

软件分析——任务单 app

1 检测过程

1.1 静态分析

1.1.1 工具

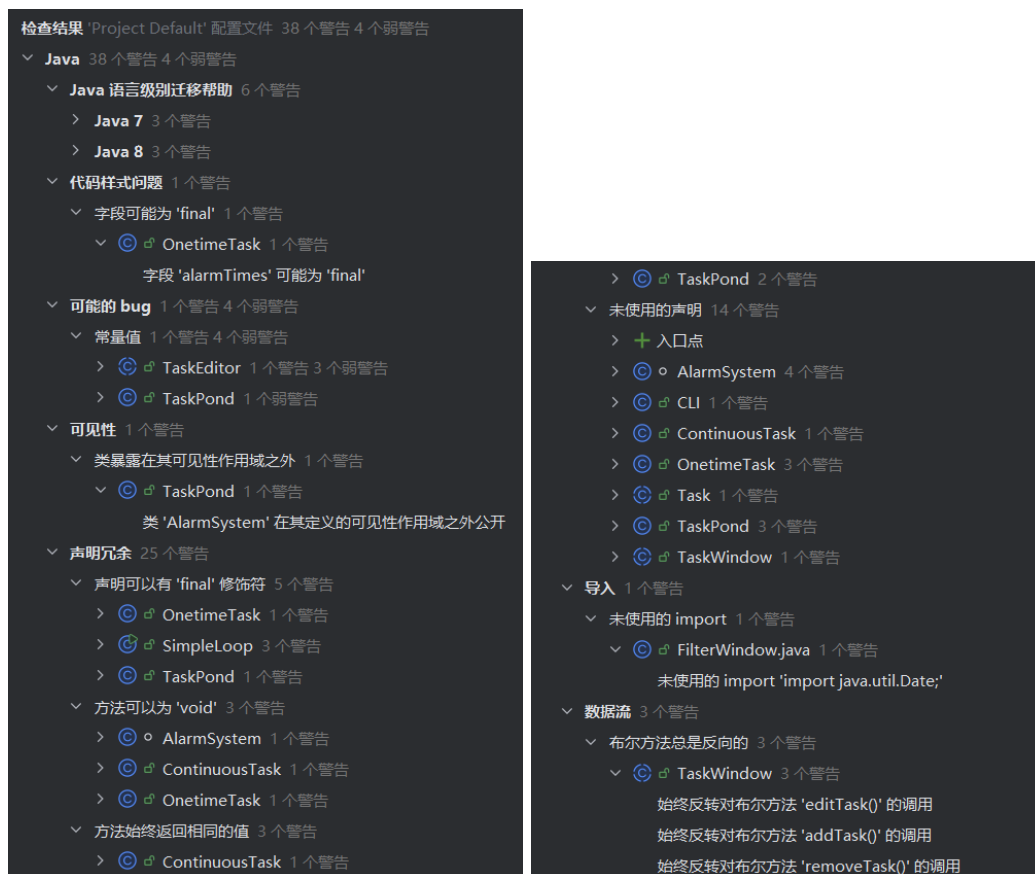
选用 IntelliJ Idea 自带的软件分析工具。

SonarQube 需要搭建服务器，消耗的内存量超出设备能力，舍弃。

1.1.2 配置

代码 > 检查代码 > 整个项目 > 默认配置文件

1.1.3 结果



1.2 大语言模型

1.2.1 工具

在 vscode 中使用 github copilot 对整个项目进行分析。

在 vscode 中，可以为 github copilot 快捷地添加上下文。这次 chat 中，将 codespace(该项目的源代码文件夹) 和实验一的需求报告 (tex 文件) 添加到 copilot 的上下文，来帮助她了解我的项目代码和需求。

1.2.2 prompt

kyhsdj: 请帮我分析这个Java项目：

项目概述

这是一个任务管理系统项目，包括单次任务管理、长期任务管理和DDL提醒功能。项目依赖Spring框架，数据库使用MySQL。

分析目标

1. 识别并报告代码中的潜在缺陷和漏洞。
2. 检查性能瓶颈，并提供优化建议。
3. 确保代码风格一致，符合最佳实践。

具体需求

1. 任务

1.1. 单次任务

- 需求ID: Req001
 - 需求描述: 单次任务只需要在DDL前完成
- 需求ID: Req002
 - 需求描述: 单次任务可以进行“开始”和“完成”操作
- 需求ID: Req003
 - 需求描述: 显示DDL、任务名称、状态，单次任务在任务时间线的界面左栏显示

1.2. 长期任务

- 需求ID: Req004
 - 需求描述: 长期任务比如每天背单词等，需按设定频率完成
- 需求ID: Req005
 - 需求描述: 长期任务可以进行“打卡”操作
- 需求ID: Req006
 - 需求描述: 长期任务在每个DDL前可提醒，设置开始和结束日期及每日提醒时间
- 需求ID: Req007
 - 需求描述: 打卡表示当日已完成，可以往后打卡并显示打的是哪天的
- 需求ID: Req008
 - 需求描述: 显示DDL、任务名称、状态，长期任务在任务时间线的界面右栏显示

2. DDL提醒

- 需求ID: Req009
 - 需求描述: 任务有系统通知和应用程序内消息提醒
- 需求ID: Req010
 - 需求描述: 消息栏显示未读消息数量
- 需求ID: Req011
 - 需求描述: 任务可以设置多个DDL提醒，如提前一天或几小时
- 需求ID: Req012
 - 需求描述: DDL日期显示日期和星期，并显示到DDL的剩余时间

3. 任务时间线界面

- 需求ID: Req013
 - 需求描述: 任务操作在任务时间线界面完成
- 需求ID: Req014
 - 需求描述: 点击任务显示详细信息，包括时间和任务备注
- 需求ID: Req015
 - 需求描述: 任务时间线类似日历，显示一个月的日期，点击某天显示该天所有任务和截止任务
- 需求ID: Req016
 - 需求描述: 显示按DDL排序的未完成任务和长期任务，点击任务查看详情
- 需求ID: Req017
 - 需求描述: 未完成任务包含代办任务和进行中任务
- 需求ID: Req018
 - 需求描述: 过期未完成任务弹出提醒，并在列表中显示状态为“过期”

1.3 结果

GitHub Copilot: ### 项目分析

1. 识别并报告代码中的潜在缺陷和漏洞

1. 潜在缺陷:

- TaskPond.java:
 - save() 和 load() 方法未实现, 可能导致数据持久化问题。
 - syncTask(Task task) 方法中未处理 alarmSystem.updateTask(task) 可能抛出的异常。
- ContinuousTask.java:
 - setEndDate(LocalDate endDate) 方法中未验证 endDate 是否早于 startDate, 可能导致逻辑错误。
- OnetimeTask.java:
 - addAlarmTime(LocalDateTime time) 方法中未处理 alarmTimes.add(i, time) 可能抛出的异常。
 - removeAlarmTime(LocalDateTime time) 方法中未处理 alarmTimes.remove(i) 可能抛出的异常。

2. 漏洞:

- CLI.java:
 - askForString 和 askForInt 方法中未对用户输入进行严格验证, 可能导致输入注入攻击。
- TaskFactory.java:
 - getNewTask() 方法中未对 taskType 进行严格验证, 可能导致不安全的任务类型创建。

2. 检查性能瓶颈, 并提供优化建议

1. 性能瓶颈:

- TaskPond.java:
 - filterByDate(LocalDate date) 和 filterByTag(String tag) 方法中使用了 O(n) 的复杂度遍历, 随着任务数量增加, 性能可能下降。
- ContinuousTask.java:
 - getAlarmTimes() 方法中使用了循环遍历, 可能导致性能问题。

2. 优化建议:

- TaskPond.java:
 - 使用索引或缓存机制优化 filterByDate 和 filterByTag 方法。
- ContinuousTask.java:
 - 优化 getAlarmTimes() 方法, 减少不必要的遍历。

3. 确保代码风格一致, 符合最佳实践

1. 代码风格问题:

- TaskPond.java:
 - 方法注释不完整, 缺少参数和返回值说明。
- ContinuousTask.java:
 - 缺少类注释和方法注释。
- CLI.java:
 - 缺少类注释和方法注释。

2. 最佳实践建议:

- TaskPond.java:
 - 补充方法注释, 说明参数和返回值。
- ContinuousTask.java:
 - 添加类注释和方法注释, 说明类的用途和方法的功能。
- CLI.java:
 - 添加类注释和方法注释, 说明类的用途和方法的功能。

具体需求分析

1. 任务

1.1. 单次任务

- Req001: 单次任务只需要在DDL前完成
 - 分析: OnetimeTask 类中 ddlTime 属性和 complete() 方法实现了该需求。
 - 建议: 确保 ddlTime 属性和 complete() 方法的逻辑正确。
- Req002: 单次任务可以进行“开始”和“完成”操作
 - 分析: OnetimeTask 类中 start() 和 complete() 方法实现了该需求。
 - 建议: 确保 start() 和 complete() 方法的逻辑正确。
- Req003: 显示DDL、任务名称、状态, 单次任务在任务时间线的界面左栏显示
 - 分析: OnetimeTask 类中 display() 方法实现了该需求。
 - 建议: 确保 display() 方法的逻辑正确。

1.2. 长期任务

- Req004: 长期任务比如每天背单词等，需按设定频率完成
 - 分析: `ContinuousTask` 类中 `startDate` 和 `endDate` 属性实现了该需求。
 - 建议: 确保 `startDate` 和 `endDate` 属性的逻辑正确。
- Req005: 长期任务可以进行“打卡”操作
 - 分析: `ContinuousTask` 类中 打卡 功能未明确实现。
 - 建议: 添加 打卡 功能，记录每日完成情况。
- Req006: 长期任务在每个DDL前可提醒，设置开始和结束日期及每日提醒时间
 - 分析: `ContinuousTask` 类中 `nextAlarmTime` 属性和 `getAlarmTimes()` 方法实现了该需求。
 - 建议: 确保 `nextAlarmTime` 属性和 `getAlarmTimes()` 方法的逻辑正确。
- Req007: 打卡表示当日已完成，可以往后打卡并显示打的是哪天的
 - 分析: `ContinuousTask` 类中 打卡 功能未明确实现。
 - 建议: 添加 打卡 功能，记录每日完成情况，并显示打卡日期。
- Req008: 显示DDL、任务名称、状态，长期任务在任务时间线的界面右栏显示
 - 分析: `ContinuousTask` 类中 `display()` 方法实现了该需求。
 - 建议: 确保 `display()` 方法的逻辑正确。

2. DDL提醒

- Req009: 任务有系统通知和应用程序内消息提醒
 - 分析: `TaskPond` 类中 `alarmSystem` 属性实现了该需求。
 - 建议: 确保 `alarmSystem` 属性的逻辑正确。
- Req010: 消息栏显示未读消息数量
 - 分析: `TaskPond` 类中 `alarmSystem` 属性实现了该需求。
 - 建议: 确保 `alarmSystem` 属性的逻辑正确。
- Req011: 任务可以设置多个DDL提醒，如提前一天或几小时
 - 分析: `OnetimeTask` 类中 `addAlarmTime(Duration duration)` 方法实现了该需求。
 - 建议: 确保 `addAlarmTime(Duration duration)` 方法的逻辑正确。
- Req012: DDL日期显示日期和星期，并显示到DDL的剩余时间
 - 分析: `OnetimeTask` 类中 `display()` 方法实现了该需求。
 - 建议: 确保 `display()` 方法的逻辑正确。

3. 任务时间线界面

- Req013: 任务操作在任务时间线界面完成
 - 分析: `TaskWindow` 类中 `showTasks()` 方法实现了该需求。
 - 建议: 确保 `showTasks()` 方法的逻辑正确。
- Req014: 点击任务显示详细信息，包括时间和任务备注
 - 分析: `TaskWindow` 类中 `showTasks()` 方法实现了该需求。
 - 建议: 确保 `showTasks()` 方法的逻辑正确。
- Req015: 任务时间线类似日历，显示一个月的日期，点击某天显示该天所有任务和截止任务
 - 分析: `TaskWindow` 类中 `showTasks()` 方法实现了该需求。
 - 建议: 确保 `showTasks()` 方法的逻辑正确。
- Req016: 显示按DDL排序的未完成任务和长期任务，点击任务查看详情
 - 分析: `TaskWindow` 类中 `showTasks()` 方法实现了该需求。
 - 建议: 确保 `showTasks()` 方法的逻辑正确。
- Req017: 未完成任务包含代办任务和进行中任务
 - 分析: `TaskWindow` 类中 `showTasks()` 方法实现了该需求。
 - 建议: 确保 `showTasks()` 方法的逻辑正确。
- Req018: 过期未完成任务弹出提醒，并在列表中显示状态为“过期”
 - 分析: `TaskWindow` 类中 `showTasks()` 方法实现了该需求。
 - 建议: 确保 `showTasks()` 方法的逻辑正确。

总结

通过以上分析，项目中存在一些潜在缺陷和漏洞，需要进行修复。同时，部分方法的性能可以通过优化算法或使用缓存机制来提升。此外，代码风格需要统一，添加必要的注释以提高代码可读性和维护性。

1.4 形式化工具

1.4.1 工具

cbmc-6.4.1 中的 jbmcc 功能

1.4.2 配置

1. 从官网下载 cbmc，安装后将 bin 文件夹加入环境变量

2. 定位到项目的输出文件夹

(IntelliJ IDEA 默认的输出文件夹: <path-to-task-list>/task-list/out/production/task-list)

3. 对含有 main 函数的类进行分析 (SimpleLoop)

```
jbmc --classpath . com.github.kyhdsjq.tasklist.ui.SimpleLoop --unwind 1
```

4. 通过 python 程序筛选结果中 ERROR 的部分 (具体见 ESBMC-results)，以下是部分过滤结果的代码:

```
def filter_paragraphs(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    paragraphs = []
    paragraph = []

    for line in lines:
        if line.strip() == '':
            if paragraph:
                paragraphs.append(paragraph)
                paragraph = []
            else:
                paragraph.append(line)

    if paragraph:
        paragraphs.append(paragraph)

    filtered_paragraphs = [p for p in paragraphs if any("ERROR" in line
        for line in p)]

    with open(file_path, 'w', encoding='utf-8') as file:
        for paragraph in filtered_paragraphs:
            for line in paragraph:
                file.write(line)
            file.write('\n')
```

jbmc 只针对.class 文件进行分析，无需改动原有的项目结构。

和 java 命令中的-cp 相同，jbmc 可以通过--classpath 将要分析的类路径手动添加。

因为命令行交互中涉及大量 while 循环，因此对整个程序形式化分析时将分析的循环次数设置为 1(-unwind 1)，以防分析时间过长。

1.4.3 结果

jbmc 采用 MAY-ANALYSIS，即分析出所有可能出问题的地方。因此它会出现很多误报的情况。

Result

运行结果在筛选后得到 250 行的错误报告，挑选其中两个函数展示：

```
com/github/kyhsdjg/tasklist/data/task/OnetimeTask.java function java::
    com.github.kyhsdjg.tasklist.data.task.OnetimeTask.getAlarmTimes:()
    Ljava/util/List;
[java::com.github.kyhsdjg.tasklist.data.task.OnetimeTask.getAlarmTimes
:()Ljava/util/List;.null-pointer-exception.1] line 45 Null pointer
check: ERROR
[java::com.github.kyhsdjg.tasklist.data.task.OnetimeTask.getAlarmTimes
:()Ljava/util/List;.null-pointer-exception.2] line 46 Null pointer
check: ERROR
[java::com.github.kyhsdjg.tasklist.data.task.OnetimeTask.getAlarmTimes
:()Ljava/util/List;.null-pointer-exception.5] line 49 Null pointer
check: ERROR
[java::com.github.kyhsdjg.tasklist.data.task.OnetimeTask.getAlarmTimes
:()Ljava/util/List;.null-pointer-exception.7] line 56 Null pointer
check: ERROR

com/github/kyhsdjg/tasklist/ui/window/TaskWindow.java function java::
    com.github.kyhsdjg.tasklist.ui.window.TaskWindow.editTask:()Z
[java::com.github.kyhsdjg.tasklist.ui.window.TaskWindow.editTask:()Z.
unwind.0] line 81 unwinding assertion loop 0: ERROR
[java::com.github.kyhsdjg.tasklist.ui.window.TaskWindow.editTask:()Z.
bad-dynamic-cast.1] line 84 Dynamic cast check: ERROR
[java::com.github.kyhsdjg.tasklist.ui.window.TaskWindow.editTask:()Z.
null-pointer-exception.10] line 84 Null pointer check: ERROR
```

2 工具评估

2.1 分析缺陷

以下都是分析后的结论，具体分析项请参考 `flaw_report/<tool>/<ture/flase>__positive.json`。

2.1.1 静态分析

静态分析所得的都是警告。

其中大部分都是只针对语法严谨性，修改后可能反而影响可读性：

- 字段可能为'final'
- 声明可以有'final' 修饰符
- 方法始终返回相同的值
- 始终反转对布尔方法的调用

但是也有重要的结构性问题

- 类暴露在可见性之外

这是因为我在中间进行了一次重构，使 AlarmSystem 可见性变为包可见，而忘记修改和它相关类的对外接口。

2.1.2 大语言模型

大语言模型主要关注宏观角度的安全性，例如：

- 开始时间一定比结束时间早
- 异常处理
- 输入注入攻击

其中大部分是正确的，也有时候过度关注语句而忽略了上下文。

例如这段代码：

```
int i = 0;
for (; i < alarmTimes.size(); i++) {
    if (alarmTimes.get(i).isEqual(time)) {
        alarmTimes.remove(i);
        if (taskPond != null)
            taskPond.sycTask(this);
        return true;
    }
    else if (alarmTimes.get(i).isAfter(time)) {
        return false;
    }
}
```

显然，i 作为下标在 alarmTimes 范围内，copilot 还提醒我对 alarmTimes.remove(i); 需要进行错误处理，显得多此一举。（不排除有些工作室的规范中要求用下标 remove 强制要写错误处理）

2.1.3 形式化工具

形式化工具通过离散数学进行分析，基本全部正确。

形式化工具分析的很多问题来自于外部库，我没有预料到外部库返回 null 结果，结果通过一系列连锁反应导致大量的空指针引用。

2.2 植入缺陷

在新分支 bug-injection 上插入缺陷。

见 <https://github.com/kyhsdjg/task-list/tree/bug-injection>

2.2.1 Memory Leak

在 Task Pond 中插入缺陷代码：

静态属性

```
private static final int SIZE = 1024 * 1024;
private static final ThreadLocal<byte[]> threadLocal = ThreadLocal.
    withInitial(() -> new byte[SIZE]);
```

常用函数

```
public boolean addTask(Task task) {
    threadLocal.set(new byte[SIZE]); // injecting memory leak
    boolean result = tasks.add(task);
    alarmSystem.addTask(task);
    task.setTaskPond(this);
    return result;
}
```

threadLocal 会在整个线程运行期间持有所有 new byte[size] 的引用，最终导致内存泄漏。

2.2.2 NULL Pointer Dereference

删除 OneTimeTask 中列表初始化的代码，使得之后对其操作都是 null pointer dereference:

构造函数

```
public OnetimeTask() state = TaskState.TODO; ddlTime = LocalDateTime.now(); // injecting
null pointer dereference // alarmTimes = new LinkedList<>();
```

2.2.3 Resource Leak

在 main 函数中打开文件后不正常关闭，造成资源泄漏：

main

```
public static void main(String[] args) // injecting resource leak String filePath =
"D:/reminder.ps1"; BufferedReader reader = null; try reader = new BufferedReader(new Fil-
eReader(filePath)); String line; while ((line = reader.readLine()) != null) System.out.println(line);
catch (IOException e) e.printStackTrace();
SimpleLoop simpleTaskList = new SimpleLoop(); simpleTaskList.startLoop();
```


2.3 识别植入缺陷

2.3.1 静态分析

- Memory Leak: 未检出
- NULL Pointer Dereference: 检出



- Resource Leak: 未检出

2.3.2 大语言模型

新的回答如下:



- Memory Leak: 检出
- NULL Pointer Dereference: 未检出
- Resource Leak: 检出

2.3.3 形式化工具

jbmc 主要用来检测运算上的错误和漏洞, 不能检测内存泄漏和资源泄漏的问题。

- Memory Leak: 未检出
- NULL Pointer Dereference: 检出

Result

```
comgithubkyhsdjqtasklistdatataskOnetimeTask.java function javacom.  
    github.kyhsdjq.tasklist.data.task.OnetimeTask.getAlarmTimes()  
        LjavautillList;  
[javacom.github.kyhsdjq.tasklist.data.task.OnetimeTask.  
    getAlarmTimes()LjavautillList;.null-pointer-exception.3] line 47  
    Null pointer check ERROR  
[javacom.github.kyhsdjq.tasklist.data.task.OnetimeTask.  
    getAlarmTimes()LjavautillList;.null-pointer-exception.6] line 50  
    Null pointer check ERROR
```

- Resouce Leak: 未检出

2.4 评估误报率

见工具比较-性能

3 工具比较

3.1 速度

由快到慢排序:

- 即时静态分析: IDE 集成的即时静态分析可以实时给出警告, 帮助程序员在构建过程中就发现错误。
- 大语言模型: vscode 集成的 github copilot 可以快捷添加上下文 (项目代码), 根据简单的 prompt 为程序员提供建议。适合在项目小模块完成后进行核验。
- 形式化工具和基于服务器的静态分析: 需要十分钟到几小时的分析时间, 适合在项目基本完成后整体分析或是有规划地增量分析。

3.2 性能

- 即时静态分析: 分析深度较浅, 可以满足基础的语法规约并对代码质量约束。
类型: 擅长单个类中语法上的错误。
漏报: 无法分析涉及多个函数的错误, 存在漏报。
错报: 会对于可以优化的语法块和可能出现错误的写法警告, 错报较多。
- 大语言模型: 不但能找出错误, 也能提供相应的解决方法。
类型: 更擅长结构性的错误。
漏报: 更加注意相近的上下文, 存在漏报。
错报: 可能产生幻觉, 还需要人工核验。但同时 copilot 也会提供解决方法, 这种 “chain of thought” 方式能减少错报。
- 形式化工具: 通过离散数学方式证明。
类型: 更擅长数值计算相关的错误:

- 空指针异常
- 数组越界
- 除以零错误
- 类型转换错误
- 算术溢出
- 断言失败
- 死锁

漏报：通过离散数学方法证明，漏报很少。

错报：可能因为过于保守存在错报。

3.3 工具推荐

按工程中不同阶段推荐：

- 每时每刻：使用实时静态分析，它擅长提供语法参考
- 完成小模块：询问大语言模型，它擅长在修正代码的同时给出对小模块的整体建议，帮助用户理清整个项目的架构
- 实现基本功能：形式化工具和基于服务器的静态分析，它擅长提供更全面严谨的分析