindBugs**检查.class

基于Bug Patterns概念，查找javabytecode中的潜在bug 主要检查bytecode中的bug patterns，如NullPoint空指针检查、没有合理关闭资源、字符串相同判断错（==，而不是equals）等 FindBugs 大多数提示有用,值得改

配置无查找功能，不过缩写能让我们很快找到某个规则

提供图形界面的独立程序，对jar进行检测，有报告生成，非常方便

很多功能插件没有实现，可独立使用FindBugs，但没法同时修改源码

**PMD** 检查源文件

检查Java源文件中的潜在问题 主要包括：

空try/catch/finally/switch语句块、未使用的局部变量、参数和private方法、空if/while语句、过于复杂的表达式，如不必要的if语句等。

过于严格，插件可以配置规则，有独立显示问题的视图，也很方便，一般来说，需要自定义规则才通过检验。

**CheckStyle** 检查源文件

主要关注格式，检查Java源文件是否与代码规范相符 主要包括：

Javadoc注释、命名规范、多余没用的Imports、Size度量，如过长的方法、缺少必要的空格Whitespace、重复代码

过于严格、按照Sun的规范太严格了，需要自定义规则，插件自定义规则没有查找功能，查找规则麻烦，只能做检查，不能修改代码。

* 从配置/使用的简便性、代码规范的可扩展性/可自定义性等角度对FindBugs、PMD、CheckStyle进行对比；

**FindBugs**

配置无查找功能，提供图形界面的独立程序，对jar进行检测，有报告生成，非常方便，很多功能插件没有实现。

**PMD**

插件可以配置规则，有独立显示问题的视图，很方便，可以自定义规则

**CheckStyle**

一般需要自定义规则，插件自定义规则没有查找功能，查找规则麻烦，只能做检查，不能修改代码。

* VisualVM如何帮助提升代码的运行时性能？

通过VisualVM可分析出函数的调用情况以及执行时间，包括各个线程的运行状态，是否有死锁、热锁等；还可以通过检测JVM 中加载的类和对象信息等帮助我们分析内存使用情况，从而分析处系统可能存在的问题。

* “代码是否符合编码规范”与“代码执行的时空复杂性”是否有直接的联系？为什么？

没有直接联系。编码规范是为了提高程序的可读性，有助于统一全局、促进团队协作，降低维护成本。代码不规范并不一定会改变代码执行的时空复杂性，但是往往不规范的代码会提高代码的时空复杂性，所以我们还是要尽可能提高代码的规范性。

* 对软件代码优化方面的其他体会。

感觉软件代码优化是一个没有止境的事情，在日常开发中，仅需要对经常出现的可优化的地方做出一些优化即可，在追求最优的时候应该注重一定的效率比，也就是说我们能够做到优化那些可以较大提高软件性能的问题就可以了。