小米便签质量分析报告

黄泽楷，秦薪淇

2024 年 12 月 30 日

目录

1　小米便签开源软件概述

MiCode便签是小米便签的社区开源版，使用Java语言编写，基于Android操作系统进行开发运行和维护。由MIUI团队发起并贡献第一批代码，遵循NOTICE文件所描述的开源协议。这个应用从Android开发者角度来看，虽然简单，但五脏俱全，涉及到Android应用编程的方方面面，非常适合作为初学者的参考。本报告将结合对小米便签项目源码的阅读和注实践，对小米便签代码质量进行多角度的分析和解读，从而加深对Java编程规范的理解，掌握软件工程开发的基本框架和思路。该应用的主要功能是帮助用户管理他们的笔记和文件夹。

应用提供了创建、编辑、删除、分享、导出、同步等功能，同时还提供了设置闹钟提醒、发送到桌面、设置背景颜色和字体大小等便利功能。编程语言为Java，使用Android Studio进行开发，遵循MIT开源协议。本项目代码行数为13000多行，其中包含6个包，39个类，在前期的标注代码的过程中，标注了2350行代码。

2　代码质量分析方法

本报告主要介绍通过使用 CodeArts Check检查项目代码和人工审查分析两种方法，对小米便签源码进行代码质量分析。通过对代码框架的举例，了解其设计思路，并结合代码实例说明一些 Java 编程的基本规范，学习高质量的编程方法和高水平的编程技能。其中，会对阅读过程中发现的一些编程规范、功能结构上的不足进行分析，提出自己的改进建议。

3　代码质量分析工具

3.1　工具简介

CodeArts 是华为云提供的一站式云端DevOps平台，提供了一整套的开发工具和服务，包括代码生成器、自动化测试、持续集成、部署和监控等功能，使开发者能够快速开发、测试和发布应用程序。

CodeArts平台中的CodeArts Check工具在此基础上可以进行项目代码的检查，并生成代码审查报告，帮助我们更好的分析小米便签的代码。

本报告在华为云的CodeArts软件开发生产线上创建项目并使用CodeArts Check工具进行代码的审查。

3.2 使用CodeArts

1. 打开https://www.huaweicloud.com/devcloud/ 网站，在注册和登录后购买软件开发生产线 CodeArts 服务（实际上是免费的）。（见图3.1）

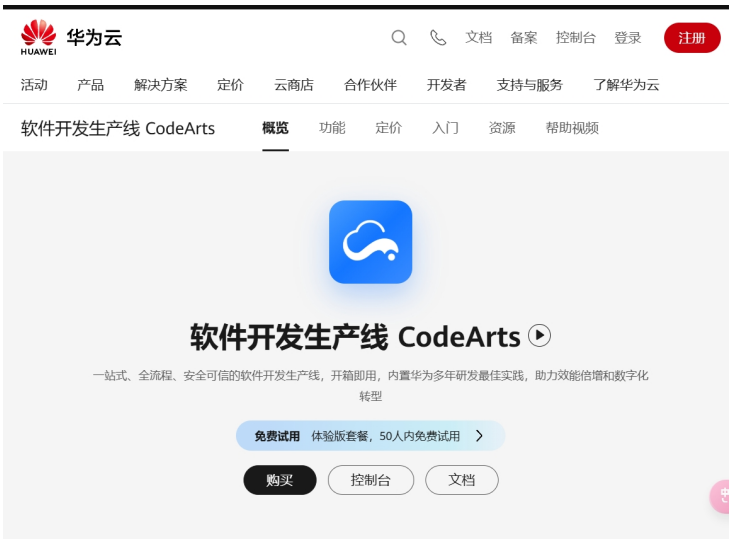
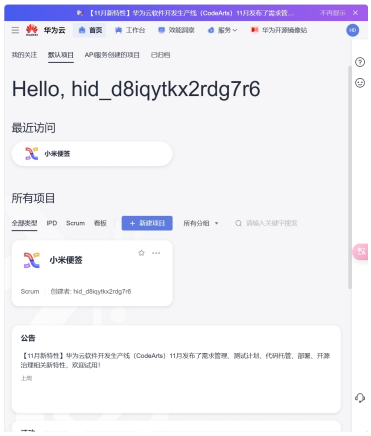


图 3.1: CodeArts开始界面

2. 然后打开工作台,点击创建项目，输入项目名称，选择项目类型为Java项目，点击创建。（见图3.2）

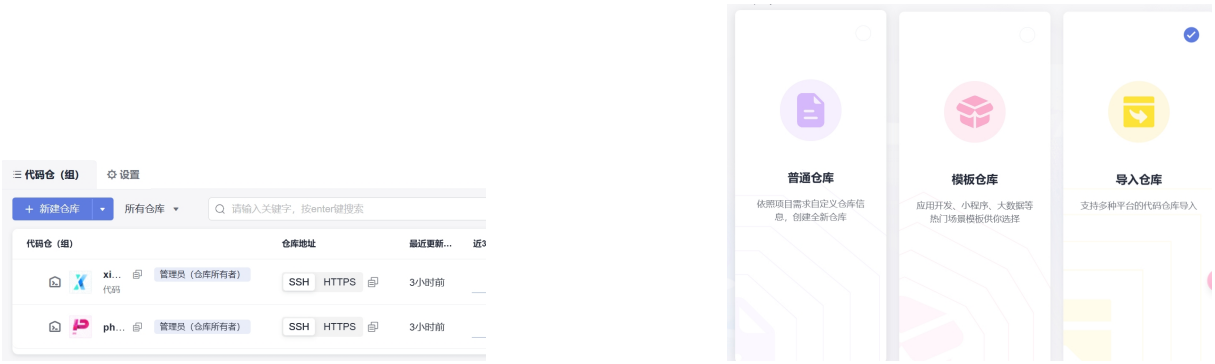


(a) 工作台界面

(b) 选择模版

图 3.2: 项目创建过程

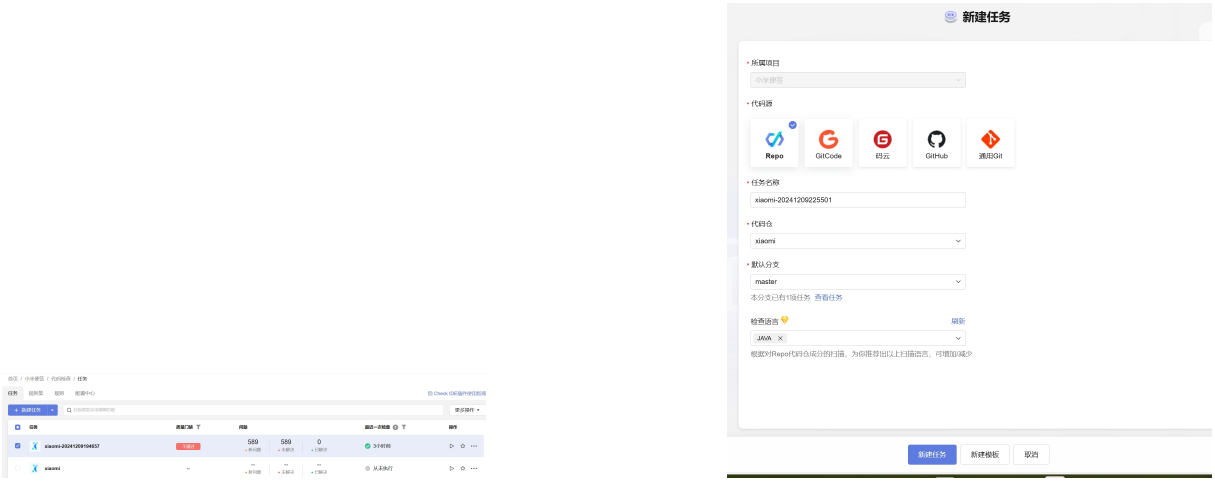
3. 创建项目后，导入自己仓库的小米便签的源代码。（见图3.3）



(a) 代码仓库　　(b) 导入代码界面

图 3.3: 代码导入过程

4. 进行代码审查并生成报告(见图3.4)



(a) 代码审查步骤1　　(b) 代码审查步骤2

图 3.4: 代码审查过程

3.3 使用说明以及分析结果

3.3.1 CodeArts代码分析缺陷等级

CodeArts代码分析工具提供了四个级别的代码分析报告，分别为：

- 致命：这类缺陷是代码中最为严重的错误，它们会导致程序崩溃或无法运行。致命缺陷通常涉及到内存访问错误、空指针引用、数组越界等问题。在软件开发中，这类缺陷需要被优先解决，以确保软件的基本功能能够正常运行。

- 严重：严重缺陷虽然不如致命缺陷那样会导致程序立即崩溃，但它们仍然会对软件的稳定性和性能产生重大影响。这类缺陷可能包括资源泄露、逻辑错误、安全漏洞等。

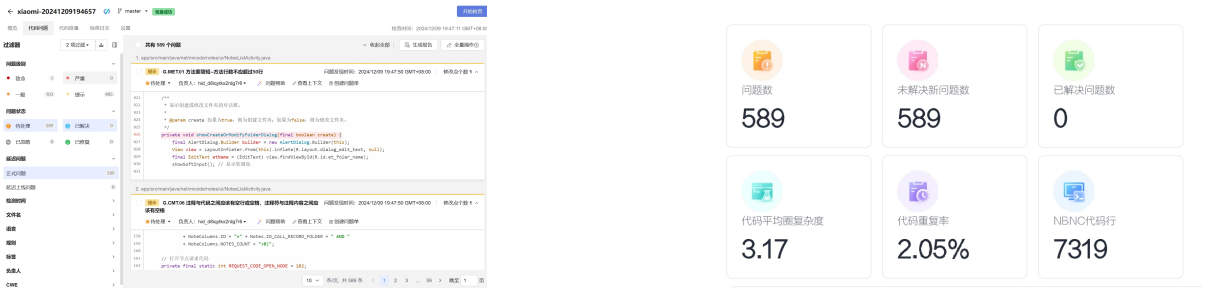
- 一般：一般缺陷是常见的编程问题，它们可能不会直接导致程序失败，会影响代码的可读性、可维护性或性能。这类缺陷可能包括命名不一致、代码重复、未使用的变量、复杂的条件语句等。

- 提示：提示是CodeArts给出的建议，它们通常不是错误，而是建议开发者考虑的改进点。这些提示可能涉及到代码风格、性能优化、可读性提升等方面。提示的数量最多，表明有很多潜在的改进空间，开发者可以根据这些提示来优化代码。

3.3.2 CodeArts代码分析结果（见图3.5、3.6）

CodeArts Check工具通过审查小米便签代码，生成了一篇代码分析报告，报告结果见图3.5。

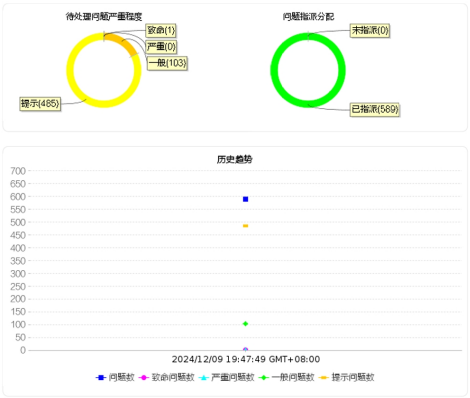
从分析的整体报告中可以看到，代码存在589个问题，其中一个致命问题，103个一般问题，485个提示问题。代码平均圈复杂度为3.17，代码重复率为2.05%，NBNC代码行为7319。



(a) 代码分析总览　　(b) 问题分布统计

图 3.5: CodeArts代码分析结果总览1

(a) 问题类型分布



(b) 代码质量指标

缺陷分类	总计
G.OBJ.07 子类覆写父类方法或实现接口时必须加上@Override注解	13
G.ERR.03 不要直接捕获可通过预检查进行处理的RuntimeException，如NullPointerException、IndexOutOfBoundsException等	6
G.CTL.01 不要在控制性条件表达式中执行赋值操作或执行复杂的条件判断	3
G.FMT.08 使用空格进行缩进，每次缩进4个空格	5
G.OTH.01 安全场景下必须使用密码学意义上的安全随机数	1
G.NAM.04 方法名应采用小驼峰命名	2
G.CMT.06 注释与代码之间应该有空行或空格，注释符与注释内容之间应该有空格	46
G.CMT.01 public或protected修饰的元素应添加Javadoc注释	260
G.TYP.13 在引用类型向下转换前用instanceof进行判断	37
G.NAM.03 类、枚举和接口名应采用大驼峰命名	2
G.FMT.07 应该避免空块，必须使用空块时，应采用统一的大括号换行风格	1
G.OTH.03 不用的代码段包括import，直接删除，不要注释掉--不用的import语句，直接删除，不要注释掉	3
G.MET.01 方法要简短--方法的代码块嵌套深度不应超过4层	14
G.MET.01 方法要简短--方法行数不应超过50行	11
G.MET.01 方法要简短--方法的参数不应超过5个	1
G.NAM.05 常量名采用全大写单词，单词间以下划线分隔	1
G.CON.12 避免不加控制地创建新线程，应该使用线程池来管控资源	3
G.FMT.13 用空格突出关键字和重要信息	1

(c) 代码复杂度分析

G.CMT.02 顶层public类的Javadoc应该包含功能说明和创建日期/版本信息	40
G.FMT.05 在条件语句和循环块中应该使用大括号	22
G.EXP.04 表达式的比较，应该遵循左侧倾向于变化、右侧倾向于不变的原则--使用equals方法进行字符串比较	2
G.FMT.12 减少不必要的空行，保持代码紧凑	38
G.OBJ.10 接口定义中去掉多余的修饰词	27
G.FMT.04 一个类或接口的声明部分应该按照类变量、静态初始化块、实例变量、实例初始化块、构造器、方法的顺序出现，且用空行分隔	32
G.CMT.03 方法的Javadoc中应该包含功能说明，根据实际需要按顺序使用@param、@return、@throws标签对参数、返回值、异常进行注释	3
G.EXP.04 表达式的比较，应该遵循左侧倾向于变化、右侧倾向于不变的原则--表达式比较左变右不变	4
G.OBJ.08 正确实现单例模式	1
G.FMT.20 数字字面量应该设置合适的后缀，long类型应该使用L作为后缀	3
G.DCL.03 禁止C风格的数组声明	5
G.FMT.06 对于非空块状结构，左大括号应该放在行尾，右大括号应该另起一行	2
总计	589

(d) 代码重复度分析

图 3.6: CodeArts代码分析结果总览2

4　人工分析质量报告

4.1　代码结构

Notes 项目共包括 6 个程序包，39 个类，一共 7393 行有效代码。

4.1.1　用例图（见图4.7）

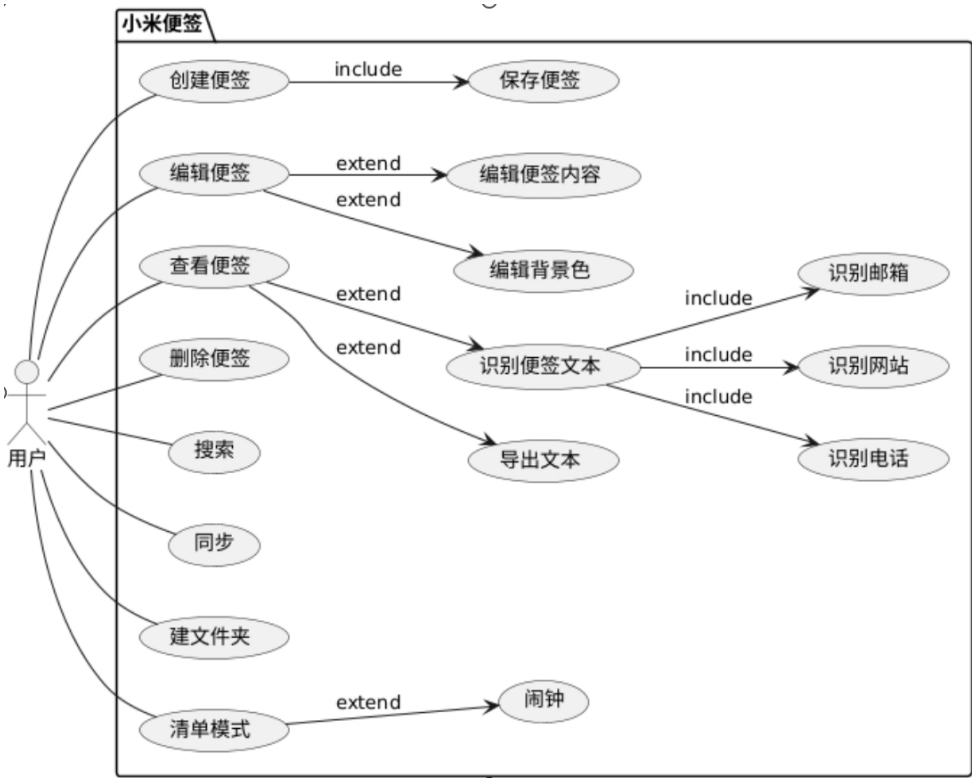


图 4.7: 小米便签用例图

4.1.2　软件功能情况

- 功能1：新建/删除/移动便签

  新建便签：通过小米便签软件的主界面下方的”写便签”快捷键和文件夹、便签视图下的选项”新建便签”可以在当前目录创建一个便签并打开进入文本编辑。其中主界面下的快速创建方式可以迅速创建一个待编辑的便签，用于满足临时迅速记的需求。

  删除便签：通过长按便签并选择删除选项可以删除已创建的便签，用于对便签集进行有效的管理。

移动便签：通过长按便签并选择移动选项可以将已创建的便签移动到某个文件夹中，用于在快速创建便签后对便签集进行有效的管理。

- 功能2：新建文件夹

  在主界面的选项中可以选择”新建文件夹”，用于分类管理便签。

- 功能3：导出文本

  在主界面的选项中可以选择”导出文件”，在Android手机提供SD卡支持的情况下，将小米便签中的便签内容逐个转化为.txt的文本文档。

- 功能4：同步

  在主界面的选项中可以选择”同步”，与Google Task中的备忘录事项，将本地的事项上传到服务器，或将Google服务器上的表单下载到本地。

- 功能5：设置

  在主界面打开菜单进入设置界面，一是同步账号，与google task同步便签记录，二是将便签背景颜色设置为随机

- 功能6：检索便签

  在主界面的选项中可以选择 ”搜索”，通过关键词查找到包含该关键词的便签，显示在界面上。

- 功能7：修改字体大小

  在便签编辑的界面，在选项中选择”字体大小”，可以将当前便签的所有字体进行放大和缩小，其中包括4中字大小：小、正常、大、超大。

- 功能8：进入清单模式

  在便签编辑的页面，可以选择进入清单模式的选项。选择后，在便签的每一行（段内部的自动换行除外）行首出现一个勾选框，用于当前便签下标记某事项的完成情况。如果该事项已完成，则用户在勾选框中轻触，此时勾选框中将出现一个对勾，框后的陈述文字被添加了中央删除线。（外侧文件夹并不能显示事项的完成状态，可优化）

- 功能9：发送到桌面

在Android操作系统的桌面创建小米便签的小部件后，在编写便签完成后，使用选项”发送至桌面”，便可在便签小部件上显示当前便签的内容。

- 功能10：添加/删除提醒

  添加提醒在便签编辑界面可以选择”添加提醒”选项，然后弹出一个对话框用于选择提醒的时间（包括月、日、星期、时、分），之后会在便签上显示一个闹钟的图标，标志提醒时间，到了提醒时间时，操作系统便会弹出一个对话框显示便签的内容并响铃，闹钟图表标志变为”已过期”。

  删除提醒：在便签编辑界面可以选择”删除提醒”选项。

- 功能11：分享

  在便签编辑页面上可以选择”分享”选项，之后可以将便签内容分享给GTask、QQ、微信等应用程序，其过以纯文本格式进行。

- 功能12：识别便签文本

  识别邮箱：小米便签可识别文本中邮箱，点击可复制相应的邮箱

  识别网站：小米便签可识别文本中网站，点击可直接打开网站

  识别电话：小米便签可识别文本中电话，可直接复制或者拨打电话

4.1.3 模块和功能对应关系(见图4.8)

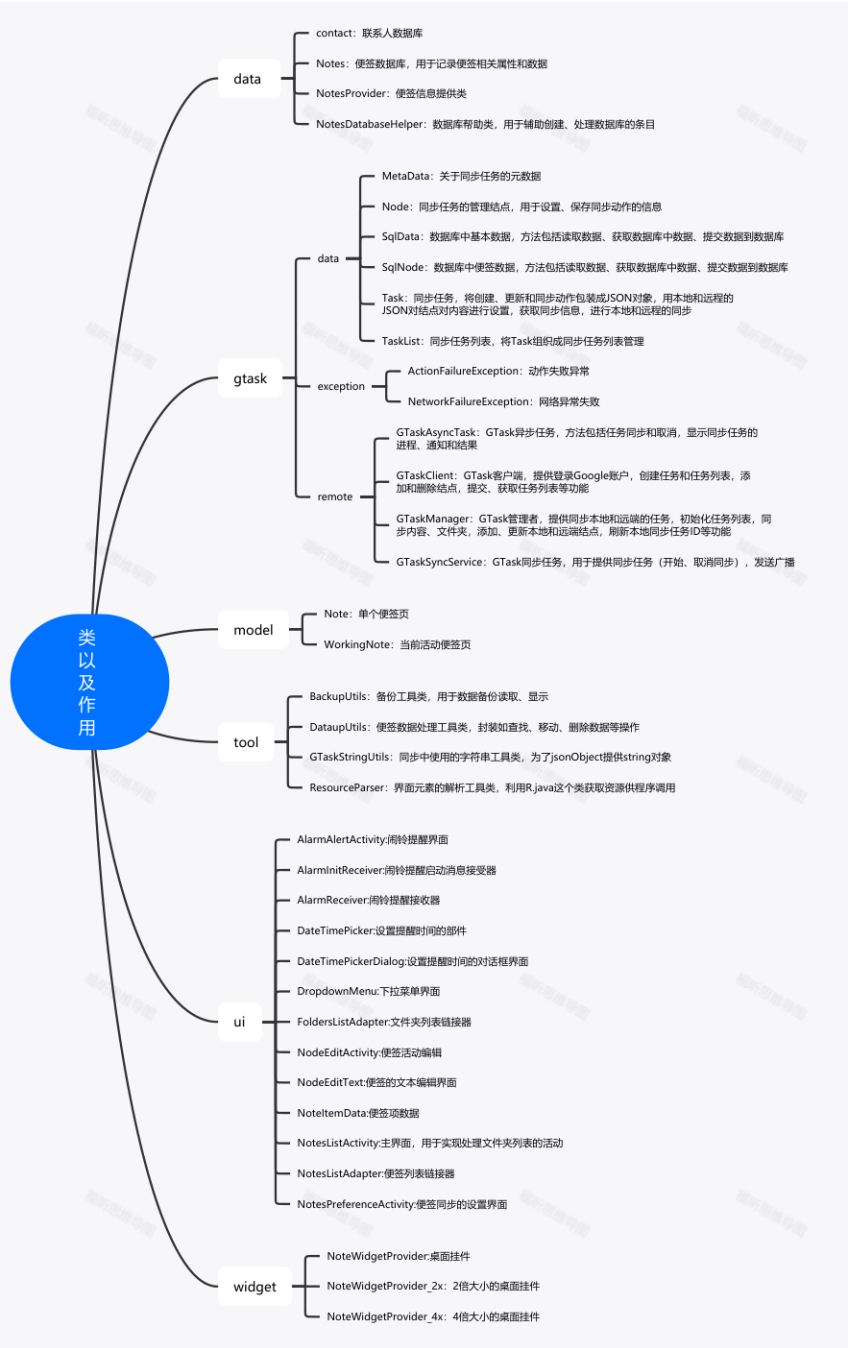


图 4.8: 模块和功能对应关系图

4.1.4 源代码体系结构

小米便签整体上是一个层次性的体系结构，由界面层、业务层、模型层和数据层 4 层构成，每层包含若干个程序包。相邻层次的程序包之间存在交互。小米便签的体系结构图见图4.9。

- 界面层：根据表格（在前置动作部分）中出的主要功能，我们可以很轻松地将ui和widget填入其中，而res包含了所有的xml文件和相对应图片，属于MVC架构里面View部分，所以也应该填入界面部分。
- 业务层：根据MVC架构的逻辑，个人判断它应该担任是C（Control控制器）的角色：处理用户输入的信息。负责从视图读取数据，控制用户输入，并向模型发送数据，是应用程序中处理用户交互的部分。负责管理与用户交互交互控制。
- 模型层：该层负责对小米便签的单个便签项进行建模，提供了便签项的基本操作功能和对应业务, 并与数据层进行交互，以支持便签的创建、访问和修改。该层主要通过 model 程序包中的 Note 类、WorkingNote 类等加以实现。
- 数据层：该层负责组织和存储小米便签的相关数据,提供数据访问、数据合法性检验、数据访问缺失异常处理等功能和服务。该层主要通过 data 和 gtask.data 程序包加以实现。

4.1.5 data包结构分析示例（见图4.10）

data包中的类构成了Android应用中处理联系人和笔记数据的核心模块。Contact类负责通过电话号码查联系人名称，并使用缓存来优化性能。Notes类定义了与笔记和文件夹相关的常量、数据列接口，并提供了系统文件夹的标识符和Intent额外数据的键。NotesDatabaseHelper是数据库辅助类，负责管理数据库的创建、版本管理和触发器的创建。NotesProvider作为内容提供者，处理对笔记数据的查询、插入、更新和删除操作，并与NotesDatabaseHelper交互以执行实际的数据库操作。这些类共同为应用提供了数据管理和访问的功能。

4.1.6 类的方法实现举例——NotesListItem（见图4.11）

NotesListItem类是一个继承自LinearLayout的自定义视图组件，用于在用户界面中显示笔记列表项。

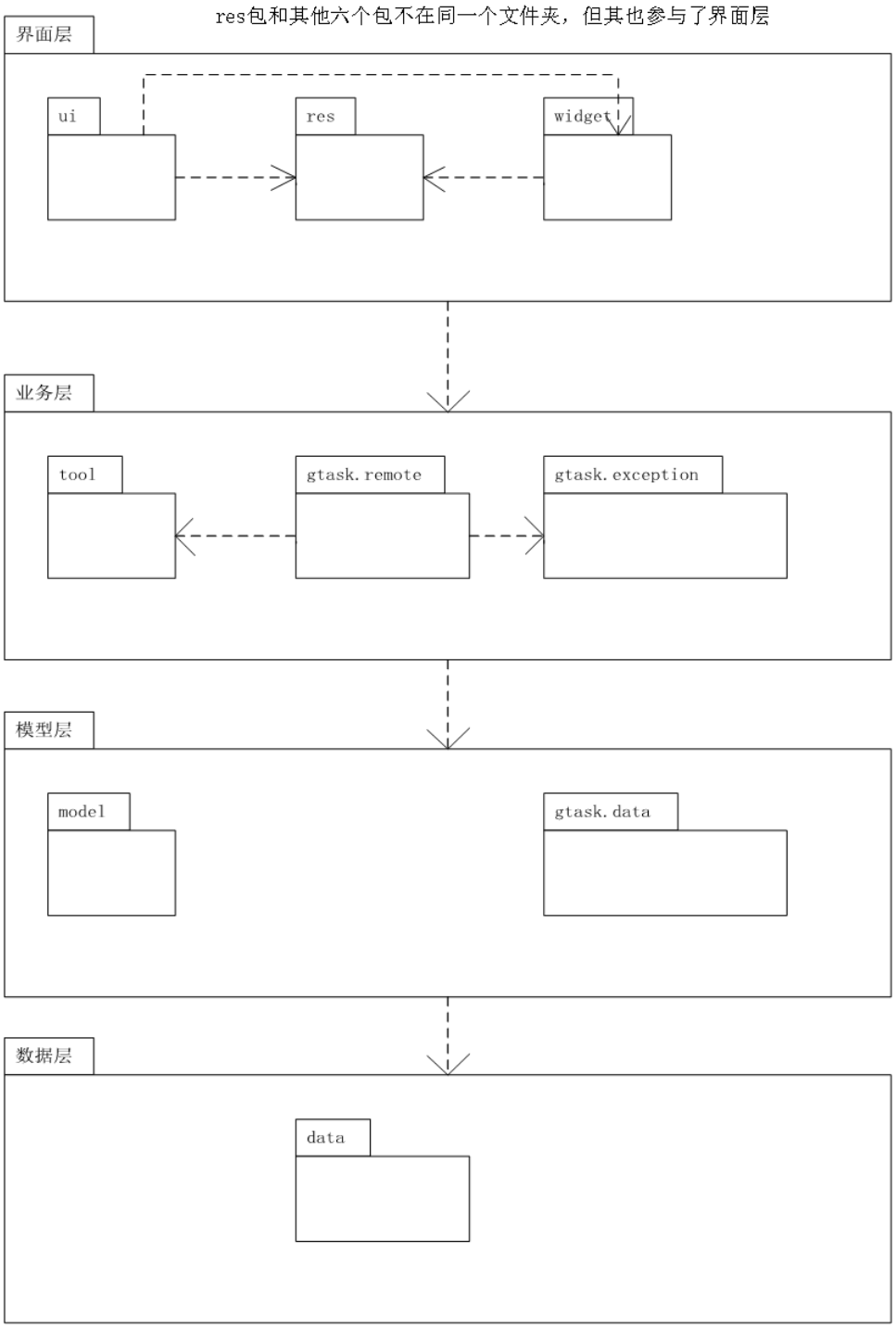


图 4.9: 小米便签体系结构图

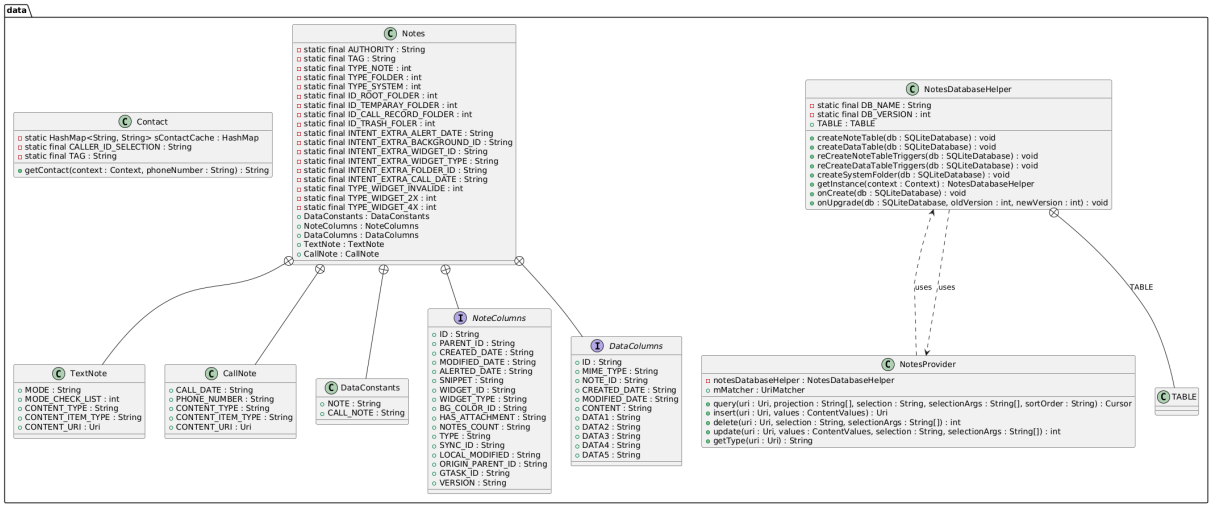


图 4.10: data包结构分析图

```java
public class NotesListItem extends LinearLayout {  XinqiQin
    private ImageView mAlert; // 用于显示提醒图标  10 usages
    private TextView mTitle; // 显示笔记标题  8 usages
    private TextView mTime; // 显示修改时间  2 usages
    private TextView mCallName; // 在通话记录笔记中显示通话名称  5 usages
    private NoteItemData mItemData; // 绑定的笔记数据  2 usages
    private CheckBox mCheckBox; // 选择框，用于多选模式  4 usages

    /*
     * 构造函数，初始化视图组件。
     */
    public NotesListItem(Context context) {  XinqiQin
        super(context);
        inflate(context, R.layout.note_item,  root: this);
        // 初始化视图组件
        mAlert = (ImageView) findViewById(R.id.iv_alert_icon);
        mTitle = (TextView) findViewById(R.id.tv_title);
        mTime = (TextView) findViewById(R.id.tv_time);
        mCallName = (TextView) findViewById(R.id.tv_name);
        mCheckBox = (CheckBox) findViewById(android.R.id.checkbox);
    }
```

图 4.11: NotesListItem类方法实现示例

从这些方法的设计来看，突出了高内聚低耦合的程序设计方法，在变量、常量的命名上也符合 Java 编程规范，体现了作者较高的软件设计水平。

4.2 高质量编码规范

- 规范的命名

```
private ImageView mAlert; // 用于显示提醒图标  10 usages
private TextView mTitle; // 显示笔记标题  8 usages
private TextView mTime; // 显示修改时间  2 usages
private TextView mCallName; // 在通话记录笔记中显示通话名称  5 usages
private NoteItemData mItemData; // 绑定的笔记数据  2 usages
private CheckBox mCheckBox; // 选择框，用于多选模式  4 usages
```

图 4.12: 规范命名示例

这段代码中的变量命名遵循了Java的驼峰命名法（camelCase），其中每个成变量都以小写字母开头，后续单词首字母大写，以提高可读性。变量名前缀m是成员变量（member variable）的常用缩写，有助于区分局部变量和成员变量。mAlert表示用于显示提醒图标的 ImageView，mTitle用于显示笔记标题的TextView，mTime用于显示修改时间的TextView，mCallName用于在通话记录笔记中显示通话名称的TextView，而mItemData则表示绑定到视图的笔记数据对象。最后一个成员mCheckBox，代表用于多选模式的选择框。整体而言，这些变量名简洁且具有描述性，使得代码易于理解，符合良好的编码规范。

- 空格和大括号的使用

```
public void bind(Context context, NoteItemData data, boolean choiceMode, boolean checked) {  1 usage  XinqiQin
    // 根据是否为选择模式和笔记类型，控制复选框的可见性和选中状态
    if (choiceMode && data.getType() == Notes.TYPE_NOTE) {
        mCheckBox.setVisibility(View.VISIBLE);
        mCheckBox.setChecked(checked);
    } else {
        mCheckBox.setVisibility(View.GONE);
    }
```

图 4.13: 空格和大括号使用规范示例

代码中的空格和大括号使用遵循了清晰的编码规范，增强了代码的可读性。在if语句中，条件表达式与后续代码之间使用了空格分隔，使得逻辑结构更加清晰。例如，在if-else结构中，else关键字后紧跟大括号，没有额外的空行或空格，

使得代码紧凑而整洁。此外，每个代码块内部的语句之间也适当地使用空格，如mCheckBox.setVisibility(View.VISIBLE)和mCheckBox.setChecked(checked)之间的空格，使得代码的执行流程一目了然。整体上，这种空格和大括号的使用方式有助于突出代码的结构，降低理解难度，符合常见的Java编码风格。

4.3 高质量代码设计

高质量的项目代码是指遵循良好编码规范、结构清晰、可读性强、易于维护和扩展的源代码。它具有以下特点：代码组织合理，文件和类结构分明，遵循单一职责原则；命名规范一致，变量和方法名称清、描述性强；代码格式统一，缩进和空格使用得当，行长度适宜；注释充分，方法和类有适当的文档注释，代码块有必要的解释避免代码重复，逻辑清晰，模块化程度高；错误处理得当，异常管理到位；代码风格一致，整个项目遵循统一的编码风格；性能优化，资源管理得当，无资源泄露；安全性考虑周全，输入验证严格，加密和安全措施得当；测试覆盖广泛，有单元测试验证代码正确性。总的来说，高质量的项目代码不仅关注功能的实现，还注重代码的可维护性、可读性和长期可持续性，使得项目易于理解和后续开发。

比如如下的代码，很好的体现了高质量代码设计，见图4.14、4.15、4.16。

```
// 请求无标题的窗口
requestWindowFeature(Window.FEATURE_NO_TITLE);

// 设置窗口在锁屏时也显示，并根据屏幕状态决定是否保持唤醒
final Window win = getWindow();
win.addFlags(WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED);
if (!isScreenOn()) {
    win.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON
            | WindowManager.LayoutParams.FLAG_TURN_SCREEN_ON
            | WindowManager.LayoutParams.FLAG_ALLOW_LOCK_WHILE_SCREEN_ON
            | WindowManager.LayoutParams.FLAG_LAYOUT_INSET_DECOR);
}
```

图 4.14: 高质量代码设计示例1

```
Intent intent = getIntent();
try {
    mNoteId = Long.valueOf(intent.getData().getPathSegments().get(1));
    mSnippet = DataUtils.getSnippetById(this.getContentResolver(), mNoteId);
    mSnippet = mSnippet.length() > SNIPPET_PREW_MAX_LEN ? mSnippet.substring(0,
            SNIPPET_PREW_MAX_LEN) + "..."
            : mSnippet;
} catch (IllegalArgumentException e) {
    e.printStackTrace();
    return;
}
```

图 4.15: 高质量代码设计示例2

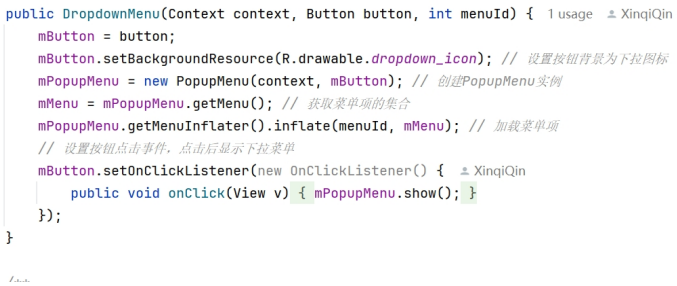


图 4.16: 高质量代码设计示例3

4.4 高质量编程的方法和高水平的编程技能

在编写代码之前，重要的事软件需求的分析，通过软件需求文档的撰写，分析软件的各项需求，画出用例图，交互图以及类图，分析各个模块，再去编程具体实现。

高质量编程，采用一系列经过验证的方法和高水平编程技能来构建可靠、可维护和高效软件。这包括深入理解编程语言的核心概念，遵循行业标准的编码规范，编写清晰、简洁且意图明确的代码；有效地使用版本控制系统进行代码管理；采用模块化和抽象化原则来提高代码复用性和降低复杂性；进行彻底的测试，包括单元测试、集成测试和系统测试，以确保代码质量和功能正确性；持续进行代码审查和重构，以识别并改进潜在的设计问题和性能瓶颈；以及运用设计模式和最佳实践来解决特定问题，从而提升软件的可扩展性和可维护性。

如何进行高质量编程？首先，深入理解编程语言和工具，遵循编码规范，编写可读性强、结构清晰的代码；其次，采用版本控制和代码审查来管理代码变更，确保代码质量和促进团队协作；再次，实施彻底的测试策略，包括单元测试和集成测试，以验证功能和提高代码的健壮性；此外，持续重构代码以优化设计，保持代码的简洁性和可维护性；同时，运用适当的算法和数据结构，以及设计模式来解决复杂问题，提高代码的可扩展性和性能。

4.5 存在的质量问题

- 缺少注释（见图4.17）

  小米便签源代码最大的问题就是缺少注释，导致刚接触这个项目进行阅读的时候会受到很大的阻碍。

  示例如图4.17（注释是我们自己加的）。

- 个别语句冗余（见图4.18）

```
private void showSelectAccountAlertDialog() {  1 usage  XinqiQin
    // 创建对话框构建器并设置自定义标题
    AlertDialog.Builder dialogBuilder = new AlertDialog.Builder( context: this);

    View titleView = LayoutInflater.from( context: this).inflate(R.layout.account_dialog_title,  root: null);
    TextView titleTextView = (TextView) titleView.findViewById(R.id.account_dialog_title);
    titleTextView.setText("Sync notes");
    TextView subtitleTextView = (TextView) titleView.findViewById(R.id.account_dialog_subtitle);
    subtitleTextView.setText("Please select a google account. Local notes will be syn...");

    dialogBuilder.setCustomTitle(titleView);
    dialogBuilder.setPositiveButton( text: null,  listener: null); // 移除默认的确定按钮
```

(a) 代码注释示例1

```
private void loadAccountPreference() {  1 usage  XinqiQin
    mAccountCategory.removeAll(); // 清空账户分类下的所有条目

    // 创建并配置账户偏好项
    Preference accountPref = new Preference( context: this);
    final String defaultAccount = getSyncAccountName( context: this); // 获取默认同步账户名称
    accountPref.setTitle("Sync account"); // 设置标题
    accountPref.setSummary("Sync notes with google task"); // 设置摘要
    accountPref.setOnPreferenceClickListener(new OnPreferenceClickListener() {  XinqiQin
        public boolean onPreferenceClick(Preference preference) {  no usages  XinqiQin

            return true;
        }
    });

    mAccountCategory.addPreference(accountPref); // 将账户偏好项添加到账户分类下
}
```

(b) 代码注释示例2

图 4.17: 代码注释示例

```
if (folderCursor != null) {
    if (folderCursor.moveToFirst()) {
        do {
            // Print folder's name
            String folderName = "";
            if(folderCursor.getLong(NOTE_COLUMN_ID) == Notes.ID_CALL_RECORI
                folderName = mContext.getString(R.string.call_record_folde
            } else {
                folderName = folderCursor.getString(NOTE_COLUMN_SNIPPET);
            }
            if (!TextUtils.isEmpty(folderName)) {
                ps.println(String.format(getFormat(FORMAT_FOLDER_NAME), fo
            }
            String folderId = folderCursor.getString(NOTE_COLUMN_ID);
            exportFolderToText(folderId, ps);
        } while (folderCursor.moveToNext());
    }
    folderCursor.close();
}
```

图 4.18: 语句冗余示例

- 日志写入不够具体准确

图4.19中，在删除发生异常时写入日志信息是项很好的编程规范，但是在写入异常描述时，应该尽可能的简明扼要且保证准确。

```
if (DataUtils.batchDeleteNotes(mContentResolver, mNotesListAdapter
        .getSelectedItemIds())) {
} else {
    Log.e(TAG,  msg: "Delete notes error, should not happens");//what error?
}
```

图 4.19: 原始日志写入示例

可修改为图4.20：

```
if (DataUtils.batchDeleteNotes(mContentResolver, mNotesListAdapter
        .getSelectedItemIds())) {
} else {
    Log.e(TAG,  msg: "Delete notes error in DataUtils.batchDeleteNotes, should not happens");
}
```

图 4.20: 改进后的日志写入示例

5 自动分析质量报告

通过在CodeArts上创建项目、导入小米便签源代码、代码审查，我们最终得到了CodeArts对于小米便签的自动分析质量报告。（分析结果见图5.21、5.22）

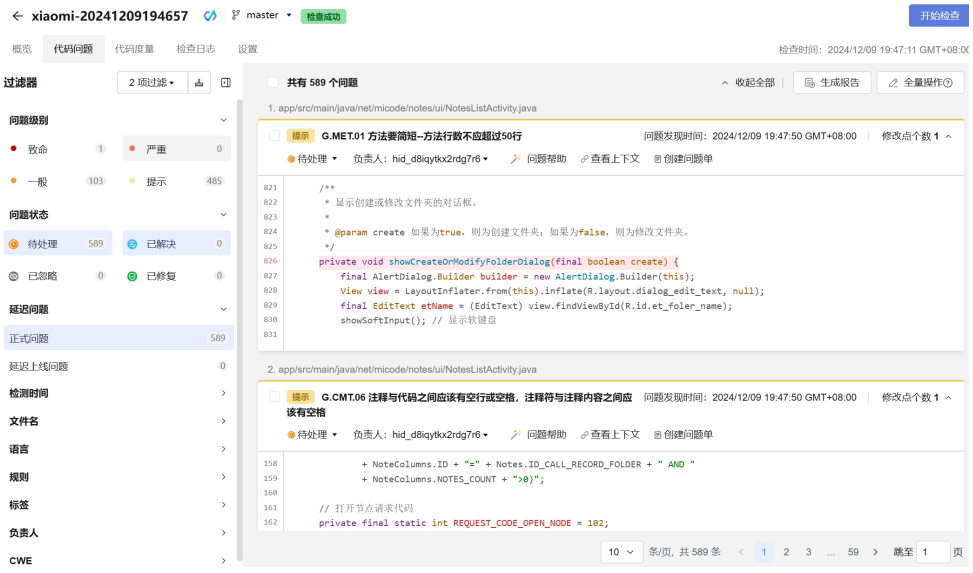


图 5.21: 代码分析总体概况

图 5.22: 问题分布统计图

从分析的整体报告中可以看到，代码存在589个问题，其中一个致命问题，103个一般问题，485个提示问题。代码平均圈复杂度为3.17，代码重复率为2.05%，NBNC代码行为7319。

5.1 致命问题

安全场景下必须使用密码学意义上的安全随机数。

不安全的随机数可能被部分或全部预测到，导致系统存在安全隐患，安全场景下使用的随机数必须是密码学意义上的安全随机数。密码学意义上的安全随机数分为两类：1.真随机数产生器产生的随机数； 2.以真随机数产生器产生的少量随机数作为种子的密码学安全的伪随机数产生器产生的大量随机数。

已知的可供产品使用的密码学安全的非物理真随机数产生器有：Linux操作系统的/dev/random设备接口存在阻塞问题）和Windows操作系统的CryptGenRandom接口。Java中的SecureRandom是一种密码学安全的伪随机数产生器，对于使用非真随机数产生器产生随机数时，要使用少量真随机数作为种子。常见安全场景包括但不限于以下场景： 1.用于密码算法用途，如生成IV、盐值、秘钥等； 2.会话标识（sessionId）的生成； 3.挑战算法中的随机数生成； 4.验证码的随机数生成。

(a) 不安全随机数使用示例

```
public byte[] generateSalt() {
    byte[] salt = new byte[8];
    try {
        SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
        random.setSeed(random.generateSeed(SEED_LEN));
        random.nextBytes(salt);
    } catch (NoSuchAlgorithmException ex) {
        // 处理异常
    }
    return salt;
}
```

(b) 安全随机数使用示例

图 5.23: 随机数使用对比示例

5.2 一般问题

一般问题在代码审查的时候存在103个，下面选取4个作为示例进行分析。

5.2.1 示例1(见图4.25)

不用的代码段包括import，直接删除，不要注释掉–不用的import语句，直接删除，不要注释掉。

5.2.2 示例2(见图4.26)

子类覆写父类方法或实现接口时必须加上@Override注解。

图 5.24: 未使用代码示例

图 5.25: Override注解缺失示例

5.2.3 示例3(见图4.27)

8. app/src/main/java/net/micode/notes/ui/NotesListActivity.java

一般 G.TYP.13 在引用类型向下转换前用instanceof进行判断　问题发现时间：2024/12/09 19:47:50 GMT+08:00　修改点个数 1

待处理　负责人：hid_d8iqytkx2rdg7r6　问题帮助　查看上下文　创建问题单

```
272         mContentResolver = this.getContentResolver();
273         // 创建后台查询处理器
274         mBackgroundQueryHandler = new BackgroundQueryHandler(this.getContentResolver());
275         mCurrentFolderId = Notes.ID_ROOT_FOLDER;
276         // 初始化ListView和相关监听器
277         mNotesListView = (ListView) findViewById(R.id.notes_list);
278         mNotesListView.addFooterView(LayoutInflater.from(this).inflate(R.layout.note_list_footer, null),
279                 null, false);
280         mNotesListView.setOnItemClickListener(new OnListItemClickListener());
281         mNotesListView.setOnItemLongClickListener(this);
282         // 初始化并设置笔记列表适配器
```

图 5.26: 类型转换问题示例

在引用类型向下转换前用instanceof进行判断。

5.2.4 示例4(见图4.28)

81. app/src/main/java/net/micode/notes/ui/NotesListAdapter.java

一般 G.EXP.04 表达式的比较，应该遵循左侧倾向于变化、右侧倾向于不变的原则--表达式比较左变右不变　问题发现时间：2024/12/09 19:47:50 GMT+08:00　修改点个数 1

待处理　负责人：hid_d8iqytkx2rdg7r6　问题帮助　查看上下文　创建问题单

```
220      *
221      * @param position 项的位置
222      * @return 选中状态
223      */
224     public boolean isSelectedItem(final int position) {
225         if (null == mSelectedIndex.get(position)) {
226             return false;
227         }
228         return mSelectedIndex.get(position);
229     }
230
```

图 5.27: 表达式比较问题示例

表达式的比较，应该遵循左侧倾向于变化、右侧倾向于不变的原则–表达式比较左变右不变。

5.3 分析结果汇总（见图5.28、5.29）

G.OBJ.07 子类覆写父类方法或实现接口时必须加上@Override注解	13
G.ERR.03 不要直接捕获可通过预检查进行处理的RuntimeException，如NullPointerException、IndexOutOfBoundsException等	6
G.CTL.01 不要在控制性条件表达式中执行赋值操作或执行复杂的条件判断	3
G.FMT.08 使用空格进行缩进，每次缩进4个空格	5
G.OTH.01 安全场景下必须使用密码学意义上的安全随机数	1
G.NAM.04 方法名应采用小驼峰命名	2
G.CMT.06 注释与代码之间应该有空行或空格，注释符与注释内容之间应该有空格	46
G.CMT.01 public或protected修饰的元素应添加Javadoc注释	260
G.TYP.13 在引用类型向下转换前用instanceof进行判断	37
G.NAM.03 类、枚举和接口名应采用大驼峰命名	2

图 5.28: 分析结果汇总1

G.FMT.07 应该避免空块，必须使用空块时，应采用统一的大括号换行风格	1
G.OTH.03 不用的代码段包括import，直接删除，不要注释掉--不用的import语句，直接删除，不要注释掉	3
G.MET.01 方法要简短--方法的代码块嵌套深度不应超过4层	14
G.MET.01 方法要简短--方法行数不应超过50行	11
G.MET.01 方法要简短--方法的参数不应超过5个	1
G.NAM.05 常量名采用全大写单词，单词间以下划线分隔	1
G.CON.12 避免不加控制地创建新线程，应该使用线程池来管控资源	3
G.FMT.13 用空格突出关键字和重要信息	1
G.CMT.02 顶层public类的Javadoc应该包含功能说明和创建日期/版本信息	40

图 5.29: 分析结果汇总2

根据分析结果图片，我们可以总结出以下主要问题：

5.3.1 最频繁出现的问题

- G.CMT.01: public或protected修饰的元素应添加Javadoc注释 (260处)

G.FMT.05 在条件语句和循环块中应该使用大括号	22
G.EXP.04 表达式的比较，应该遵循左侧倾向于变化、右侧倾向于不变的原则--使用equals方法进行字符串比较	2
G.FMT.12 减少不必要的空行，保持代码紧凑	38
G.OBJ.10 接口定义中去掉多余的修饰词	27
G.FMT.04 一个类或接口的声明部分应该按照类变量、静态初始化块、实例变量、实例初始化块、构造器、方法的顺序出现，且用空行分隔	32
G.CMT.03 方法的Javadoc中应该包含功能说明，根据实际需要按顺序使用@param、@return、@throws标签对参数、返回值、异常进行注释	3

图 5.30: 分析结果汇总3

G.EXP.04 表达式的比较，应该遵循左侧倾向于变化、右侧倾向于不变的原则--表达式比较左变右不变	4
G.OBJ.08 正确实现单例模式	1
G.FMT.20 数字字面量应该设置合适的后缀，long类型应该使用L作为后缀	3
G.DCL.03 禁止C风格的数组声明	5
G.FMT.06 对于非空块状结构，左大括号应该放在行尾，右大括号应该另起一行	2

图 5.31: 分析结果汇总4

- G.CMT.06: 注释与代码之间应该有空行或空格，注释符与注释内容之间应该有空格 (46处)
- G.CMT.02: 顶层public类的Javadoc应该包含功能说明和创建日期/版本信息 (40处)

5.3.2 代码规范性问题

- G.OBJ.07: 子类覆写父类方法或实现接口时必须加上@Override注解 (13处)
- G.TYP.13: 在引用类型向下转换前用instanceof进行判断 (37处)
- G.FMT.12: 减少不必要的空行，保持代码紧凑 (38处)

5.3.3 方法相关问题

- G.MET.01: 方法相关的多个规范问题：
  - 方法的代码块嵌套深度不应超过4层 (14处)
  - 方法行数不应超过50行 (11处)
  - 方法的参数不应超过5个 (1处)

5.3.4 安全性问题

- G.OTH.01: 安全场景下必须使用密码学意义上的安全随机数 (1处)
- G.ERR.03: 不要直接捕获可通过预检查进行处理的RuntimeException (6处)

5.3.5 代码格式问题

- G.FMT.08: 使用空格进行缩进，每次缩进4个空格 (5处)
- G.FMT.04: 类或接口的声明部分应该按照固定顺序出现并用空行分隔 (32处)

5.3.6 主要改进建议

- 加强代码注释规范，特别是Javadoc文档注释的完整性
- 提高代码的安全性，尤其是在随机数生成等关键场景
- 优化方法的结构，控制方法的长度和复杂度

- 规范代码格式，保持统一的缩进和空行使用
- 注意类型安全，在类型转换时进行必要的检查

这些问题虽然多数不会影响程序的功能，但会影响代码的可维护性、安全性和可读性。建议按照问题的严重程度逐步进行优化。