开源代码阅读、标注和维护文档

撰写人员：*****

1. 开源软件的代码泛读

1.1 开源软件的功能描述

根据对小米便签开源软件的使用以及对代码的阅读和理解，该软件的整体功能描述如下，其软件需求的用例模型如图 1 所示。

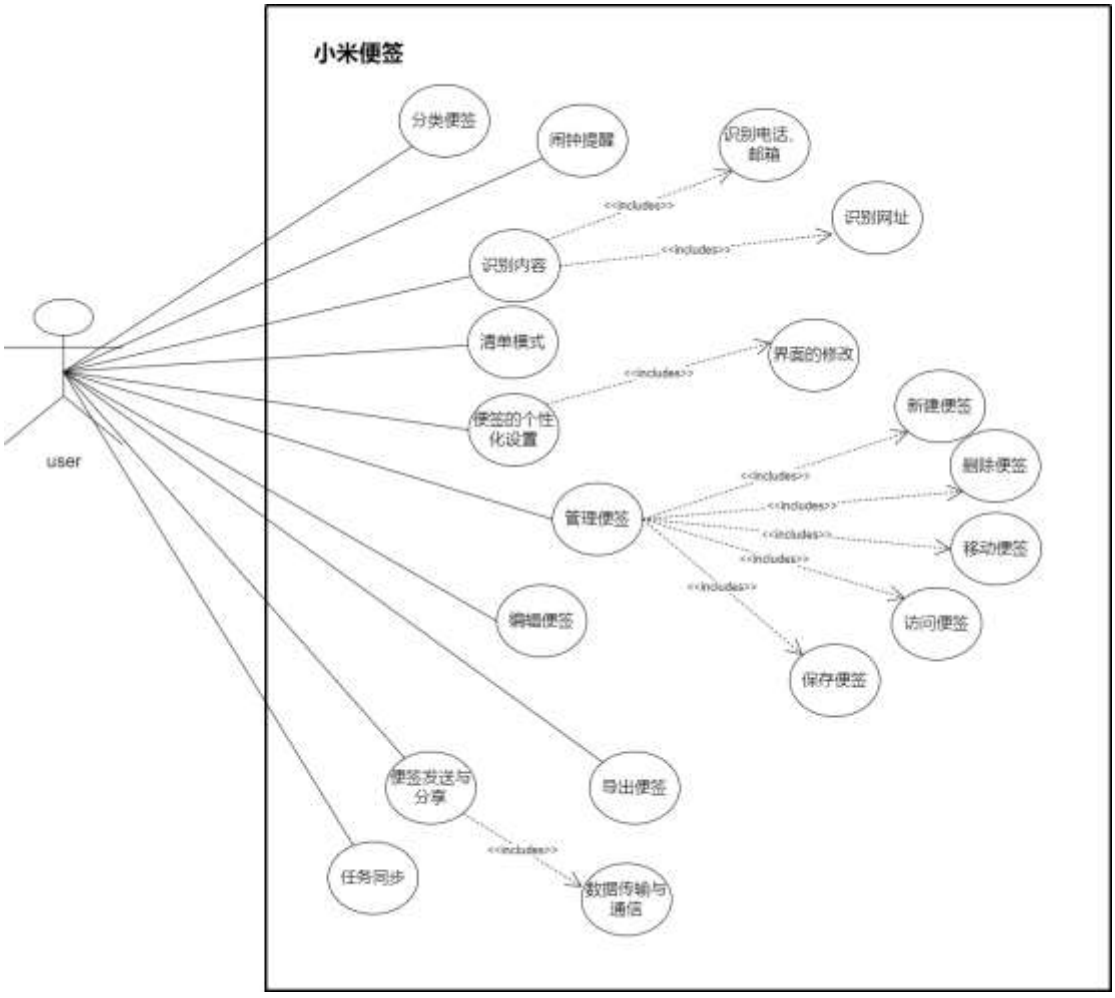


图 1 “小米便签”开源软件的用例图

（1）**功能 1--便捷便签管理：**

小米便签提供了一个直观的用户界面，让您能够迅速新建、编辑、搜索和访问便签。所有更改都会自动保存，确保您的信息随时更新且不会丢失。

（2）**功能 2--便签自由分类：**

便签分类功能，您可以轻松组织和管理您的笔记。无论是工作计划还是个人备忘录，分类功能都能帮助您保持条理清晰。

（3）**功能 3--独特清单模式：**

小米便签的清单模式让您能够以一种简单直观的方式管理日常事务。无论是购物清单还是待办事项，清单模式都能帮助您保持井然有序

（4）**功能 4--闹钟提醒功能：**

设置闹钟提醒，确保您不会错过任何重要的日程或截止日期。小米便签的提醒功能让您的日常生活更加高效。

（5）**功能 5--智能识别：**

小米便签能够智能识别文本中的邮箱、电话号码和网址，让您能够快速进行复制、拨打电话或访问链接，提高您的工作效率

（6）**功能 6--与 Google Tasks 同步：**

通过与 Google Tasks 的同步功能，您可以轻松地将本地便签上传到云端，或者将 Google 工作表中的工作项同步到本地，实现跨设备的信息共享。

（7）**功能 7--快速发送与分享：**

小米便签支持一键发送和分享功能，让您能够迅速与他人共享您的笔记或想法，无论是通过邮件、社交媒体还是其他通讯工具。

（8）**功能 8--个性化设置：**

您可以根据个人喜好调整便签界面和字体等显示设置，打造一个既美观又实用的个性化便签环境。

（9）**功能 9—便签导出：**

导出需要的便签内容和数据，便于进一步使用。

（10）**功能 10—搜索便签：**

利用搜索框，输入想要查找的内容，快速定位便签。

1.2 开源软件的软件架构及各个包和类的作用

小米便签应用的架构设计遵循了分层原则，将不同的功能和职责分配到不同的层次中，以实现清晰的结构和高效的管理。这种设计不仅有助于维护和扩展，还能促进团队的协作开发。以下是小米便签各层及其包之间的关系(图 2 所示)。

1.界面层：这一层直接与用户交互，负责展示便签内容和接收用户的操作。它包括用户界面（UI）组件，用于构建直观的界面；资源（res）存储了应用的静态资源，如图标和布局；控件（widget）则是可复用的界面元素，用于创建丰富的用户界面。

2.业务层：这一层处理小米便签的核心业务逻辑。gtask.remote 包负责与远程服务器的通信，执行如同步便签数据等操作；gtask.exception 包管理应用中的异常，确保业务流程的稳定性；tool 包则包含了通用的工具函数，这些函数在应用的多个地方被复用，提高了代码的效率和一致性。

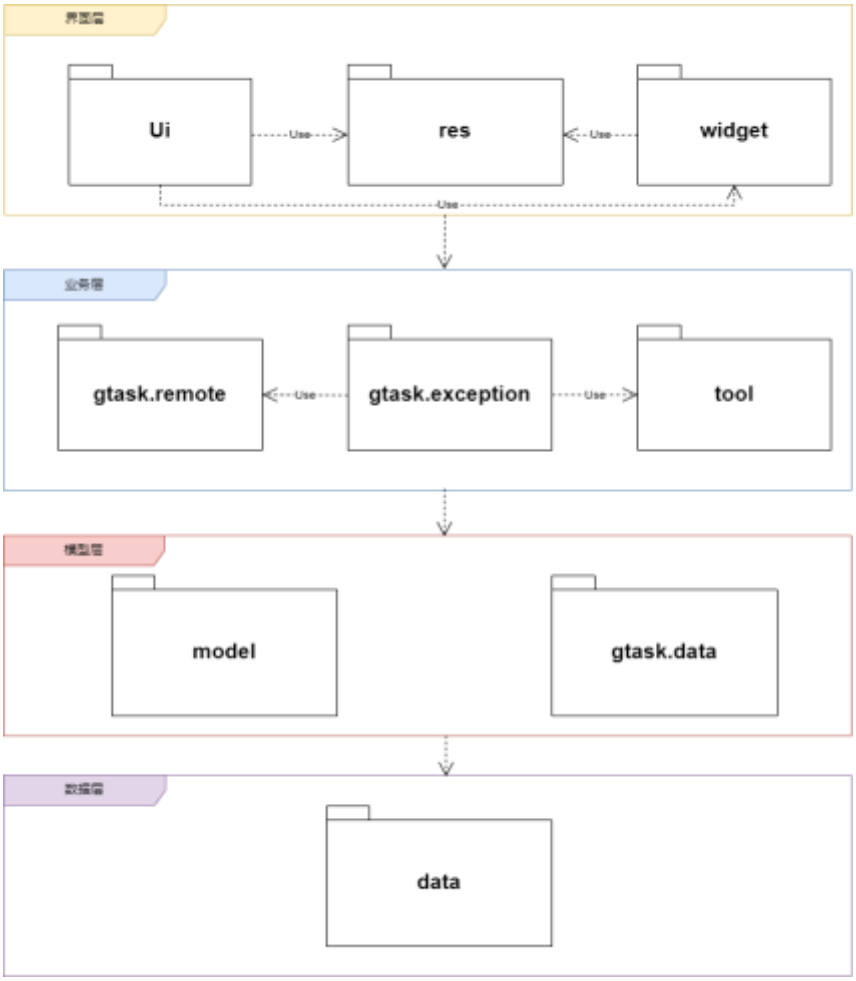


图 2 开源软件的架构图

3.模型层：这一层关注数据的管理和业务规则的执行。model 包定义了便签数据的结构和对象，而 gtask.data 包则负责便签数据的存取和转换，确保数据的一致性和完整性。

4.数据层：这一层负责小米便签的数据持久化。data 包与底层存储系统交互，执行数据的存储、检索和更新操作，确保用户便签信息的安全和可靠。

通过这种分层结构，小米便签的各个组件能够独立开发和测试，同时保持低耦合和高内聚，使得应用更加健壮和易于维护。

小米便签应用以其直观的用户界面和强大的功能，为用户提供了一个高效、便捷的便签管理平台。其背后的软件架构设计遵循了模块化的原则，确保了代码的清晰性、可维护性和扩展性。以下是构成小米便签应用核心架构的六个主要包，每个包都承担着独特的职责（其对应功能如表 1 所示），共同支撑起整个应用的运行：

1. **Data（数据包）**：这个包是小米便签应用的数据管理核心，负责存储和维护便签以及联系人信息。它确保用户的数据被安全地保存在本地数据库中，并且可以通过便签信息提供类方便地访问和操作这些数据。
2. **Gtask（任务包）**：这个包是小米便签应用中处理任务和同步功能的关键部分。它包括：
   - **Data**：管理同步任务的元数据。
   - **Exception**：处理同步过程中可能遇到的异常情况。
   - **Remote**：负责与远程服务的交互，包括任务的同步和异步处理。
3. **Model（模型包）**：这个包定义了小米便签应用的数据模型，包括便签项和当前活动便签项。这些模型是应用业务逻辑的基础，为便签数据的展示和操作提供了结构化的方式。
4. **Tool（工具包）**：这个包提供了一系列的工具类，用于支持便签应用的各种辅助功能。这些工具类简化了常见的操作，如数据备份、便签数据处理和界面元素解析，提高了开发效率和应用的稳定性。
5. **Ui（界面包）**：这个包包含了小米便签应用的所有用户界面组件。它负责展示便签内容、处理用户交互和提供便签管理功能。通过这个包，用户可以方便地创建、编辑、查看和同步便签。
6. **Widget（控件包）**：这个包提供了小米便签的桌面挂件功能。通过不同的挂件大小，用户可以直接在桌面上查看和访问便签，增加了便签应用的便捷性和可访问性。

表 1 小米便签应用包与类功能概览

包	子包	类	主要作用
Data		Contact	联系人数据库，从联系人数据库获取与给定手机号匹配的联系人姓名，包含一个 Contact 类，一个 getContact 函数
		Notes	便签数据库，用于记录便签相关属性和数据
		NotesProvider	便签信息提供类
		NotesDataBaseHelper	数据库帮助类，用于辅助创建、处理数据库

			的条目
Gtask	Data	MetaData	关于同步任务的元数据
		Node	同步任务的管理结点，用于设置、保存同步动的信息
		SqlData	数据库中基本数据，方法包括读取数据、获取数据库中数据、提交数据到数据库
		SqlNode	数据库中便签数据，方法包括读取便签内容、从数据库中获取便签数据、设置便签内容、提交便签数据到数据库
		Task	同步任务，将创建、更新和同步动作包装成 JSON 对象，用本地和远程的 JSON 对结点内容进行设置，获取同步信息，进行本地和远程的同步
		TaskList	同步任务列表，将 Task 组织成同步任务列表进行管理
	Exception	ActionFailureException	动作失败异常
		NetworkFailureException	网络异常失败
	Remote	GTaskAsyncTask	GTask 异步任务，方法包括任务同步和取消，显示同步任务的进程、通知和结果
		GTaskClient	GTask 客户端，提供登录 Google 账户，创建任务和任务列表，添加和删除结点，提交、重置更新、获取任务列表等功能
		GTaskManager	GTask 管理者，提供同步本地和远端的任务，初始化任务列表，同步内容、文件夹，添加、更新本地和远端结点，刷新本地同步任务 ID 等功能
		GTaskSyncService	GTask 同步服务，用于提供同步服务 （开始、取消同步），发送广播
Model		Note	单个便签项
		WorkingNote	当前活动便签项
Tool		BackupUtils	备份工具类，用于数据备份读取、显示
		DataupUtils	便签数据处理工具类，封装如查找、移动、删除数据等操作
		GTaskStringUtils	同步中使用的字符串工具类，为了 jsonObject 提供 string 对象
		ResourceParser	界面元素的解析工具类，利用 R.java 这个类获取资源供程序调用
Ui		AlarmAlertActivity	闹铃提醒界面
		AlarmInitReceiver	闹铃提醒启动消息接收器
		AlarmReceiver	闹铃提醒接收器
		DateTimePicker	设置提醒时间的部件

		DateTimePickerDialog	设置提醒时间的对话框界面
		DropdownMenu	下拉菜单界面
		FoldersListAdapter	文件夹列表链接器（链接数据库）
		NoteEditActivity	便签编辑活动
		NoteEditText	便签的文本编辑界面
		NoteItemData	便签项数据
		NotesListActivity	主界面，用于实现处理文件夹列表的活动
		NotesListAdapter	便签列表链接器（链接数据库）
		NotesListItem	便签列表项
		NotesPreferenceActivity	便签同步的设置界面
Widget		NoteWidgetProvider	桌面挂件
		NoteWidgetProvider_2x	2 倍大小的桌面挂件
		NoteWidgetProvider_4x	4 倍大小的桌面挂件

在深入分析小米便签应用程序的代码结构时，我们采用了类图（每个子包的类图如图 3 至图 8 所示。）来详细描绘各个包中类的交互逻辑。这些类图不仅展示了包内类之间的直接关系，而且还精确地映射了它们之间的依赖、关联、聚合、组合、继承以及实现等六种关系，从而清晰地揭示了包的内部结构。

具体来说，通过类图，我们可以直观地看到类之间的消息传递路径，以及它们如何通过不同的关系相互作用。例如，某些类可能通过依赖关系使用其他类的方法，而关联关系则表明类之间的连接。聚合和组合关系则用于表示类之间的“整体-部分”关系，其中聚合关系表示弱拥有关系，而组合关系则表示强拥有关系。继承关系则用于表示类之间的层次结构，子类继承父类的属性和方法。最后，实现关系则用于表示类如何实现接口或抽象类中定义的方法。

通过这些详细的关系描绘，我们可以更好地理解小米便签应用程序的代码结构，为后续的代码维护和功能扩展提供有力的支持。

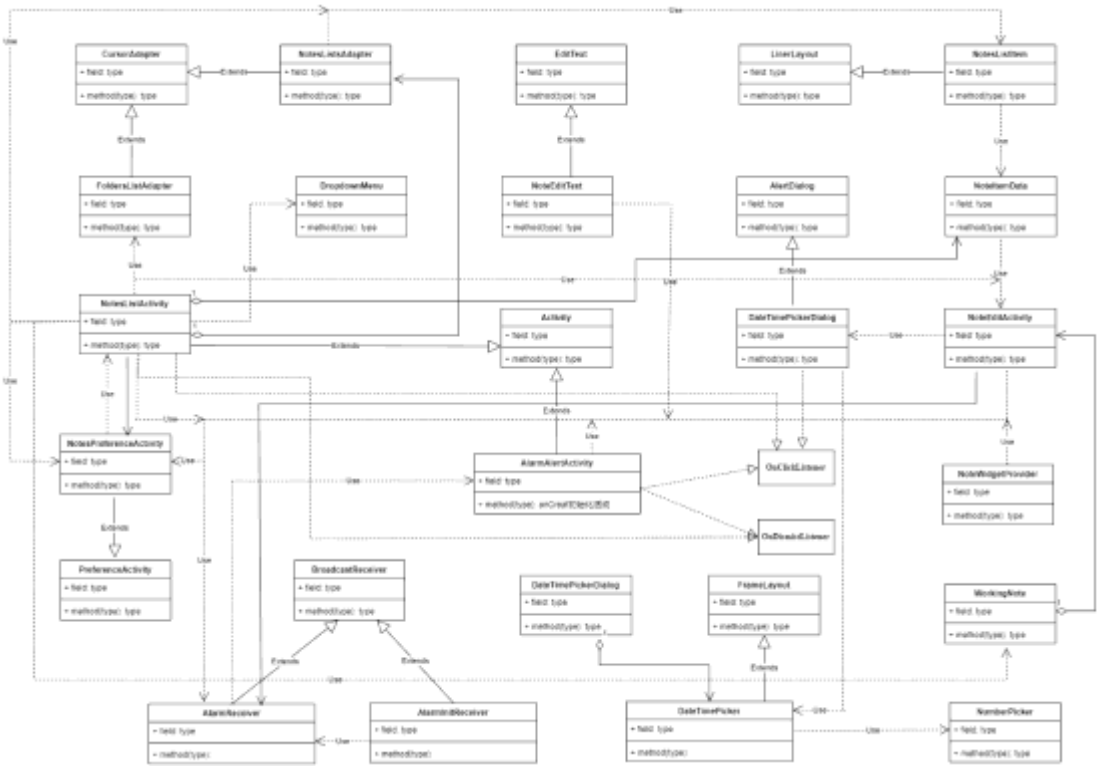


图 3 UI 包的类图

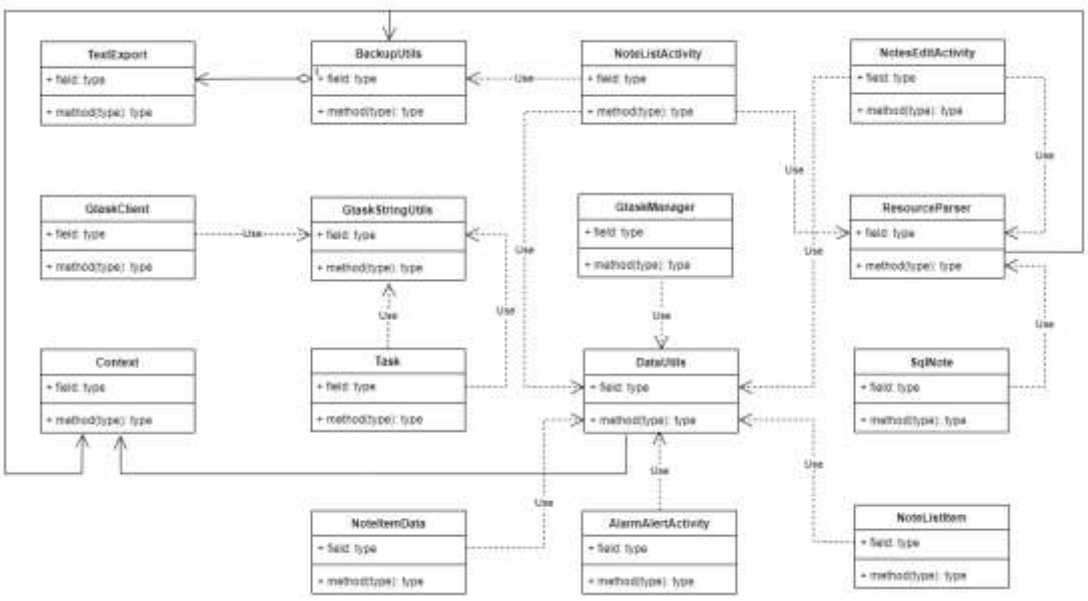


图 4 Tool 包的类图

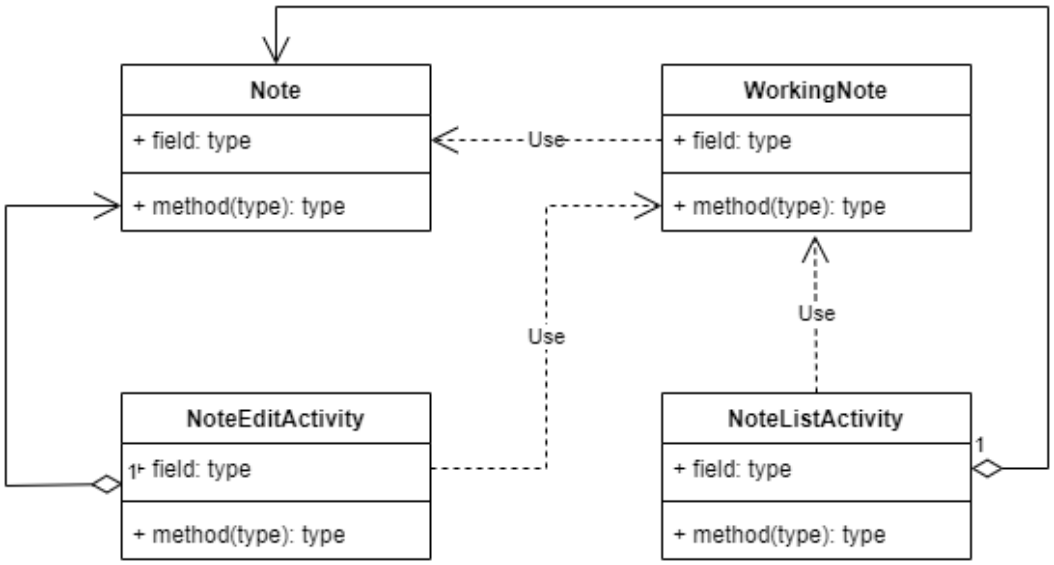


图 5 Model 包的类图

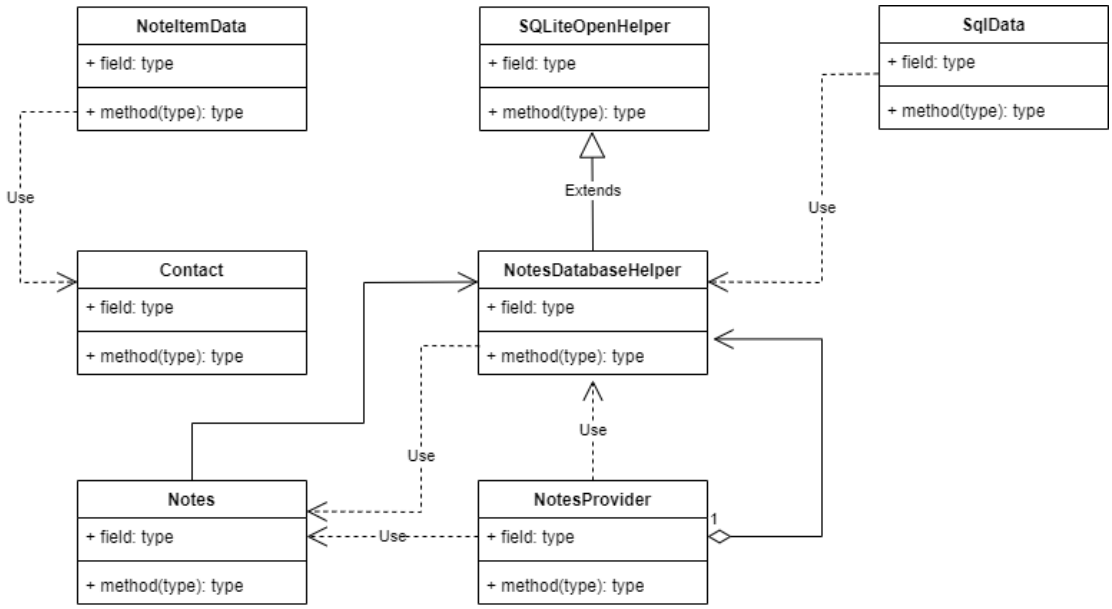


图 6 Data 包的类图

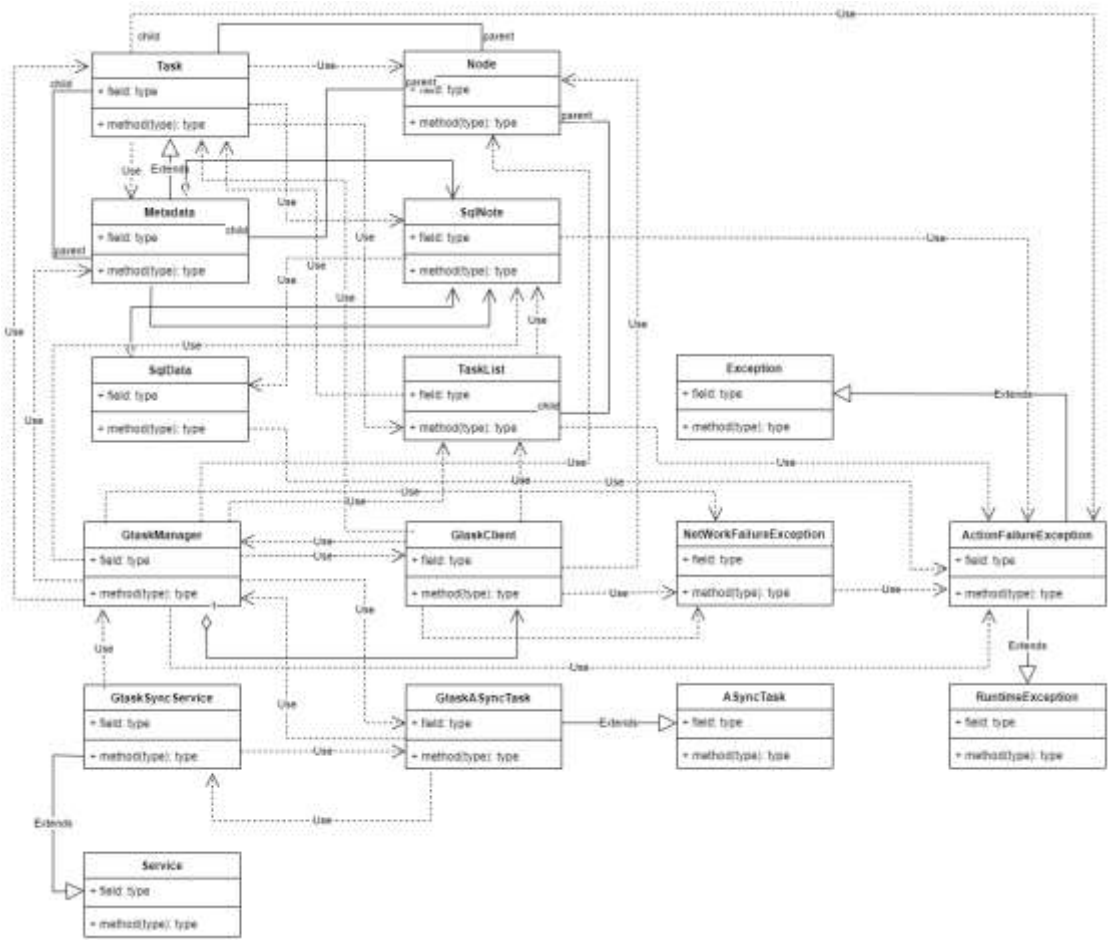


图 7 GTask 包的类图

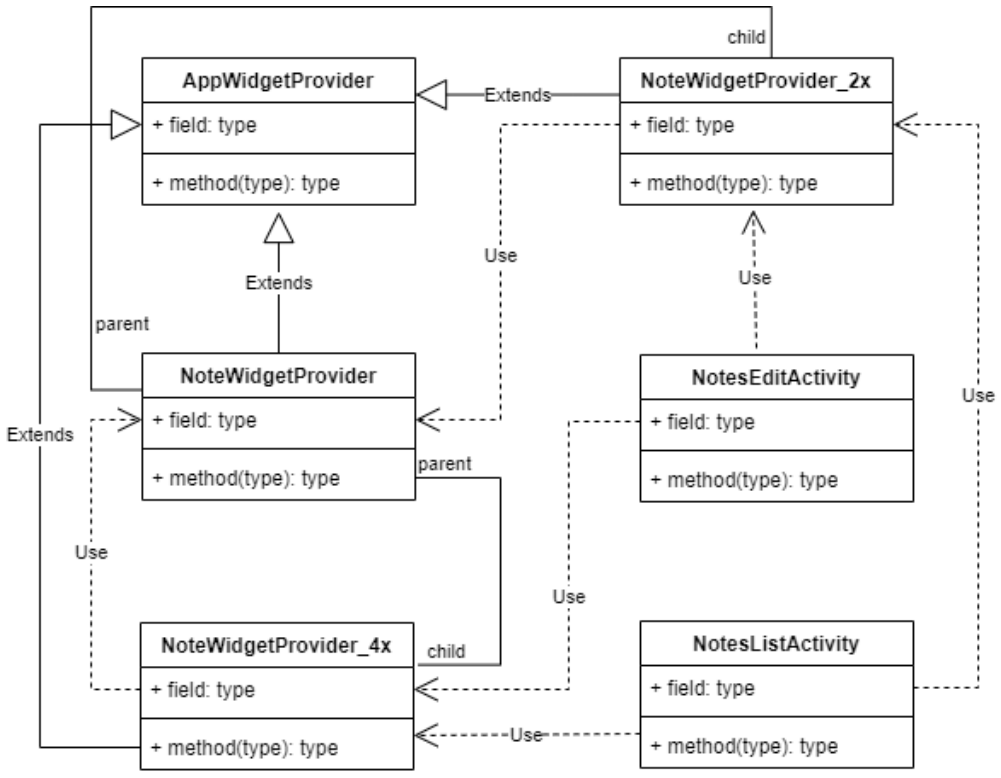


图 8 Widget 包的类图

1.3 软件功能与类间的对应关系

根据对小米便签开源软件代码的理解、标注和分析，软件功能与类之间的关系如表 2 所示。

这张表格是小米便签应用的软件功能与类间的关系对应关系表，它详细记录了小米便签各项功能与其在代码实现中的具体类和方法之间的映射。通过这张表格，我们可以清晰地看到，从新建便签到同步操作，每个功能点是如何通过特定的类和方法协同工作来实现的。

表 2. 软件功能与类间的关系

序号	功能名称	实现模块	实现方法
1	新建便签	NoteEditActivity Notes WorkingNote	createNewNote() getFolerId() startActivity()
2	删除便签	NoteEditActivity Notes WorkingNote	deleteCurrentNote() getNoteId() getContentResolver()
3	移动便签	DataUtils Notes WorkingNote	moveNoteToFolder(), batchMoveToFolder()
4	显示创建或修改文件夹的对话框并处理	NotesListActivity Notes WorkingNote	showCreateOrModifyFolderDialog() showSoftInput() setOnClickListener() addTextChangedListener()
5	打开指定的文件夹，并加载其笔记列表。	NotesListActivity NoteItemData DataUtils Notes FoldersListAdapter	openFolder()
6	删除指定的文件夹。如果是在同步模式下，文件夹会被移动到回收站，	NotesListActivity DataUtils Notes ContentResolver	deleteFolder() batchMoveToFolder() batchDeleteNotes() updateWidget()

	否则直接删除。		
7	显示文件夹列表的菜单。使用查询结果构建一个对话框，让用户选择一个文件夹。	NotesListActivity FoldersListAdapter DataUtils NoteItemData	showFolderListMenu () batchMoveToFolder() makeText()
8	将笔记导出为文本文件。在后台任务中执行导出操作，并根据操作结果展示不同的对话框。	NotesListActivity BackupUtils BackupUtils.TextExport	exportNoteToText() getInstance() onPostExecute()
9	同步	GTaskSyncService DataUtils GTaskManager	startSync()
10	搜索	NotesListActivity ContentResolver NotesListAdapter Notes	onSearchRequest()

11	处理选项菜单项的选择事件。	NoteEditActivity
		NotesListAdapter
		DataUtils
		NotesListActivity
			onOptionsItemSelected()
			getItemId()
			createNewNote()
			setCheckListMode()
			getWorkingText();sendTo();
			sendToDesktop();
			setReminder();

12	修改便签背景颜色	WorkingNote NoteBgResources ResourceParser NoteEditActivity	setBgColorId()
13	进入清单模式	WorkingNote NoteEditText	setCheckListMode()

14	分享笔记到到桌面	NoteEditActivity WorkingNote Notes	sendToDesktop()
15	弹出日期时间选择器，用于设置提醒时间	NoteEditActivity WorkingNote DateTimePicker DateTimePickerDialog	setReminder() OnDateTimeSet() setAlertDate()
16	设置提醒日期，并根据需要触发状态监听器。	NoteEditActivity WorkingNote DateTimePicker DateTimePickerDialog	setAlertDate()
17	识别电话号码	Contact WorkingNote	getContact()
18	分享	NoteEditActivity BackupUtils DataUtils NotesListActivity	getWorkingText(), sendTo()
19	保存便签（自动）	WorkingNote SqlNote NoteEditActivity	saveNote()
20	处理选项菜单项的选择事件	NoteEditActivity NoteEditActivity	onOptionsItemSelected() createNewNote()

1.4 收获

在深入研究小米便签应用的代码后，本组获得了宝贵的经验和知识：

1、深化架构理解：通过对小米便签代码的广泛阅读，我们对应用的架构设计有了更深入的认识。本组理解了它是如何通过分层和模块化来组织代码的，以及它可能采用的设计模式，这为我们提供了一个视角来看待复杂应用的结构。

2、掌握代码规范：阅读代码的过程中，我们学习到了小米开发团队所遵循的代码规范和命名约定。这些规范不仅提升了代码的可读性，也有助于维护和扩展，让我们对行业标准有了更加深刻的理解。

3、提升代码阅读与分析技能：通过分析小米便签的代码，我们的代码阅读和分析能力得到了显著提升。我们学会了如何快速定位关键功能，理解类之间的关系，以及评估代码的质量和性能，这些技能对于我们未来的开发工作将大有裨益。

1.5 存在的问题

1、对 Java 语言的掌握程度不足，对 Java 的应用不了解，缺乏对 Java 代码深入理解和分析的能力。

2、对 Android Studio 开发环境不够熟悉，未能完全掌握其如何高效地管理和关联项目中的各个依赖包。

3、虽能理解部分代码片段，但对整体项目架构中各类之间的交互机制及如何共同构建成一个完整应用的认识尚不清晰。

2. 开源软件的代码标注

在本研究中，我们针对小米便签应用程序中的中共 6 个包和 39 个类进行了详细的代码标注。这些标注涵盖了 4185 行代码，为我们深入理解应用程序的内部结构和功能逻辑提供了宝贵的信息。这些代码注释的详细信息如表 3 所示。

通过这些注释，我们可以更好地理解每个类和包的设计意图、功能实现以及与其他部分的交互逻辑。这些信息不仅有助于我们更好地维护和优化现有代码，还可以为后续的功能扩展和代码重构提供有力的支持。

表 3. 软件功能与类间的关系

序号	包名称	类名称	标注的代码行数
1	data	Contact	14
2		Notes	55
3		NotesDatabaseHelper	119
4		NotesProvider	109
5	Gtask -data	MetaData	41
6		Node	33
7		SqlData	54

8		SqlNote	145
9		Task	97
10		TaskList	160
11	Gtask -exception	ActionFailureException	27
12		NetworkFailureException	9
13	Gtask -remote	GTaskASyncTask	42
14		GTaskClient	219
15		GTaskManager	245
16		GTakSyncService	51
17	model	Note	110
18		WorkingNote	257
19	tool	BackupUtils	148
20		DataUtils	121
21		GTaskStringUtils	53
22		ResourceParser	46
23	ui	AlarmAlertActivity	59
24		AlarmInitReceiver	21
25		AlarmReceiver	16
26		DateTimePicker	196
27		DateTimePickerDialog	47
28		DropdownMenu	35
29		FoldersListAdapter	47
30		NoteEditActivity	369
31		NoteEditText	82
32		NoteItemData	46
33		NotesListActivity	419
34		NotesListAdapter	104

35		NotesListItem	54
36		NotePreferenceActivity	163
37	widget	NoteWidgetProvider_2x	34
38		NoteWidgetProvider_4x	33
39		NoteWidgetProvider	52
共计	6 个包	39 类	3932 行代码标注

3. 开源软件的维护

3.1 维护的内容

我们对开源软件进行了全面的维护工作，具体包括增加新功能、修复代码缺陷等多个方面。这些维护工作的详细情况如表 4 和图 9 所示。这些维护工作显著增强了软件的功能性和稳定性，为其长远发展打下了坚实基础。新增功能满足了用户需求，提升了软件的市场竞争力。同时，修复代码缺陷提高了软件的可靠性，改善了用户体验，并降低了安全风险。

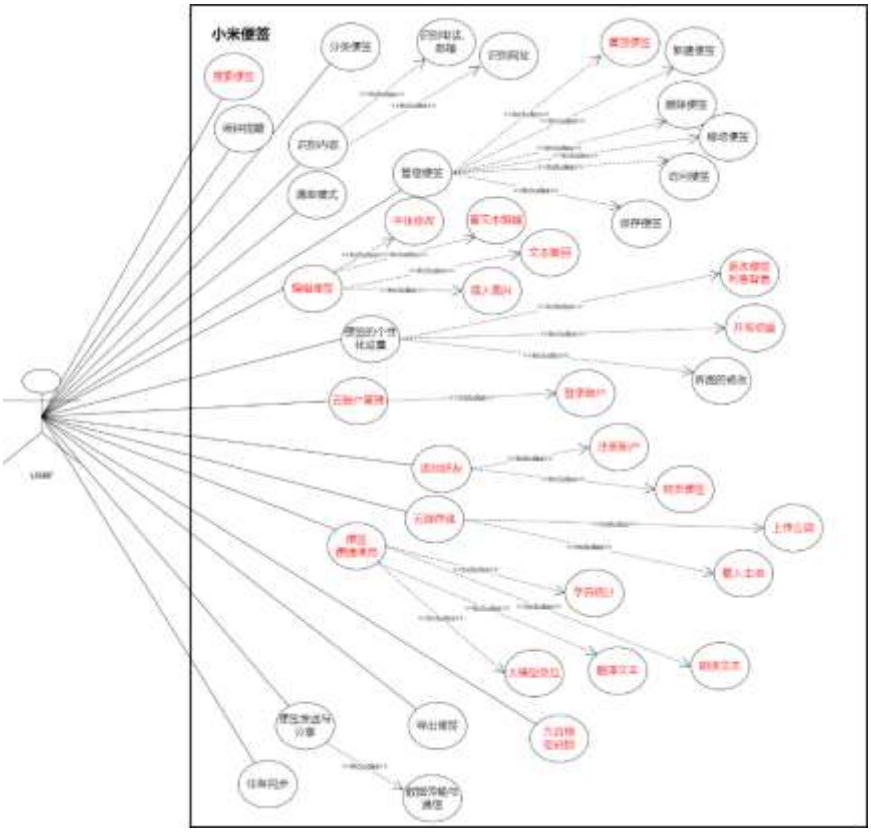


图 9 维护后小米便签用例图

表 4. 开源软件维护的要素

序号	维护类别	名称	描述
1	新增功能	登录/注册	实现登录和注册功能、界面的设计
2	新增功能	云端存储	设计连接阿里云服务器中的数据库实现云端存储
3	新增功能	上传云端	将本地便签上传到云端和便签状态提示
4	新增功能	载入本地	将云端的数据同步到本地
5	新增功能	置顶	将重要便签置顶
6	新增功能	更换背景	更换便签主界面背景图片
7	新增功能	插入图片	在便签编辑界面插入图片
8	新增功能	搜索便签	在主界面搜索相关便签并跳转
9	新增功能	字体修改	修改小米便签字体
10	新增功能	开场动画	增加开场动画
11	新增功能	九宫格密码锁	给单个便签进行手势密码加密
12	新增功能	字符统计	统计便签中的字符数
13	新增功能	文本撤回	撤回上一步输入的文本
14	新增功能	朗读文本	将便签内容朗读出来
15	新增功能	大模型交互	可以与大模型进行聊天对话并复制粘贴有用信息。可以体验讯飞的语音识别、语音合成等功能。
16	新增功能	富文本编辑	增加了便签字体的加粗、下划线、删除线、高亮、斜体功能，并持久化存储。
17	新增功能	百度翻译	实现便签内容中英、中日、中

			韩语言互相转化
18	新增功能	好友管理	添加便签分享好友
19	新增功能	转发便签	将便签笔记分享给好友

3.2 开源软件维护所产生的设计

3.2.1 架构设计

在维护后的小米便签应用中，总体架构设计（如图 10 所示）。依然遵循了分层原则，分为界面层、业务层、模型层和数据层，并在原有基础上新增了多个功能模块，进一步增强了应用的扩展性和用户体验。界面层新增了`SearchAdapter`、`BaiduTranslator`、`SetLockActivity`、`UnlockActivity`、`ChatAdapter`、`LockPatternView`和`SpeechHelper`等类，分别实现了便签搜索、多语言翻译、九宫格手势密码锁、好友管理、语音朗读等功能，提升了用户交互的多样性和便捷性。业务层通过新增的`account`包、`DBConnection`类以及`gtask.remote`和`gtask.exception`模块，支持了用户登录注册、云端存储与同步、远程任务调度和异常处理等功能，确保了数据的安全性和应用的稳定性。模型层通过`model`和`gtask.data`包扩展了数据模型，支持富文本编辑、字体修改、插入图片、置顶功能等个性化设置。数据层通过`data`包实现了便签数据的持久化存储，确保新增功能的长期可用性。新增功能包括登录注册、云端存储与同步、置顶、更换背景、插入图片、搜索便签、字体修改、九宫格密码锁、字符统计、文本撤回、朗读文本、大模型交互、富文本编辑、百度翻译、好友管理和转发便签等，极大地丰富了应用的功能性和用户体验。整体架构设计保持了低耦合和高内聚的特点，新增模块与原有架构无缝集成，确保了代码的清晰性和可维护性，使小米便签应用在功能丰富性和用户体验上得到了全面提升。

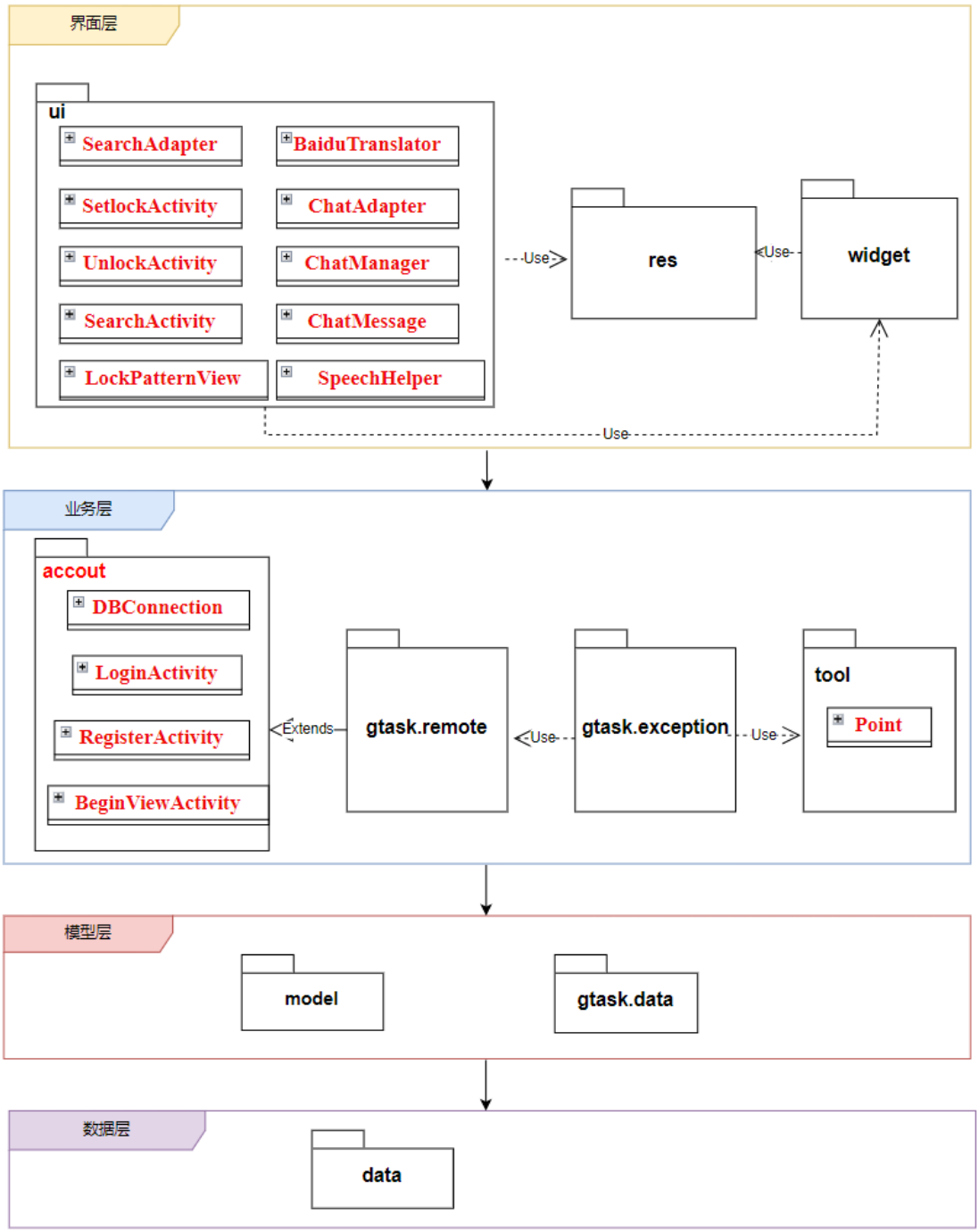


图 10 维护后的软件架构图

3.2.2 界面设计

在小米便签应用的最新维护更新中，我们对用户界面设计进行了全面优化，旨在为用户提供更加直观、易用且功能丰富的交互体验。整个界面设计（如图 11 图 12 所示）围绕用户的核心需求展开，结合新增功能模块，打造了一个功能强大且易于操作的便签管理平台。

应用启动时，用户首先会看到由 SplashActivity 呈现的开场动画界面。这一

界面通过简洁而富有动感的动画设计，为用户带来视觉上的愉悦感，同时传递品牌形象。动画播放完毕后，系统自动跳转到 BeginViewActivity，用户无需任何操作即可进入下一界面。

在 BeginViewActivity 中，用户可以选择通过 LoginActivity 进行登录或通过 RegisterActivity 进行注册。登录和注册界面采用简洁的布局设计，包含用户名、密码输入框以及登录/注册按钮。界面风格统一，操作引导清晰，确保用户能够快速完成账户管理操作。登录成功后，用户将进入 NotesListActivity，即主界面。主界面采用列表形式展示用户的便签内容，顶部包含搜索栏和菜单按钮，底部设有“上传云端”、“载入本地”、“添加好友”等功能按钮。用户可以通过搜索栏输入关键词，触发 SearchActivity 进行便签搜索，搜索结果以列表形式展示，用户可点击进入 NoteEditActivity 进行编辑或查看。

在 NotesListActivity 中，用户可以通过 DBConnection 实现便签的云端管理功能，包括“上传云端”和“载入本地”操作，确保数据在不同设备间的无缝同步。用户还可以通过 BackgroundActivity 更换主界面背景图片，满足个性化需求。新增的好友管理功能允许用户在 NotesListActivity 中添加好友，并通过 ChatManager 实现便签的分享与协作，进一步增强了应用的社交属性。为了提升数据安全性，新增的九宫格手势密码锁功能通过 SetLockActivity 和 UnlockActivity 实现。用户可以为单个便签设置手势密码，并通过解锁界面输入密码访问加密内容，确保隐私数据的安全。

在 NoteEditActivity 中，用户可以进行便签的创建和编辑操作。编辑界面顶部包含保存、返回、字体更改、插入图片等功能按钮，底部设有富文本编辑工具栏（加粗、斜体、下划线等）。用户点击“字体更改”按钮，跳转到 FontChangeActivity，选择字体样式后返回编辑界面。点击“插入图片”按钮，跳转到 ImageInsertActivity，用户可以选择图片插入到便签中。通过富文本编辑工具栏，用户可以对文本进行加粗、斜体、下划线等操作，编辑完成后点击保存按钮，数据通过 database 持久化存储。用户还可以通过 speech 进行语音听写，通过 read 朗读便签内容，通过 tran 选择语言进行翻译，并通过 chat 与大模型交互，获取有用信息。

在 SearchActivity 中，用户可以通过搜索栏输入关键词，系统通过 SearchAdapter 展示匹配的便签内容。用户点击搜索结果中的某一项，系统会跳转

到 NoteEditActivity，用户可以在编辑界面查看或修改便签内容。九宫格密码锁界面采用经典的九宫格布局，用户可以通过绘制手势密码来设置或解锁便签。在 SetLockActivity 中，用户设置手势密码后，系统会保存密码并返回主界面。在 UnlockActivity 中，用户输入正确的手势密码后，系统会跳转到 NoteEditActivity，允许用户查看或编辑加密便签。

语音听写与朗读界面包含一个语音输入按钮和一个文本显示区域，用户可以通过语音输入便签内容。朗读界面则提供一个播放按钮，用户点击后系统会朗读便签内容。用户点击语音输入按钮，系统通过 ise 进行语音识别，并将识别结果显示在文本区域。用户点击朗读按钮，系统通过 iat 进行语音合成，播放便签内容。翻译界面包含语言选择下拉菜单和翻译结果显示区域，用户可以选择目标语言并查看翻译结果。用户选择语言后，系统通过 tran 进行翻译，并将结果显示在界面中，用户可以将翻译结果复制到便签中。

好友管理界面包含好友列表和添加好友按钮，用户可以通过该界面管理好友并分享便签。用户点击“添加好友”按钮，输入好友信息后，系统会保存好友数据。用户可以选择便签并通过 NoteEditActivity 分享给好友。整体而言，小米便签应用的用户界面设计注重功能性与美观性的平衡，新增功能通过直观的交互设计和即时的视觉反馈，进一步提升了用户的操作便捷性和满意度。通过这些优化，小米便签应用在功能丰富性和用户体验上实现了质的飞跃。

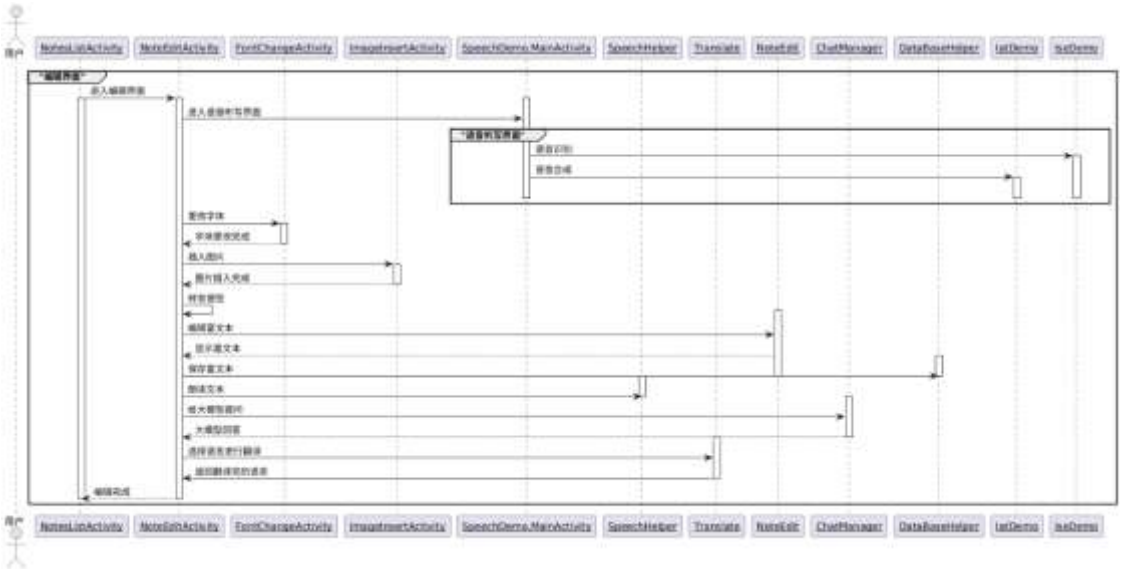


图 11 新增功能界面设计顺序图

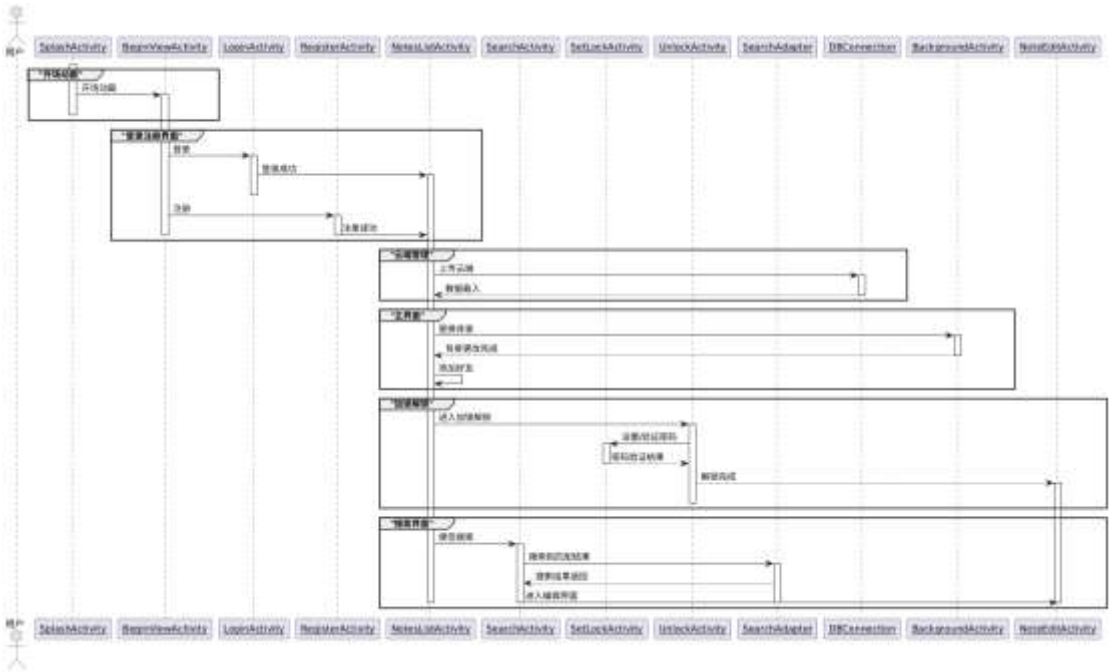


图 12 新增功能界面设计顺序图

3.2.3 详细设计

3.2.3.1 登录和注册功能

(一) 功能介绍

为了保障用户数据的安全性和隐私性，并实现跨设备数据同步，我们小组对小米便签进行了维护，并设计开发了用户账号管理功能。用户可通过注册新账号，系统将新注册的用户信息上传至云端数据库。在用户登录时，系统自动在数据库中检索用户信息，若用户未注册，则提示注册；若用户已注册，则进行账号密码验证。验证成功后，系统将用户数据库信息载入本地，以实现不同用户的个性化使用体验。

(二) 功能详细设计

针对登录和注册功能的设计，本小组添加了一个 account 包（如图 13 所示），其中包含四个类分别是 RegisterActivity、LoginActivity、RegisterAcitvity、DBConnection，并对 ui 包和 data 包进行了修改。account 包的类图如图 14 所示。

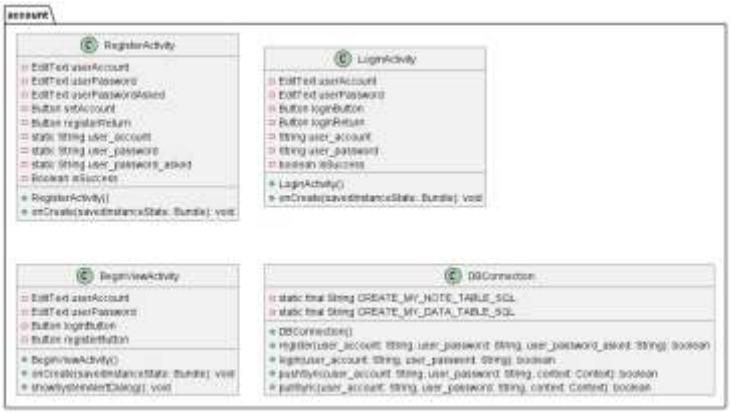


图 13 account 包图

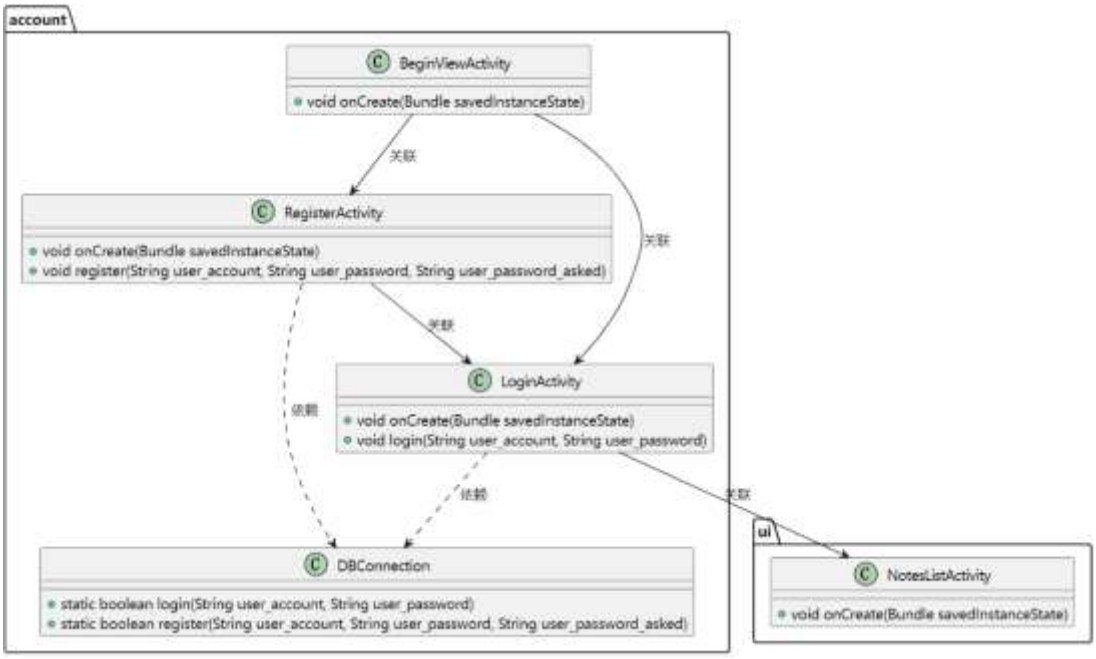


图 14 登录注册功能类图

类之间的逻辑是 LoginActivity 和 RegisterActivity 这两个类分别处理用户的登录和注册逻辑。用户在登录界面输入账号和密码后，LoginActivity 会调用 DBConnection 类中的 login 方法进行验证。用户在注册界面输入账号和密码后，RegisterActivity 会调用 DBConnection 类中的 register 方法进行注册。DBConnection 该类负责与数据库进行交互，提供 login 和 register 方法。login 方法用于验证用户的账号和密码。register 方法用于将新用户的信息保存到数据库中。BeginViewActivity 该类在应用启动时显示系统提示对话框。如果用户登录失败，会显示提示对话框，提示用户注册。NotesListActivity 该类在用户成功登录后显示笔记列表。从 LoginActivity 或 RegisterActivity 中传递用户账号和密码，用于同步用户数据。Notes 该类定义了应用程序中的数据结构和常量。在数据库操作和数据传递中使用。登录和注册功能的顺序图如图 15 图 16 注册功能的顺序

图所示。

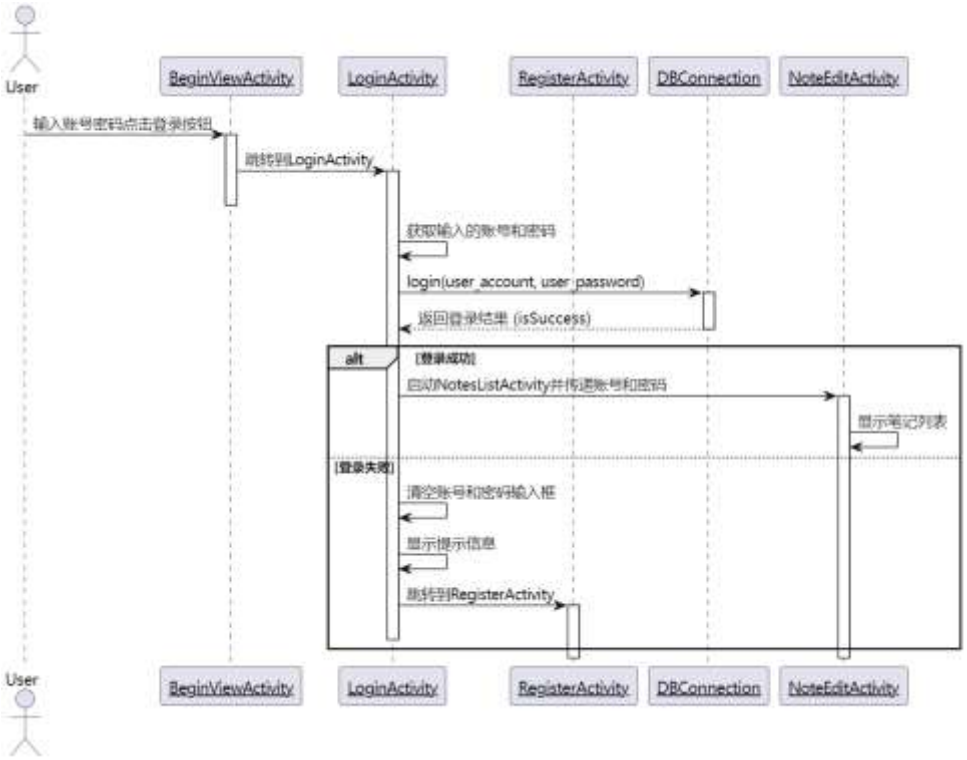


图 15 登录功能顺序图

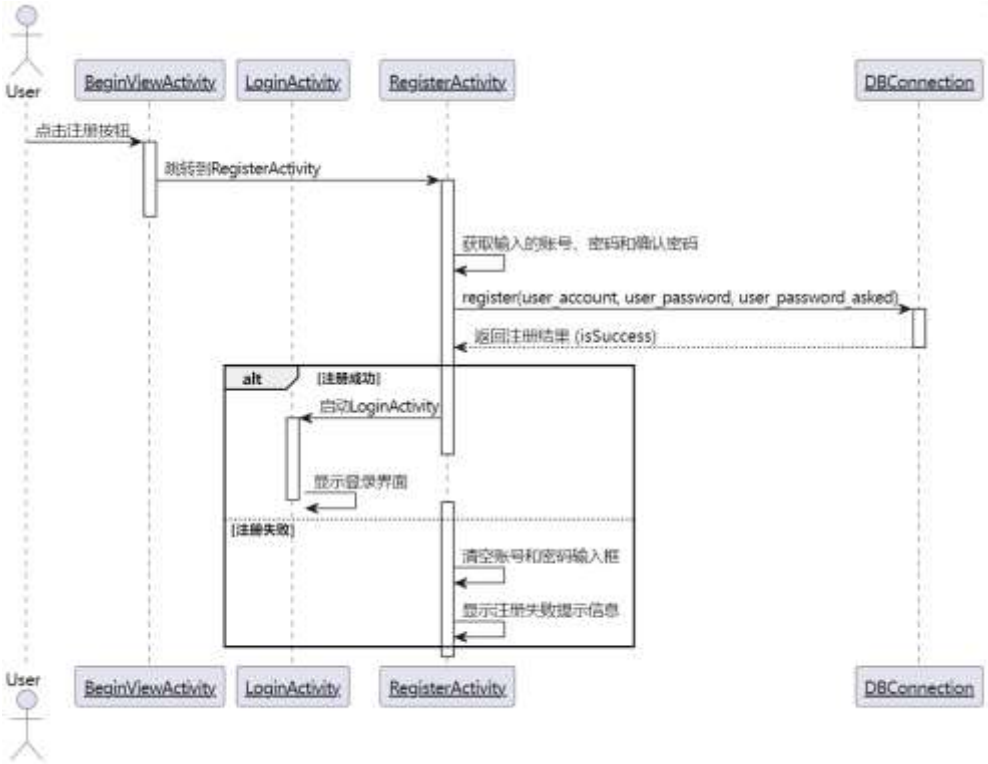


图 16 注册功能的顺序图

(三) 代码实现

本小组通过添加 account 包来实现便签应用中的登录和注册功能，确保用户

认证和账户管理的安全性和有效性。

1.用户登录流程：用户登录的流程涉及到多个类的协作，从用户输入账户密码到系统验证并跳转至便签列表界面。

在 BeginViewActivity 这个类中，启动了登录的界面，本小组在 layout 包里面设计添加了 begin_viewplus 这个 xml 文件，提供美观简洁的登录界面。用户在该界面输入用户信息，如果系统收到登录按钮的响应则会进入登录功能的流程，即获取信息，信息传入 DBConnection 类进行数据匹配，最后根据匹配结果提供系统响应，该功能的具体实现代码如图 17 所示。如果匹配成功则跳转到 NotesLis tActivity 提供用户便签列表。完成登录操作。图 18 为具体的代码实现。

```
try {
    //2、登录
    /**
     * 查找用户是否存在
     */
    sqlFind = "SELECT * FROM xiaomi_serve.note ";
    preparedStatement = conn.prepareStatement(sqlFind);
    resultSet = preparedStatement.executeQuery();
    while (resultSet.next()) {
        String user_account_registered = resultSet.getString(columnLabel:"account");
        String user_password_registered = resultSet.getString(columnLabel:"password");
        //判断账号密码正确性
        if (user_account_registered.equals(user_account)) {
            System.out.println(x:"账号正确");
            if (user_password_registered.equals(user_password)) {
                System.out.println(x:"密码正确");
                return true;
            } else {
                System.out.println(x:"密码不正确");
                return false;
            }
        } else {
            System.out.println(x:"账号不正确");
        }
    }
} catch (SQLException e) {
    System.out.println(x:"MySQL操作错误");
    e.printStackTrace();
```

图 17 验证登录信息合法性的具体代码实现图

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.begin_viewplus);

    userAccount = (EditText)findViewById(R.id.login_edit_account);
    userPassword = (EditText)findViewById(R.id.login_edit_pwd);
    loginButton = (Button)findViewById(R.id.login_btn_login);
    registerButton = (Button)findViewById(R.id.login_btn_register);

    loginButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String account = userAccount.getText().toString();
            String password = userPassword.getText().toString();
            new Thread(new Runnable() {
                @Override
                public void run() {
                    boolean isSuccess = DBConnection.login(account, password);
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            if (isSuccess) {
                                Intent intent = new Intent(BeginViewActivity.this, NotesListActivity.class);
                                intent.putExtra("user_account", account);
                                intent.putExtra("user_password", password);
                                intent.setAction("android.intent.action.CALLr");
                                startActivity(intent);
                                finish();
                            } else {
                                showSystemAlertDialog();
                            }
                        }
                    });
                }
            }).start();
        }
    });
```

图 18 登录功能的具体代码实现图

2. 用户注册流程：用户注册流程同样涉及多个类的协作，从用户输入信息到系统验证并创建新用户。用户可以在 BeginViewActivity 提供的“注册”按钮进入注册流程，该消息的传递由图 19 实现。

```
registerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(BeginViewActivity.this, RegisterActivity.class);
        intent.setAction("android.intent.action.CALLr");
        startActivity(intent);
        finish();
    }
});
```

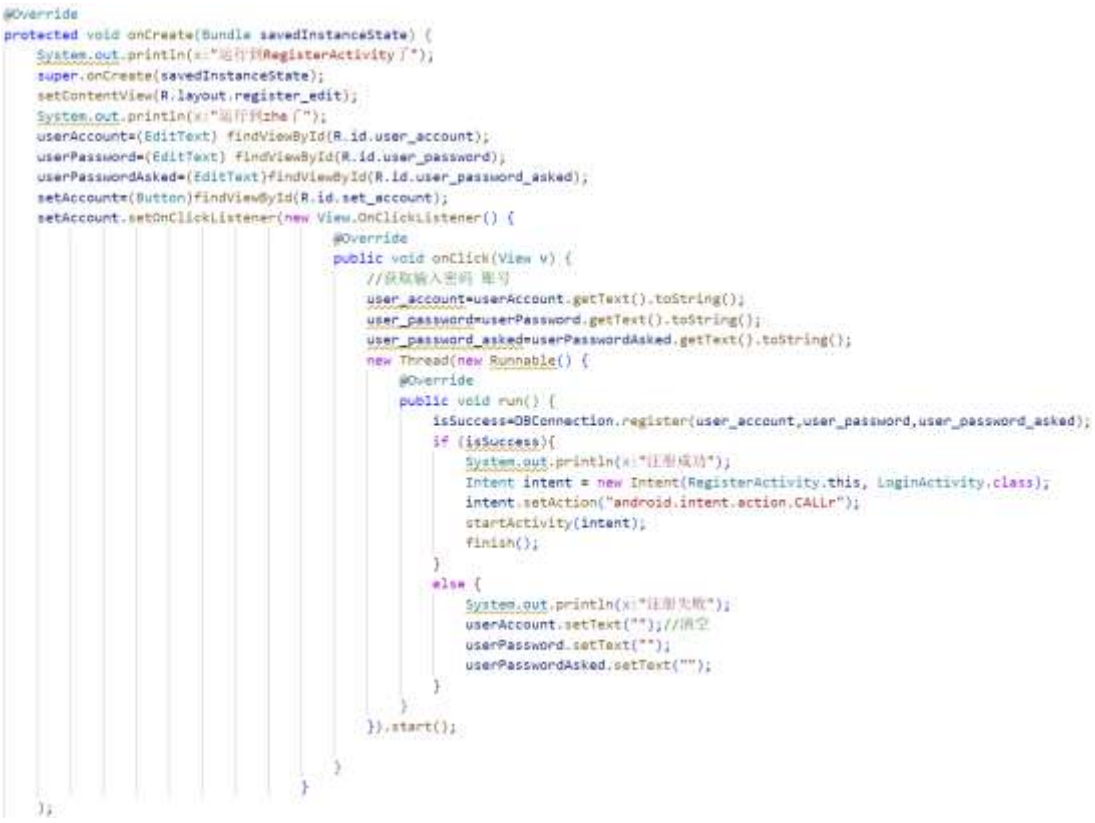
图 19 注册按钮的响应代码实现图

用户进入注册流程后即进入到 RegisterActivity 类中进行处理用户的注册逻辑，包括获取用户输入、调用数据库注册新用户以及处理注册结果。本小组在 layout 包中添加了一个 register_edit 的 xml 文件来显示实现注册功能的界面，用户通过完成注册信息的输入，系统获取用户输入的注册信息，并将注册信息传入 DBConnection 类中的 register 方法，进行注册功能的数据库处理，具体的实现代码如图 20 所示。该数据库操作返回注册的状态，系统通过该状态进行响应，如

果失败则显示日志信息，如果成功则跳转到 LoginActivity 类中进行登录的功能流程。LoginActivity 的具体代码实现流程已经在之前介绍了，这里就不再赘述。具体关于注册功能的系统响应的具体代码如图 21 所示。

```
public class DBConnection {
    public static boolean register(String user_account, String user_password, String user_password_asked) {
        try {
            //1. 注册 传送数据到数据库
            if (!user_password.equals(user_password_asked)) {//判断两次密码输入是否一致
                System.out.println(x:"两次输入密码不一致");
                return false;
            } else {
                System.out.println(x:"两次输入密码一致");
            }
            /**
             * 查找用户是否存在
             */
            sqlFind = "SELECT * FROM xiaomi_serve.note ";
            preparedStatement = conn.prepareStatement(sqlFind);
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                String user_account_registered = resultSet.getString(columnLabel:"account");
                if (user_account_registered.equals(user_account)) {
                    System.out.println(x:"该账号已经注册");
                    return false;
                }
            }
            sqlRegister = "REPLACE INTO xiaomi_serve.note(account,password,note_id,data_id,is_pushsync) values (?,?,?,?,?)";
            preparedStatement = conn.prepareStatement(sqlRegister);
            preparedStatement.setString(parameterIndex:1, user_account);
            preparedStatement.setString(parameterIndex:2, user_password);
            preparedStatement.setString(parameterIndex:3, user_account + "_note");
            preparedStatement.setString(parameterIndex:4, user_account + "_data");
            preparedStatement.setString(parameterIndex:5, x:"0");//0标志该账户还没有同步上传过，即没有还没有创建相应的表
            System.out.println("注册账号是: " + user_account);
            System.out.println("注册密码是: " + user_password);
            //注册的时候创建相应的表
            sqlFind = "CREATE TABLE " + user_account + "_" + CREATE_MY_NOTE_TABLE_SQL;
            PreparedStatement preparedStatementTable = conn.prepareStatement(sqlFind);
            preparedStatementTable.execute();

            sqlFind = "CREATE TABLE " + user_account + "_" + CREATE_MY_DATA_TABLE_SQL;
            preparedStatementTable = conn.prepareStatement(sqlFind);
            preparedStatementTable.execute();
            System.out.println(x:"创建成功");
            //
            int result = preparedStatement.executeUpdate();
            if (result != 0) {
                return true;
            }
```

图 20 注册功能的数据库操作代码实现图



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    System.out.println(x:"运行到RegisterActivity了");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.register_edit);
    System.out.println(x:"运行到zhe了");
    userAccount=(EditText) findViewById(R.id.user_account);
    userPassword=(EditText) findViewById(R.id.user_password);
    userPasswordAsked=(EditText)findViewById(R.id.user_password_asked);
    setAccount=(Button)findViewById(R.id.set_account);
    setAccount.setOnClickListener(new View.OnClickListener() {
                                      @Override
                                      public void onClick(View v) {
                                          //获取输入密码 账号
                                          user_account=userAccount.getText().toString();
                                          user_password=userPassword.getText().toString();
                                          user_password_asked=userPasswordAsked.getText().toString();
                                          new Thread(new Runnable() {
                                              @Override
                                              public void run() {
                                                  isSuccess=DBConnection.register(user_account,user_password,user_password_asked);
                                                  if (isSuccess){
                                                      System.out.println(x:"注册成功");
                                                      Intent intent = new Intent(RegisterActivity.this, LoginActivity.class);
                                                      intent.setAction("android.intent.action.CALLr");
                                                      startActivity(intent);
                                                      finish();
                                                  }
                                                  else {
                                                      System.out.println(x:"注册失败");
                                                      userAccount.setText("");//清空
                                                      userPassword.setText("");
                                                      userPasswordAsked.setText("");
                                                  }
                                              }
                                          }).start();

                                      }
                                  }
    );
```

图 21 注册功能的具体代码实现图

3. 用户界面反馈：在登录和注册过程中，系统通过界面反馈用户操作结果，提升用户体验。在登录失败时，系统会清空输入框并显示提示信息“系统提示：账号或密码错误，请先注册”。当用户选择确认按钮，系统将自动跳转到注册界面，方便用户进行注册的操作。具体的代码实现如图 22。

```
private void showSystemAlertDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(BeginViewActivity.this);
    View view = LayoutInflater.from(BeginViewActivity.this).inflate(R.layout.dialog_is_to_register, null);
    builder.setView(view);

    TextView title = (TextView)view.findViewById(R.id.dialogTitle);
    title.setText("系统提示");

    TextView message = (TextView)view.findViewById(R.id.dialogMessage);
    message.setText("账号或密码错误，请先注册");

    Button positiveButton = (Button)view.findViewById(R.id.positiveButton);
    positiveButton.setText("确定");
    positiveButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(BeginViewActivity.this, RegisterActivity.class);
            intent.setAction("android.intent.action.CALLr");
            startActivity(intent);
            finish();
        }
    });

    Button negativeButton = (Button)view.findViewById(R.id.negativeButton);
    negativeButton.setText("返回");

    final AlertDialog dialog = builder.create();
    negativeButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            dialog.dismiss();
        }
    });

    dialog.show();
}
```

图 22 登录失败系统提示的代码实现图

3.2.3.2 云端存储功能

(一) 功能介绍

该功能主要是搭建部署云服务，功能旨在为用户提供一个数据同步解决方案，使得用户可以在多个设备之间同步和访问他们的便签数据。这项功能通过与云端服务器的集成，允许用户将本地便签上传到云端，同时也能从云端载入最新的便签数据到本地设备。

(二) 功能详细设计

通过购买阿里云服务，使用 xshell 和宝塔面板对 ECS 进行部署与管理，使用 Navicat Premium 16 连接和管理云数据库，通过设置数据库用户权限和安全组规则，实现了数据库的远程连接。最后通过设计 DBConnection 这个类，实现数据库的连接，完成对云端数据库的创建。

(三) 代码实现

1. 云服务搭建：首先进行需求识别，确定需要云数据库服务，可选择购买 ECS（Elastic Compute Service）或直接购买 RDS（Relational Database Service）。

之后进行服务器的连接可以利用 Xshell 软件连接和管理云服务器。云服务的配置方面，本小组通过安装宝塔面板进行云服务的配置，具体步骤是在 Xshell 中添加云服务器地址，执行宝塔面板的安装命令。在阿里云中修改安全组策略，开放宝塔面板端口（默认 8888）。使用用户密码登录宝塔面板，并在官网注册宝塔账户。之后进行云数据库配置，在云服务器中下载 MySQL，如果安装了宝塔面板，可以直接在软件商店下载。在宝塔面板中可视化创建数据库账户和密码。设置安全组规则，开放数据库默认端口 3306。在云服务器中设置允许远程连接数据库。

2.连接云数据库：本小组通过使用 Navicat Premium 16 中添加数据库完成连接。通过对云数据库的主机信息进行填写，测试连接。在完成数据库的连接之后，新建小米便签的数据库，为登录、注册、云同步功能完成数据库的设计与构建。具体设计如图 23 所示。

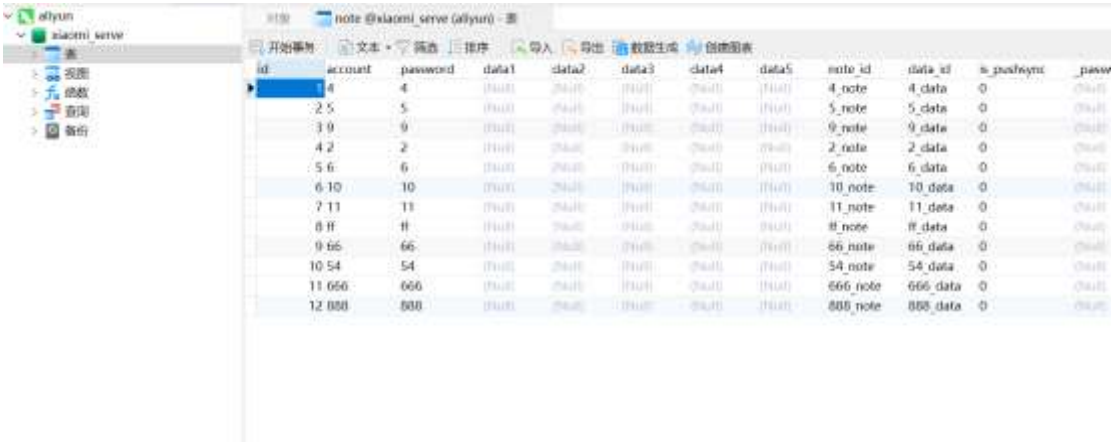


图 23 小米云服务数据库的设计表图

3.2.3.3 上传云端功能

(一) 功能介绍

小米便签应用的“上传云端”功能允许用户将本地便签数据安全地同步到云端服务器，实现跨设备的数据共享和备份。用户通过简单的操作即可触发上传，系统会自动检查用户认证，加载 MySQL 驱动，并从本地数据库读取便签内容，随后加密上传至云端。此功能不仅确保了用户数据的安全性，防止因设备丢失或损坏导致数据丢失，还提供了便捷的跨设备访问能力，使得用户可以在任何设备上访问最新的便签信息。操作过程中，系统会显示上传进度和状态，以便用户了解同步情况，且在上传完成后确保云端数据与本地数据的一致性。这一功能的设计，旨在提供数据安全、操作便捷和跨设备同步的优势，同时提醒用户注意网络

连接要求和隐私保护。

(二) 功能详细设计

本小组在 account 包中添加了 DBConnection 类，在类中设计了 pullSync 这个方法，并在 NoteListActivity 这个类中增加调用 pushSync。云同步功能具体实现的类关系图如图 24 所示。用户通过使用菜单中的“上传云端”按钮，系统自动加载 MySQL 驱动并建立与云端数据库的连接，连接成功后系统检查用户是否存在于云端数据库中，验证成功后从本地数据库中读取笔记和数据表的内容，并将其插入到云数据库中。并将用户的同步标志更新为已同步。云同步的具体功能之间的交互如顺序图（图 25）。

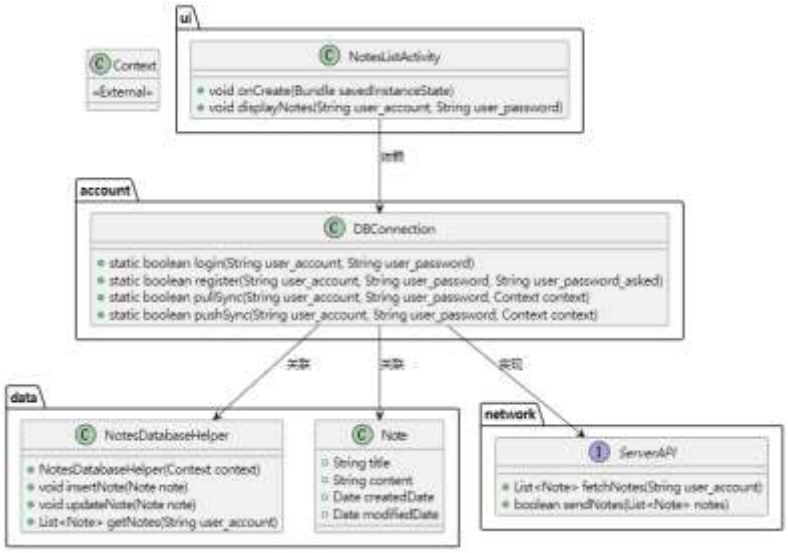


图 24 实现云同步功能的类图

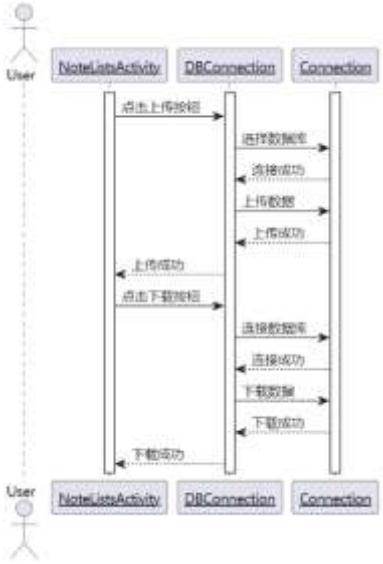


图 25 数据库操作顺序图

(三) 代码实现

上传云端功能流程：上传云端功能涉及多个类的协作，从用户触发上传操作到系统将便签数据上传至云端服务器。在本功能中由 NotesListActivity 提供上传按钮供用户触发上传操作，具体实现代码如图 26 所示。

```
/**
 * 启动同步操作。
 * 如果用户确认操作，将启动一个新线程执行同步操作。
 */
public void upSync(Context context,String user_account,String user_password) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            DBConnection.pushSync(user_account,user_password, NotesListActivity.this);
        }
    }).start();
}
```

图 26 上传云端功能响应代码实现图

之后由 DBConnection 类中的 pushSync 处理上传云端的逻辑，包括从本地数据库获取便签数据并上传至服务器。具体的操作是，系统通过数据库操作连接云数据库，连接成功后查询用户是否存在，通过对本地 note 表和 data 表表项的同步设计，设计出属性相同的云数据库表项，通过插入语句完成将本地数据上传到云数据库的操作。具体的代码实现如图 27 所示。

```
public class DBConnection {
    public static boolean pushSync(String user_account, String user_password, Context context) {
        try {
            /**
             * 查找用户是否存在
             */
            sqlFind = "SELECT * FROM xiaomi_serve.note ";
            conn = DriverManager.getConnection(url, user, password);
            preparedStatement = conn.prepareStatement(sqlFind);
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {…
            //conn=DriverManager.getConnection(url, user, password);
            conn = DriverManager.getConnection(url, user, password);
            Cursor cursor = context.getContentResolver().query(Notes.CONTENT_NOTE_URI, WorkingNote.UP_NOTE_PROJECTION,
                    null, null, null);
            //上传先删除云端库对应的表
            sql = "DELETE FROM " + user_account + "_note";
            PreparedStatement psDelNote = conn.prepareStatement(sql);
            psDelNote.execute();
            //检查是否删除成功
            sql = "SELECT * FROM " + user_account + "_note";
            PreparedStatement psCheckDelNote = conn.prepareStatement(sql);
            ResultSet rsCheckDelNote = psCheckDelNote.executeQuery();
            if (rsCheckDelNote.next()) {
                System.out.println(x:"删除失败");
            } else {
                System.out.println(x:"删除成功");
            }

            //再同步
            sql = "INSERT INTO " + user_account + "_note " + INSERT_NOTE_TABLE_SQL +
                    " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"; // 20 个占位符
            if (cursor != null) {
                if (cursor.moveToFirst()) {…
                } else {…
                //************上传DATA表原理同Note表**************
                sql = "DELETE FROM " + user_account + "_data";
                PreparedStatement psDelData = conn.prepareStatement(sql);
                psDelData.execute();
                cursor = context.getContentResolver().query(Notes.CONTENT_DATA_URI, WorkingNote.UP_DATA_PROJECTION,
                        null, null, null);
                sql = "INSERT INTO " + user_account + "_data " + INSERT_DATA_TABLE_SQL +
                        " VALUES (?,?,?,?,?,?,?,?,?,?)";
```

图 27 方法 pushSync 具体代码实现图

3.2.3.4 载入本地功能

(一) 功能介绍

小米便签应用的“载入本地”功能旨在将云端数据库中的最新便签数据同步到用户的本地设备上，确保多设备间的数据一致性。用户通过简单的操作即可触发载入过程，系统会自动检查网络连接和用户身份，加载 MySQL 驱动以建立与云端的连接，并安全地下载云端数据。在数据合并与更新过程中，系统会提供进度反馈，并在完成后提示用户，确保本地便签列表显示最新的信息。此功能强调数据的实时同步、安全性以及用户体验的优化，使用户能够在任何设备上便捷地访问最新的便签内容，同时保证了数据的隐私性和安全性。

(二) 功能详细设计

本功能主要是在 DBConnection 这个类中添加了 pullSync 这个方法，通过修改 Notes 和 NotesListActivity 这两个类完成笔记和文件夹相关常量和数据列接口的设计和同步操作触发方式的设计。整体来说，类之间的关系是 NotesListActivi

ty 类通过调用 DBConnection 类中的 pullSync 方法来实现从云端数据库载入数据到本地数据库。DBConnection 类使用 Notes 类中定义的常量和数据列接口来操作本地数据库。云同步功能具体实现的类关系图如图 24 所示。

具体而言，本小组设计的 pullSync 这个方法先加载 MySQL 数据库的驱动建立与云端数据库的连接，之后再云端数据库中是否存在指定用户的账户信息，验证完成后删除本地数据库中现有的笔记和数据表内容，以便插入新的数据，将查询到的云端数据库中的笔记数据插入到本地数据库的笔记表中。最后使用 Toast 显示提示信息，这就完成了从云端数据库下载数据并插入到本地数据库的操作。云同步的具体功能之间的交互如顺序图 25 所示。

(三) 代码实现

便签载入流程：便签载入流程涉及多个类的协作，从用户进入便签列表界面到系统从本地数据库加载便签数据并显示。具体的工作流程是在登录便签获取便签列表的时候，我们在 NotesListActivity 中添加触发响应，实现在用户进入便签列表界面时，自动触发便签数据的载入操作。具体代码的实现如图 28 所示。

```
//获得从登录活动中传递过来得账号密码
Intent intent =getIntent();
USERACCOUNT=intent.getStringExtra("user_account");
USERPASSWORD=intent.getStringExtra("user_password");
//登录及下拉数据库内容

new Thread(new Runnable() {
    @Override
    public void run() {
        DBConnection.pullSync(USERACCOUNT, USERPASSWORD, NotesListActivity.this);
    }
}).start();
```

图 28 载入功能响应代码实现图

本小组也在菜单列表中添加“载入本地”功能的按钮，用户可以实现手动将云数据库中的内容同步到本地的操作。不仅如此，我们还在 layout 包中添加了 dialog_is_to_pull 的 xml 文件来给出用户系统提示，方便用户正确处理载入本地功能的操作。具体实现的代码如图 29 所示。

```
/*
 * 作用总结
    显示对话框：提示用户输入密码并确认是否进行同步操作。
    启动同步操作：如果用户确认操作，将启动一个新线程执行同步操作。
    取消操作：如果用户取消操作，对话框将关闭。
 */
public Boolean isSurePull() {
    //创建dialog，插入edittext，用户输入密码并做出检验。
    final AlertDialog.Builder builder = new AlertDialog.Builder(this);
    View view = LayoutInflater.from(this).inflate(R.layout.dialog_is_to_pull, null);
    builder.setTitle("警告：");
    builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    DBConnection.pullSync(USERACCOUNT, USERPASSWORD, NotesListActivity.this);
                }
            }).start();
        }
    });
    builder.setNegativeButton("cancel", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            return;
        }
    });
    final Dialog dialog = builder.setView(view).show();
    dialog.show();
    return false;

}
```

图 29 系统提示代码实现图

对于 pullSync 方法的具体实现流程，我们首先通过数据库操作连接远程的数据库，设计日志信息跟踪载入本地操作的系统状态以便于对功能实现的修改和调试。类似于上传云端功能，系统首先检查用户信息，锁定用户的 note 表和 dat a 表，查询成功后删除本地数据库内容，将云数据库内容逐行遍历，写入本地数据库。最后将系统状态可视化的反馈给用户。具体代码的实现如图 30 所示。

```
public class DBConnection {
    public static boolean pullSync(String user_account, String user_password, Context context){
        try{
            /**
             * 查找用户是否存在
             */
            sqlFind = "SELECT * FROM xiaomi_serve.note ";
            conn = DriverManager.getConnection(url, user, password);
            preparedStatement = conn.prepareStatement(sqlFind);
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                String user_account_registered = resultSet.getString(columnLabel:"account");
                if (user_account_registered.equals(user_account)) {
                    break;
                }
            }
            ContentResolver cr = context.getContentResolver();
            conn = DriverManager.getConnection(url, user, password);
            //先删除本地数据库中两个表的内容
            String selectionNote= Notes.NoteColumns.TYPE+" >= 0";
            String selectionData=Notes.NoteColumns.ID+" >= 0";
            cr.delete(Notes.CONTENT_NOTE_URI, selectionNote, null);
            cr.delete(Notes.CONTENT_DATA_URI, selectionData, null);
            //
            sql = "SELECT * FROM " + user_account + "_note";
            PreparedStatement psDownNoteTable = conn.prepareStatement(sql);
            ResultSet rs = psDownNoteTable.executeQuery();
            //在一行行便利，每行逐列写入本地数据库
            while(rs.next()){…
            Log.d("midd", "data");
            /****同步data和同步Note同理*******/
            sql = "SELECT * FROM " + user_account + "_data";
            PreparedStatement psDownDataTable = conn.prepareStatement(sql);
            rs= psDownDataTable.executeQuery();
            while(rs.next()){…
            Looper.prepare();
            Toast.makeText(context, "下载成功!", Toast.LENGTH_SHORT).show();
            Looper.loop();
        }
        catch (SQLException e) {
            System.out.println(x:"MySQL操作错误");
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
```

图 30 方法 pullSync 具体代码实现图

3.2.3.5 置顶功能

(一) 功能介绍

小米便签应用的置顶功能允许用户将关键便签快速定位到便签列表的顶部，以提高处理紧急或重要事项的效率。用户可以通过简单的操作选择便签并将其置顶，系统会自动将选定的便签移动到列表的首位，并在所有设备上通过云端同步保持这一状态，确保用户在任何地方都能快速访问到最重要的笔记。这一功能通过直观的用户界面和多设备同步机制，增强了便签管理的个性化体验，同时提升了用户在生活和工作中的组织和效率。

(二) 功能详细设计

本小组通过设计 note_edit.xml 文件添加“置顶功能”的响应按钮，实现该功能的界面布局设计。之后通过对数据库中的内容进行修改，在 Notes.java 的 NoteColumn 接口里添加对象，然后在创建数据库中添加列，升级数据库版本。在 NoteEditActivity 类中实现置顶功能的响应函数设计。并在 WorkingNote 中添加置顶功能状态函数。完善 NoteEditActivity 类和 WorkingNote 中的数据库的读写操作实现了将置顶状态保存到数据库的工作。之后，我们对 DBconnection 类进行便签，完成置顶状态的同步。最后为了用户更好的使用过体验，我们在便签列表中添加 icon 图标来标识置顶的便签。置顶功能的类图如图 31 所示。

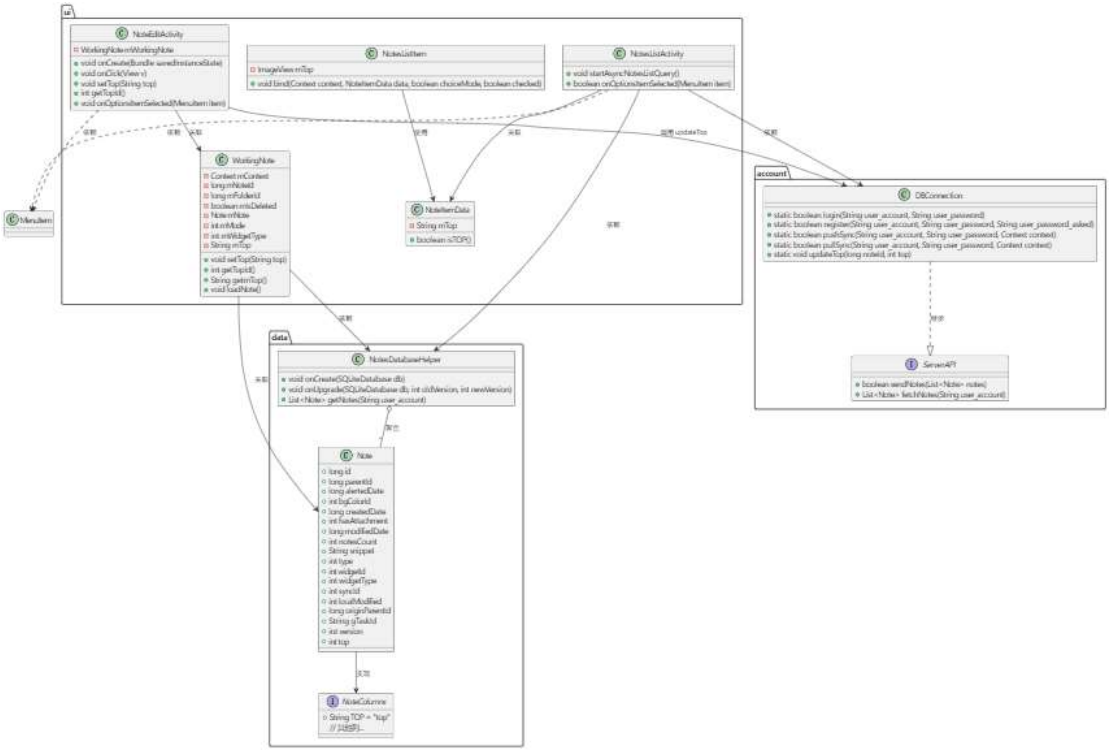


图 31 置顶功能类图

(三) 代码实现

1. 置顶和取消置顶便签功能流程：置顶便签功能涉及多个类的协作，从用户点击置顶按钮到系统更新数据库及界面显示。通过在 note_edit.xml 进行维护，在便签编辑界面添加“置顶”按钮。之后为本地数据库添加“TOP”字段用于表示是否置顶，同时升级数据库版本。一下完成功能具体逻辑的编写：在 NoteEditActivity 中处理置顶按钮点击事件；WorkingNote 类中添加 mTop 参数及相关方法，如图 32 所示；在 Note 类中实现 setTopValue 方法；在 NotesDatabaseHelper 中读取和写入 TOP 值。之后通过读取 TOP 参数，根据 TOP 值对便签进行排序，确保置顶的便签显示在顶部。

```
//置顶
public int getTopId(){
    if(mTop.equals(anObject:"1")){
        return 1;
    }else{
        return 0 ;
    }
}
```

图 32 WorkingNote 维护代码实现图

2. **用户界面反馈：**显示置顶图标和更新菜单项标题。在 NoteItemData 类中定义 isTOP 方法，在 NotesListItem 中绑定置顶图标，在 NoteEditActivity 中更新菜单项标题，在 NotesListItem 的 bind 方法中，根据 top 值显示或隐藏置顶图标，如图 33 所示。在 NoteEditActivity 的 onPrepareOptionsMenu 方法中，根据当前 top 状态更新菜单项标题。

```
if(data.isTOP()){
    mTop.setImageResource(R.drawable.menu_top);
    mTop.setVisibility(View.VISIBLE);
}
else {
    mTop.setVisibility(View.GONE);
}
```

图 33 绑定置顶图标代码实现图

3. **数据库与便签管理：**管理便签数据的存储和获取，确保便签数据的完整性和一致性。这一过程主要涉及两个核心类：NotesDatabaseHelper 类和 DBConnection 类，其中 NotesDatabaseHelper 类负责便签数据的存储、查询及数据库版本升级。查询的具体代码实现如图 34。DBconnection 类负责将本地的置顶属性同步到云数据库中，便于用户数据的同步。在 note 字段添加 top 属性，以便于系统同步置顶便签的状态，保证数据的一致性。

```
/**
 * 加载指定笔记的信息。
 * 从数据库中查询指定ID的笔记的详细信息，并更新当前实例的状态。
 * 注意：此方法不处理查询失败或笔记不存在的情况。
 */
private void loadNote() {
    // 查询指定ID的笔记信息
    Cursor cursor = mContext.getContentResolver().query(
            ContentUris.withAppendedId(Notes.CONTENT_NOTE_URI, mNoteId), NOTE_PROJECTION, null,
            null, null);

    if (cursor != null) {
        // 如果查询结果不为空，尝试读取数据
        if (cursor.moveToFirst()) {
            // 从查询结果中获取笔记的各个属性
            mFolderId = cursor.getLong(NOTE_PARENT_ID_COLUMN);
            mBgColorId = cursor.getInt(NOTE_BG_COLOR_ID_COLUMN);
            mWidgetId = cursor.getInt(NOTE_WIDGET_ID_COLUMN);
            mWidgetType = cursor.getInt(NOTE_WIDGET_TYPE_COLUMN);
            mAlertDate = cursor.getLong(NOTE_ALERTED_DATE_COLUMN);
            mModifiedDate = cursor.getLong(NOTE_MODIFIED_DATE_COLUMN);
            mPasscode = cursor.getString(NOTE_PASSCODE_COLUMN); // 读取 PASSCODE 列的值
            mFont = cursor.getInt(NOTE_FONT_COLUMN); // 读取字体信息
            mTop = cursor.getString(NOTE_TOP_COLUMN);
        }
        // 关闭查询结果集
        cursor.close();
```

图 34 查询 top 属性字段代码实现图

3.2.3.6 九宫格手势密码锁功能

(一) 功能介绍

小米便签应用中的手势密码锁功能为用户提供了一个简便而有效的加密手段，以保护便签内容的安全。通过设置手势密码，用户可以为便签添加额外的安全层，确保只有授权用户才能访问敏感信息。在便签列表中，加密的便签会以特殊标记显示，提醒用户需要解锁才能查看详细内容。这项功能不仅提升了便签的隐私保护。

(二) 功能详细设计

小米便签的九宫格密码锁功能为用户带来了一种直观的便签保护方式。用户可以通过在九宫格上绘制手势来设定密码，这一密码随后存储于数据库并与便签相绑定。当便签处于锁定状态时，用户必须在 UnlockActivity 中输入正确的手势密码以解锁并访问内容。该功能背后的设计由类图(图 35、图 36、图 37、图 38 所示)和顺序图(下图 39、图 40、图 41、图 42、图 43、图 44 所示)简洁地呈现，清晰地展示了功能实现的结构和流程。

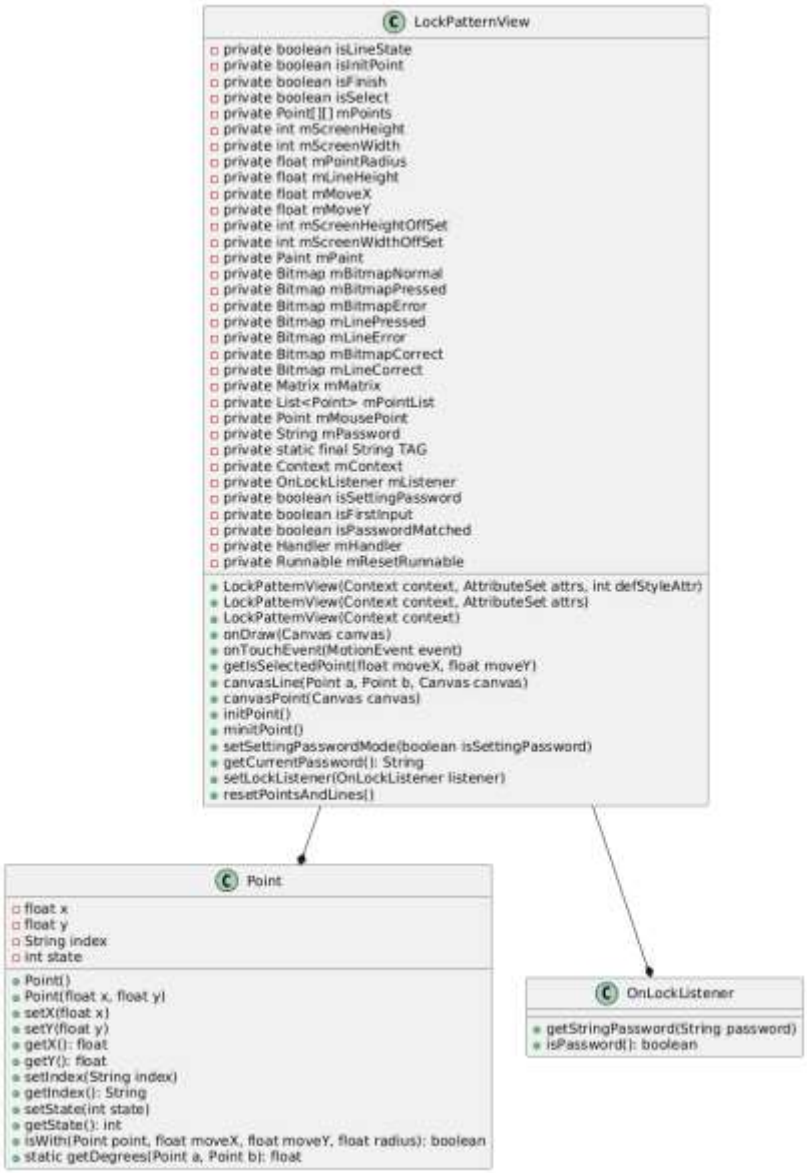


图 35 锁界面类的类图

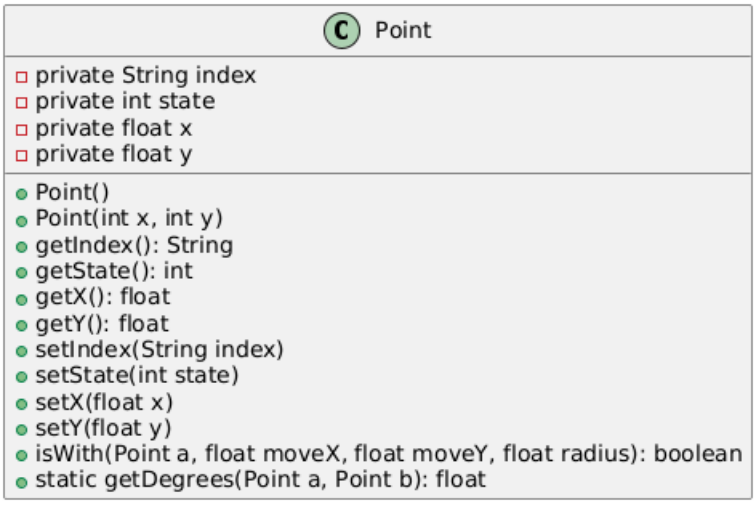


图 36 密码锁点的类图

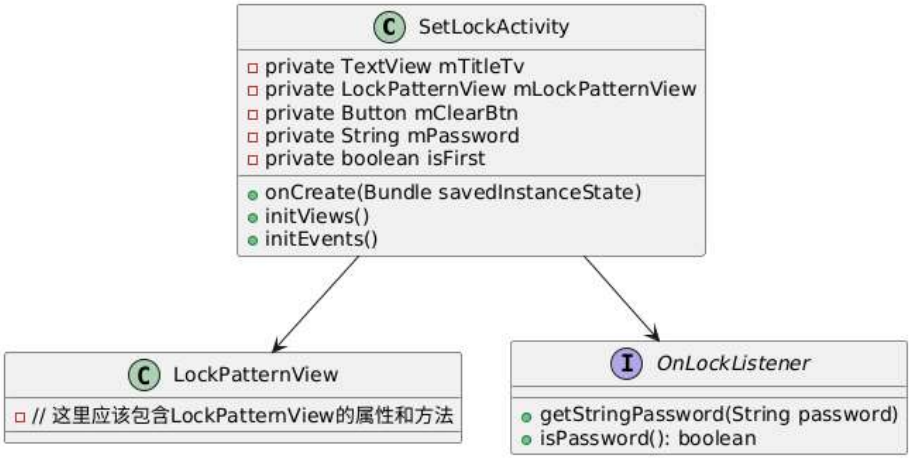


图 37 上锁类的类图

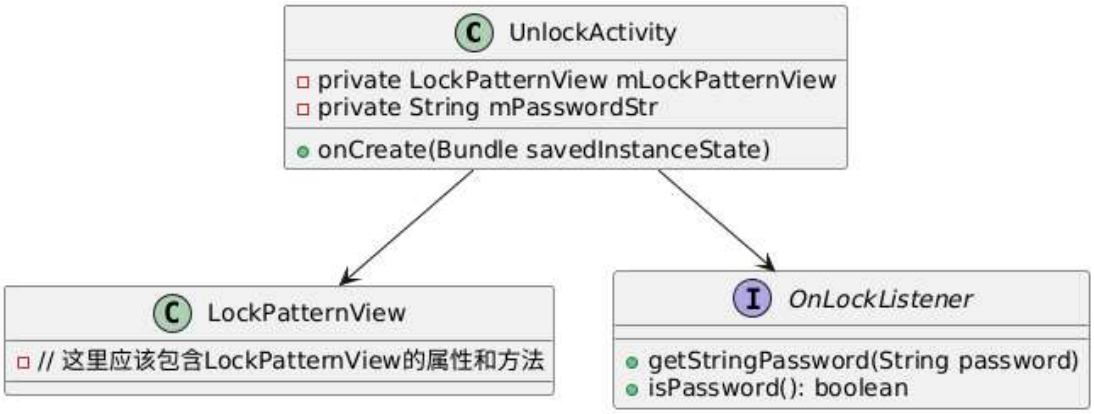


图 38 解锁类的类图

功能设计详细描述

1. **密码设置与存储**：在 SetLockActivity 中，用户首次设置手势密码。系统会提示用户绘制手势，并在确认后存储这个手势密码。密码以一种安全的方式存储在数据库中，确保即使数据库被非法访问，密码信息也不会被泄露。

2. **便签锁定与解锁**：当用户选择锁定便签时，系统会在数据库中标记该便签为已加密。在用户尝试访问这个便签时，系统会检查便签的加密状态。如果便签已加密，用户将被引导至 UnlockActivity 输入手势密码。只有当输入的密码与存储的密码匹配时，用户才能解锁并访问便签。

3. **用户界面交互**：在 UnlockActivity 中，用户通过绘制手势来解锁便签。系统会实时反馈用户的输入是否正确。如果密码正确，用户将被引导至 NoteEditActivity 继续编辑便签；如果错误，系统会提示错误并允许用户重新尝试。

4. **密码管理**：在 NoteEditActivity 中，用户可以管理便签的加密状态。如果便签未加密，用户可以选择设置密码；如果便签已加密，用户可以选择删除密码。

这些操作都会即时更新数据库中的便签加密状态。

5. **安全性与用户体验**：整个密码锁功能设计注重安全性和用户体验的平衡。通过直观的图形界面和清晰的操作反馈，用户可以轻松地管理便签的加密状态，同时确保他们的数据安全。

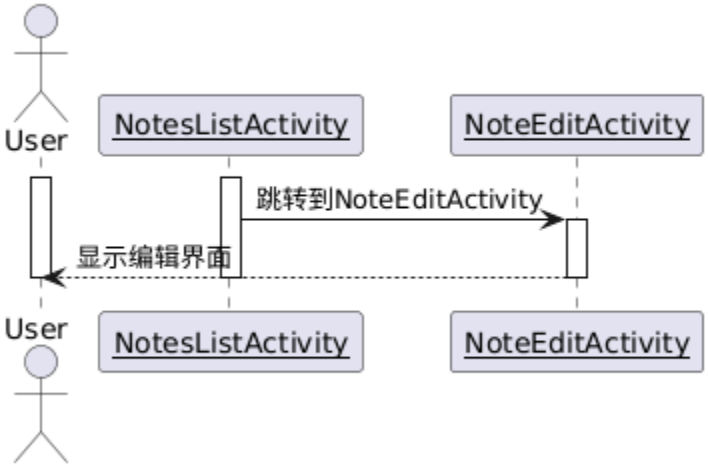


图 39 便签未加密时的直接编辑功能顺序图

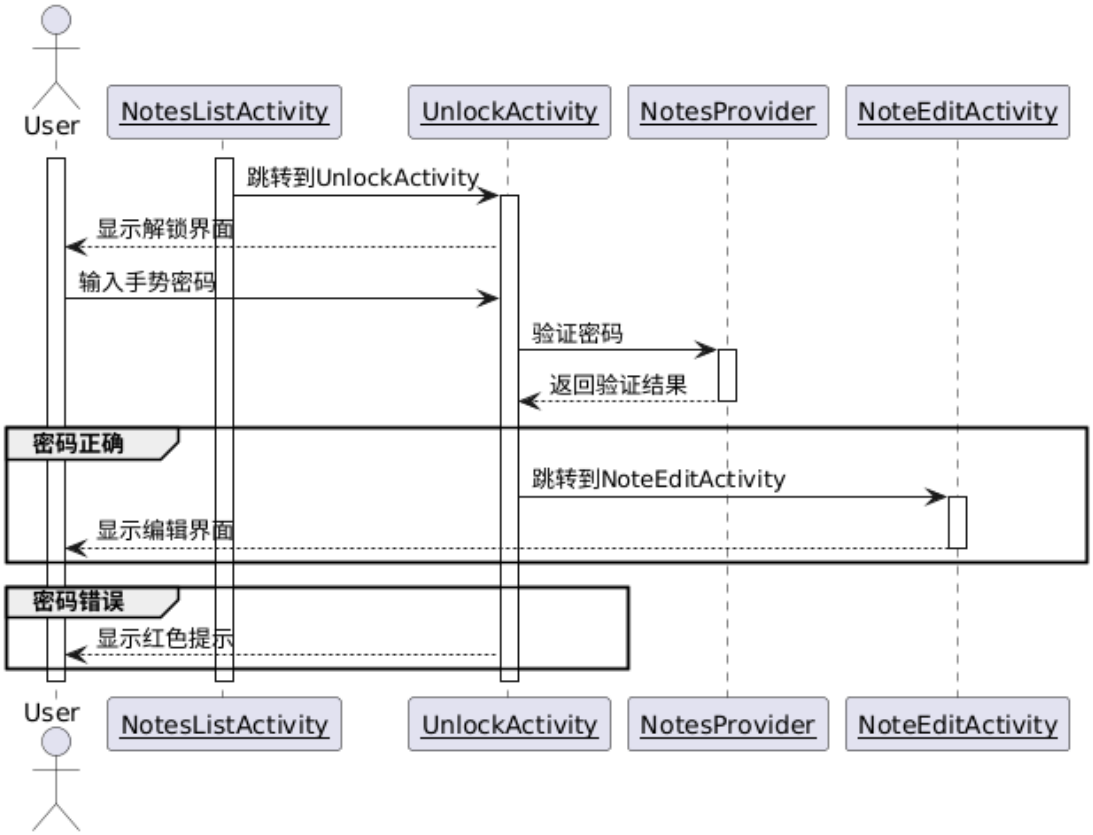


图 40 便签已加密时的解锁功能顺序图

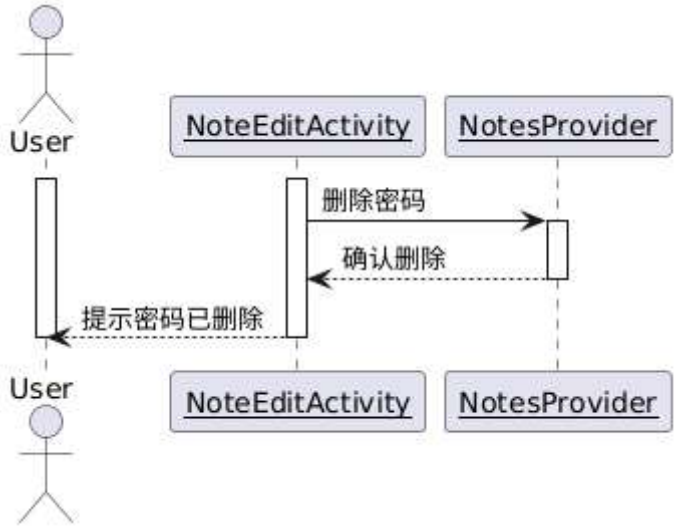


图 41 删除密码功能顺序图

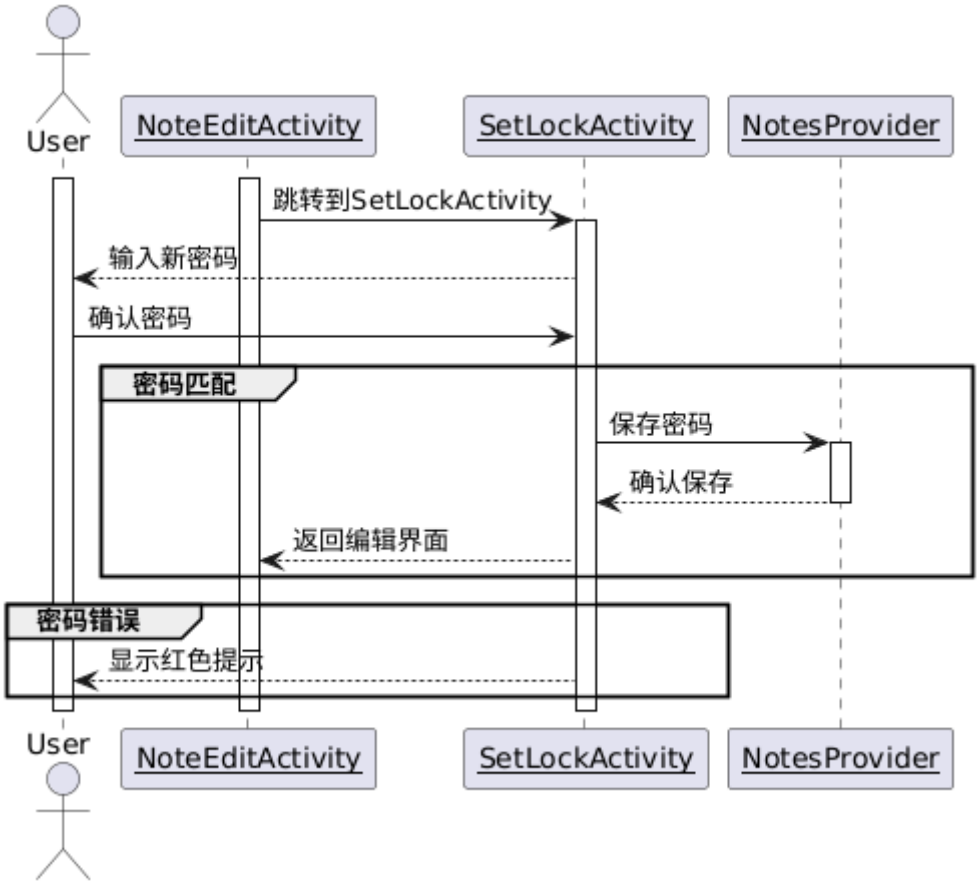


图 42 设置密码功能顺序图

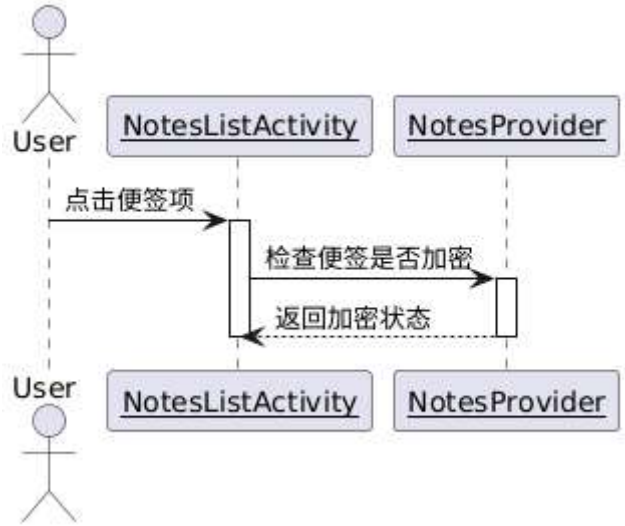


图 43 点击便签项并检查加密状态功能顺序图

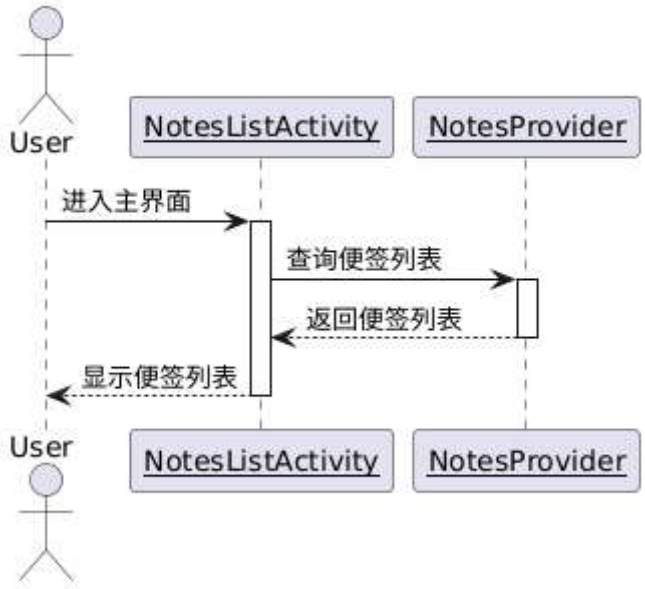


图 44 进入主界面并显示便签列表顺序图

(三) 代码实现

小米便签应用引入了九宫格密码锁功能，允许用户为便签设置手势密码，从而增强了便签内容的安全性。在用户首次进入便签主界面时，系统会自动检查每个便签项的加密状态。未加密的便签将正常显示，而已加密的便签则会隐藏原标题，并显示“已加密”字样，同时配以锁图标。当用户尝试访问加密便签时，系统将引导用户进入解锁界面，通过手势密码验证用户身份。

在接下来的部分，我们将详细介绍这一功能的代码实现过程，包括如何检测便签的加密状态、如何引导用户设置和验证手势密码，以及如何确保便签内容的安全存储和访问。

1. **便签加密状态检查**：在 NotesListActivity 中，系统查询便签列表并检查每个便签项的加密状态。通过 NoteItemData 类中的 hasPasscode()方法(如图 45 所示)，可以判断便签是否设置了密码。

```
/**判断是否已经设置密码*/
public boolean hasPasscode() {
    return !TextUtils.isEmpty(mPasscode); // 使用 TextUtils.isEmpty 避免 null 检查
}
```

图 45 检查锁方法图

2. **便签访问与解锁流程**：用户点击便签项时，如果便签已加密，NotesListActivity 将启动 UnlockActivity。在 UnlockActivity 中，用户输入手势密码，通过 OnLockListener 接口的 isPassword()方法验证密码正确性(如图 46 所示)。

```
@Override
public boolean isPassword() {
    if (mPasswordStr.equals(password)) {
        Toast.makeText(UnlockActivity.this, "密码正确", Toast.LENGTH_SHORT).show();
        Intent resultIntent = new Intent();
        resultIntent.putExtra(Intent.EXTRA_UID, noteId); // 传递 noteId
        setResult(RESULT_OK, resultIntent); // 设置返回结果为 RESULT_OK
        UnlockActivity.this.finish(); // 结束 UnlockActivity
        return true; // 返回 true，表示密码验证成功
    } else {
        Toast.makeText(UnlockActivity.this, "密码不正确", Toast.LENGTH_SHORT).show();
        setResult(RESULT_CANCELED); // 设置返回结果为 RESULT_CANCELED
        UnlockActivity.this.finish(); // 结束 UnlockActivity
        return false; // 返回 false，表示密码验证失败
    }
}
```

图 46 检验密码正确性方法

3. **密码设置与删除**：在 NoteEditActivity 中，用户可以通过菜单项设置或删除便签的手势密码。点击“设置密码”时，用户被引导至 SetLockActivity，在这里用户创建新的手势密码并确认，其逻辑如图 47 所示。密码设置成功后，通过 WorkingNote 类的 setPasscode()方法更新数据库中的便签信息。

```
@Override
public void getStringPassword(String password) {
    if (isFirst) {
        mPassword = password;
        mTitleTv.setText("再次输入手势密码");
        isFirst = false;
        mClearBtn.setVisibility(View.VISIBLE);
    } else {
        if (password.equals(mPassword)) {
            Intent pre = getIntent();
            //将密码写入数据库
            long noteId = pre.getLongExtra(Intent.EXTRA_UID, 0);
            WorkingNote mWorkingNote = WorkingNote.load(SetLockActivity.this, noteId);
            mWorkingNote.setPasscode(password);
            boolean saved = mWorkingNote.saveNote();//保存密码
            Intent intent = new Intent(SetLockActivity.this, NoteEditActivity.class);
            intent.setAction(Intent.ACTION_VIEW);
            intent.putExtra("lock", 0);
            intent.putExtra(Intent.EXTRA_UID, noteId);
            startActivity(intent);
            SetLockActivity.this.finish();
        } else {
            Toast.makeText(SetLockActivity.this, "两次密码不一致，请重新设置", Toast.LENGTH_SHORT).show();
            mPassword = "";
            mTitleTv.setText("设置手势密码");
            isFirst = true;
            mClearBtn.setVisibility(View.GONE);
        }
    }
}
```

图 47 设置密码逻辑

4. **数据库与密码管理**：NotesDatabaseHelper 类负责管理便签数据的存储和版本升级。在数据库升级过程中，通过 upgradeToV5()方法添加了 PASSCODE 列，用于存储便签的手势密码，如下图 48 所示。

```
private void upgradeToV5(SQLiteDatabase db) {
    db.execSQL("ALTER TABLE " + TABLE.NOTE + " ADD COLUMN " + NoteColumns.PASSCODE + " TEXT NOT NULL DEFAULT ''");
}
```

图 48 更新数据库

5. **用户界面反馈**：在 NotesListItem 中，根据便签的加密状态，通过设置锁图标和“已加密”文本，为用户提供直观的加密状态反馈，如图 49 所示。

```
// 判断是否有锁，并设置锁图标和文本
if (data.hasLock()) {
    mLock.setImageResource(R.drawable.lock);
    mLock.setVisibility(View.VISIBLE);
    String text = "已加密";
    mTitle.setText(text);
} else {
    mLock.setVisibility(View.GONE);
}
```

图 49 便签上锁

3.2.3.7 便签搜索并跳转功能

(一) 功能介绍

小米便签应用的便签搜索功能是一项便捷工具，允许用户通过关键词迅速查

找便签内容。用户在主界面激活搜索后，输入关键字即可获取匹配结果，结果以列表形式展示，支持直接跳转至相应便签进行查看或编辑。这一流程不仅提高了用户查找便签的效率，也增强了应用的交互性和实用性。

(二) 功能详细设计

小米便签的便签搜索功能，为用户提供了一种快速检索便签内容的方式。通过关键词搜索，用户能够迅速定位所需信息。这一功能通过简洁的用户界面和高效的后台处理，保障了搜索的便捷与准确。其设计细节，可通过顺序图(图 50、图 51、图 52、图 53)和类图(图 54、图 55)得到清晰展示。

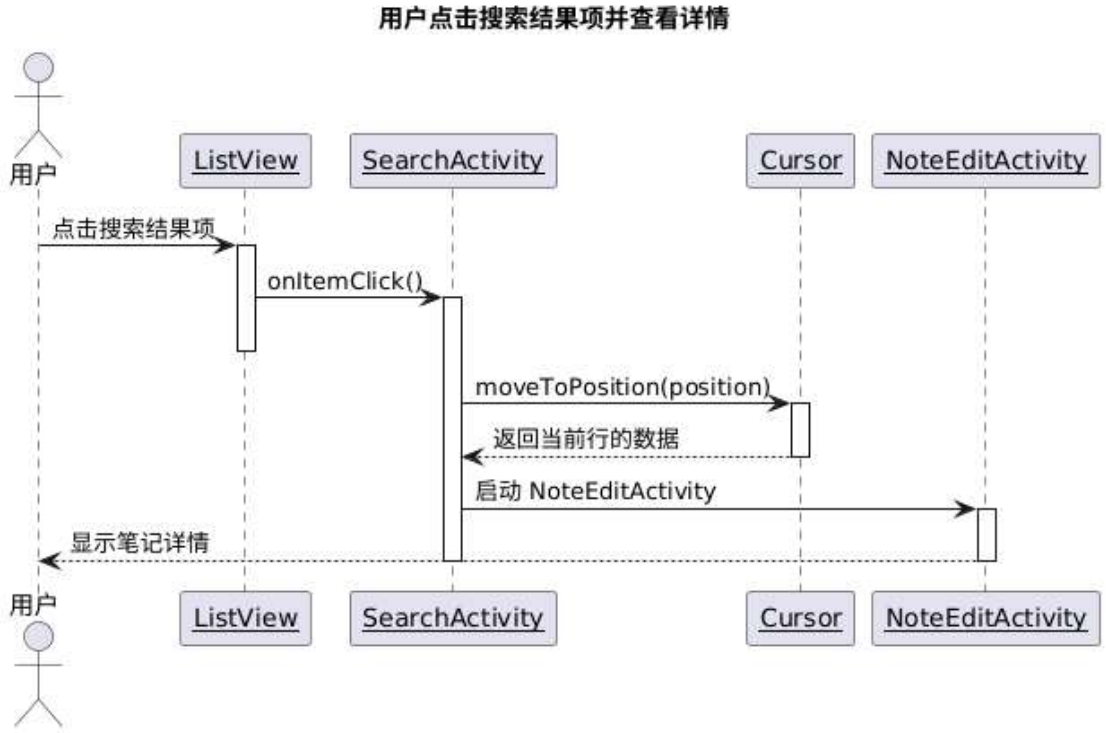


图 50 点击搜索结果项功能的顺序图

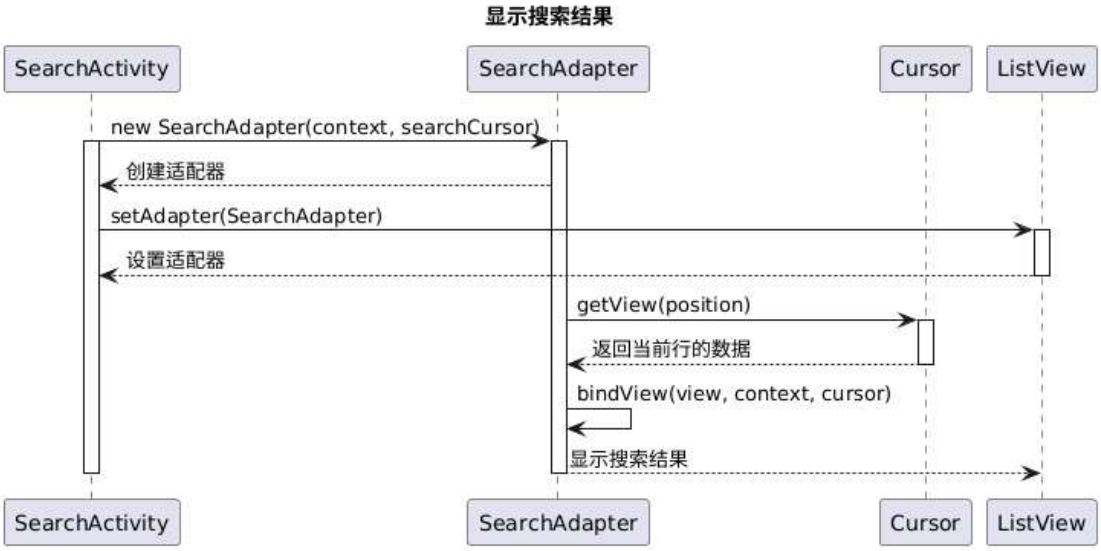


图 51 显示搜索结果功能顺序图

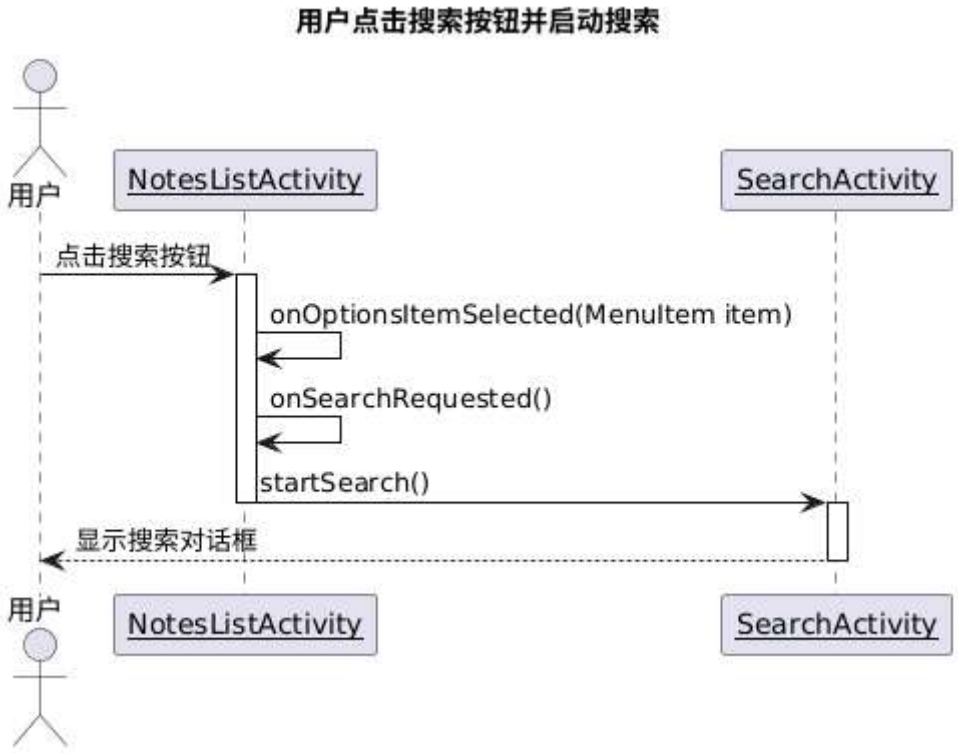


图 52 用户点击搜索按钮并启动搜索功能顺序图

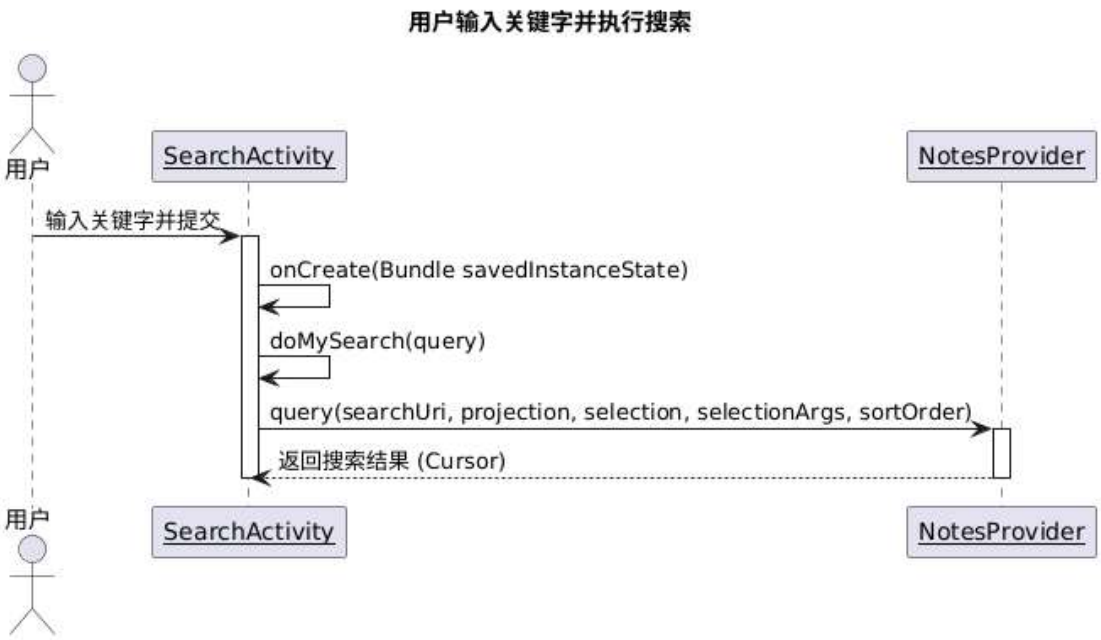


图 53 用户输入关键字并执行搜索功能顺序图

功能设计详细描述

1. **用户界面与交互**：在 NotesListActivity 中，用户通过点击搜索按钮触发搜索流程。系统响应用户操作，显示搜索对话框，用户输入搜索关键词后，提交以启动搜索。

2. **搜索请求处理**：SearchActivity 接收到搜索请求后，通过 doMySearch(String query)方法执行搜索操作。该方法构建查询语句，利用 ContentResolver 与 NotesProvider 交互，检索包含关键词的便签项。

3. **搜索结果展示**：搜索结果以 Cursor 形式返回，并传递给 SearchAdapter。SearchAdapter 负责将搜索结果绑定到 ListView，为用户提供清晰的搜索结果列表。每个结果项都突出显示搜索关键词，便于用户识别。

4. **结果项交互**：用户点击搜索结果中的某个便签项，系统通过 NoteEditActivity 展示该便签的详细内容。这一过程中，便签的 ID 作为参数传递，确保用户能够直接访问到特定的便签。

5. **数据持久化与恢复**：搜索功能不仅关注当前的搜索操作，还考虑到用户可能在搜索后对便签进行的编辑。因此，搜索结果中的便签信息能够被持久化存储，并在用户返回时恢复，确保搜索体验的连贯性。

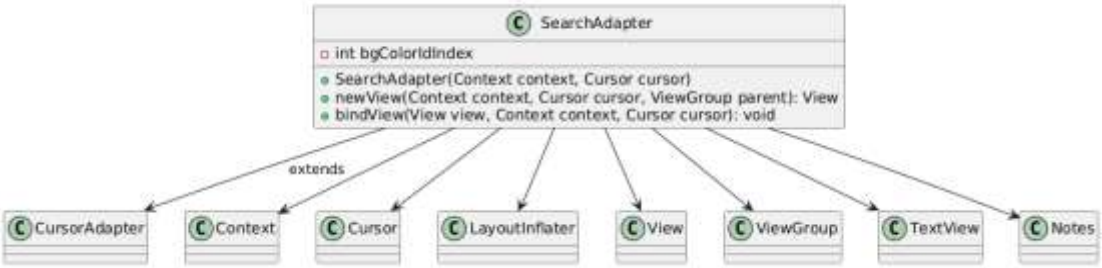


图 54 SearchAdapter 类的类图

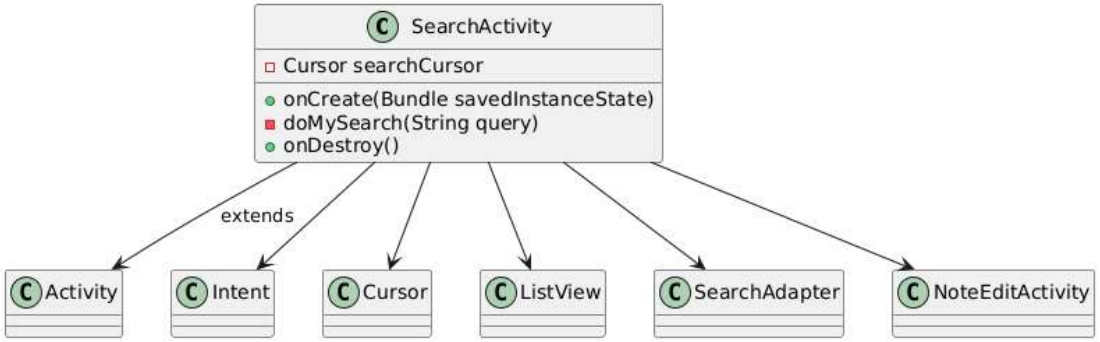


图 55 SearchActivity 类的类图

(三) 代码实现

小米便签的搜索功能通过在 NotesListActivity 中集成搜索入口，引导用户进入 SearchActivity，该活动专门处理搜索请求并展示结果。搜索结果通过自定义的 SearchAdapter 展示在 ListView 中，为用户提供了一个直观且易于操作的搜索体验。

代码实现细节：

1. **搜索活动配置：**在 AndroidManifest.xml 中，SearchActivity 被明确配置为处理搜索请求(如图 56 所示)。通过<intent-filter>标签，它声明了对 android.intent.action.SEARCH 的响应能力，确保当搜索动作发生时，系统能够识别并启动该活动。此外，一个<meta-data>标签指向 searchable.xml，后者详细定义了搜索的元数据，包括搜索建议的提供者，这使得搜索功能更加丰富和用户友好。

```
<!--搜索活动-->
<activity
    android:name=".ui.SearchActivity"
    android:configChanges="keyboardHidden|orientation|screenSize"
    android:launchMode="singleTop"
    android:theme="@style/NoteTheme"
    android:uiOptions="splitActionBarWhenNarrow"
    android:windowSoftInputMode="adjustPan">
    <meta-data android:name="android.app.searchable"
        android:resource="@xml/searchable"/>
    <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>

</activity>
```

图 56 SearchActivity 的配置

2. **触发搜索请求**：在 NotesListActivity 中，用户界面提供了一个搜索菜单项。当用户选择这个菜单项时，onOptionsItemSelected 方法会被触发，进而调用 onSearchRequested 方法(如图 57 所示)。这个方法是搜索流程的起点，它调用 startSearch 函数来激活系统的搜索对话框，允许用户输入搜索关键字，从而开始搜索过程。

```
/**
 * 处理搜索请求。
 *
 * @return 总是返回true。
 */
@Override
public boolean onSearchRequested() {

    startSearch(null, false, null /* appData */, false);
    return true;
}
```

图 57 搜索请求响应

3. **处理搜索请求**：SearchActivity 的 onCreate 方法是处理搜索请求的核心(如图 58 所示)。它首先检查接收到的 Intent 是否为搜索请求。如果是，它将从 Intent 中提取搜索关键字，并调用 doMySearch 方法来执行搜索操作。这一步骤是搜索流程的关键，它确保了用户输入的搜索词能够被正确处理。

```
// 获取搜索请求的 Intent
Intent intent = getIntent();
if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
    String query = intent.getStringExtra(SearchManager.QUERY); // 获取搜索查询
    doMySearch(query); // 执行搜索操作
}
```

图 58 搜索请求处理

4. **执行搜索操作**：doMySearch 方法在 SearchActivity 中负责执行实际的搜索操作(如图 59 所示)。它构建了一个查询语句，使用 ContentResolver 与 NotesPro

vider 交互，筛选出 snippet 字段中包含搜索关键字的便签项。查询结果存储在 Cursor 对象中，这个对象包含了便签的相关信息，为后续的结果展示提供了数据基础。

```
private void doMySearch(String query) {
    // 定义搜索时使用的URI和查询参数
    Uri searchUri = Notes.CONTENT_NOTE_URI; // 使用 Notes.CONTENT_NOTE_URI
     String[] projection = new String[]{
            Notes.NoteColumns.ID,
            Notes.NoteColumns.BG_COLOR_ID,
            Notes.NoteColumns.SNIPPET
    };
    String selection = Notes.NoteColumns.SNIPPET + " LIKE ?";
    String[] selectionArgs = new String[]{"%" + query + "%"}; // 使用通配符 %
    String sortOrder = Notes.NoteColumns.TYPE + " DESC, " + Notes.NoteColumns.MODIFIED_DATE + " DESC"; // 排序条件

    // 使用 NotesProvider 获取搜索结果
    searchCursor = getContentResolver().query(searchUri, projection, selection, selectionArgs, sortOrder);
```

图 59 搜索操作的执行

5. **展示搜索结果**：搜索结果的展示是通过 SearchAdapter 实现的，它继承自 CursorAdapter。SearchAdapter 负责将搜索结果的 Cursor 数据转换为用户界面上的视图(如图 60 所示)。每个搜索结果项都显示便签的摘要（snippet），并根据便签的背景颜色设置背景，使得搜索结果既信息丰富又视觉上吸引人。

```
@Override
public void bindView(View view, Context context, Cursor cursor) {

    TextView snippetTextView = view.findViewById(R.id.text_view_snippet);

    String snippet = cursor.getString(cursor.getColumnIndex(Notes.NoteColumns.SNIPPET));

    snippetTextView.setText(snippet);

    if (bgColorIdIndex != -1 && !cursor.isNull(bgColorIdIndex)) {
        int bgColorId = cursor.getInt(bgColorIdIndex);
        switch (bgColorId) {
```

图 60 搜索结果的展示

6. **用户交互**：用户与搜索结果的交互是通过点击操作实现的(如图 61 所示)。当用户点击搜索结果中的某个便签项时，ListView 的 OnItemClickListener 会被触发。它获取点击项的便签 ID，然后启动 NoteEditActivity，并传递便签 ID 作为参数。这样，用户就可以直接查看或编辑他们感兴趣的便签。

```
// 设置 ListView 的 OnItemClickListener
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        if (searchCursor != null && searchCursor.moveToPosition(position)) {
            long noteId = searchCursor.getLong(searchCursor.getColumnIndex(Notes.NoteColumns.ID));
            Intent intent = new Intent(SearchActivity.this, NoteEditActivity.class);
            intent.setAction(Intent.ACTION_VIEW);
            intent.putExtra(Intent.EXTRA_UID, noteId);
            startActivity(intent);
        }
    }
});
```

图 61 搜索结果点击跳转

3.2.3.8 便签更改字体功能

(一) 功能介绍

小米便签应用允许用户个性化选择字体，增强便签的视觉体验。默认字体为宋体，用户可在编辑界面通过菜单轻松切换至其他字体。所选字体将保存至数据库，确保每次打开便签时，都能呈现用户偏好的字体样式，让便签管理更加符合个人风格。

(二) 功能详细设计

小米便签应用的字体更改功能，使用户能够根据个人喜好定制便签的视觉风格。这项设计不仅优化了用户体验，也使得便签内容的呈现更加个性化。通过直观的操作流程和简洁的界面设计，用户可以轻松切换字体，实现个性化展示。功能设计的类图(如下图 62 所示)和顺序图(如下图 63 所示)提供了清晰的设计概览，指导了字体更改的具体实现过程。以下是该功能设计的详细阐释：

1. **用户界面交互**：在便签编辑界面，用户可以通过点击右上角的菜单按钮，选择“更改字体”选项，来触发字体选择对话框的显示。

2. **字体选择对话框**：弹出的对话框提供了多种字体样式供用户选择。用户可以根据自己的喜好，浏览并选择不同的字体样式。

3. **字体更改确认**：用户在选择完想要的字体后，点击确认按钮，便签应用会将所选字体的标识信息更新到 WorkingNote 类的 mFont 变量中。

4. **字体信息保存**：更改后的字体信息通过 WorkingNote 类的 saveNote()方法保存到数据库中，确保用户的选择能够被持久化存储。

5. **即时反馈**：在用户确认字体更改后，NoteEditActivity 会立即在编辑器中应用新的字体样式，为用户提供即时的视觉反馈。

6. **数据同步**：当用户完成编辑并退出时，NoteEditActivity 会再次调用 saveNote()方法，确保所有更改，包括字体样式，都已同步到数据库中。

7. **再次打开便签**：当用户在未来再次打开这个便签时，应用将自动从数据库中读取 mFont 列的值，加载对应的字体，并将其呈现在便签文本中。

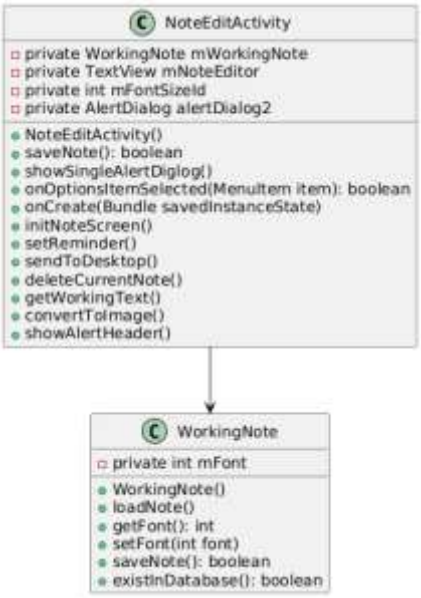


图 62 修改字体相关类图

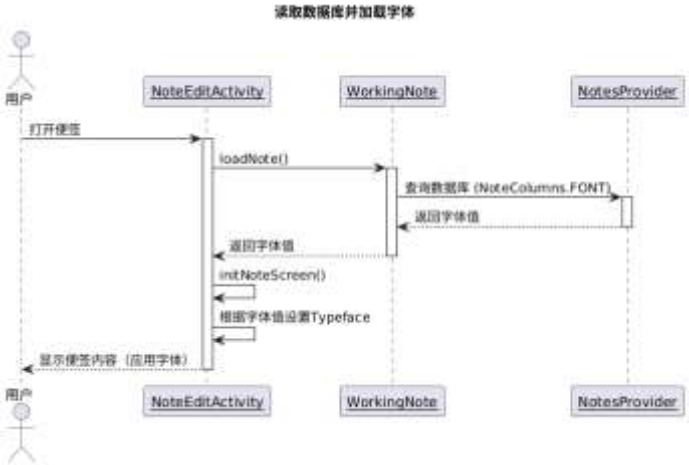


图 63 读取数据库并加载字体功能顺序图

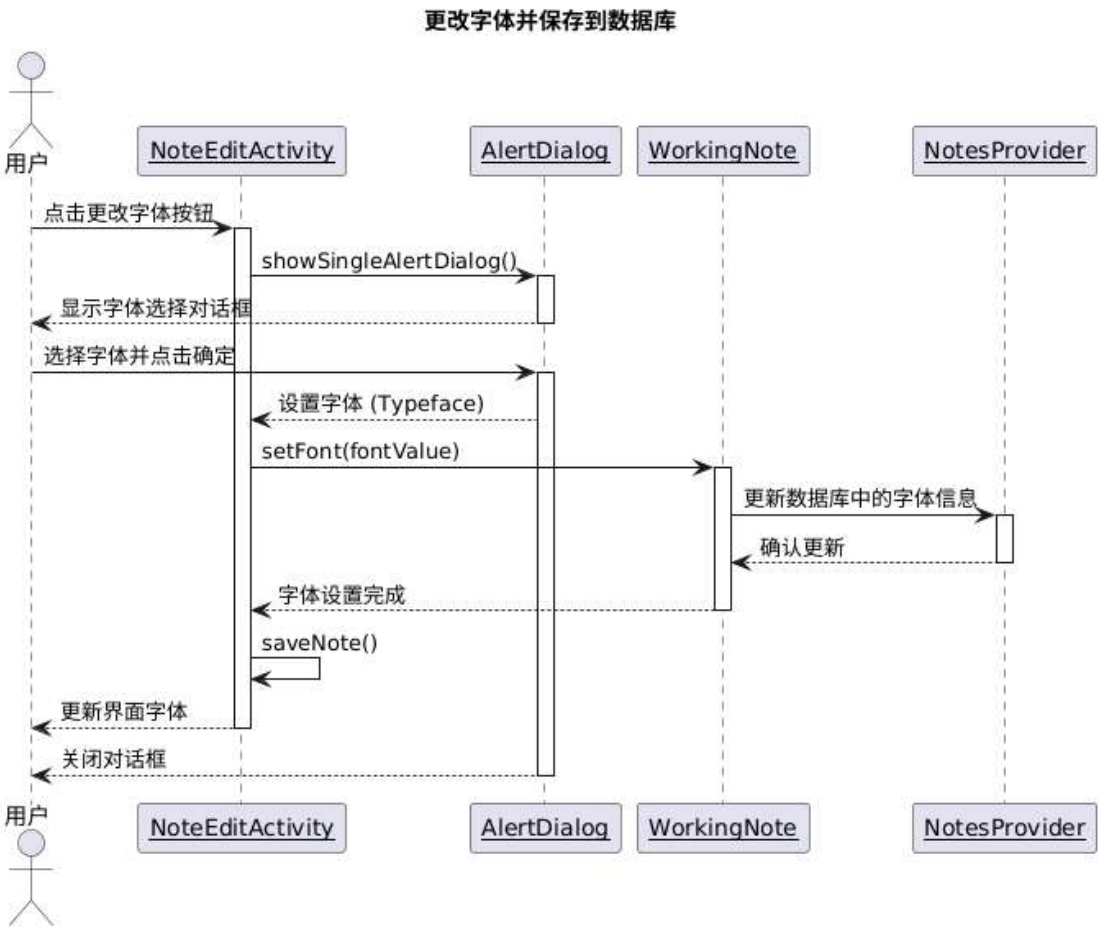


图 64 更改字体并保存到数据库功能顺序图

(三) 代码实现

小米便签应用中的更改字体功能是一项提升用户体验的重要特性，它允许用户根据自己的偏好选择不同的字体样式，使得便签内容的展示更加个性化和多样化。该功能通过精心设计的界面交互和后台逻辑，确保了用户操作的便捷性和数据的持久化存储。

代码实现细节：

1. **字体选择与应用：**在便签编辑界面，用户可以通过点击右上角菜单中的“更改字体”按钮，触发字体选择对话框的显示。在字体选择对话框中，用户选择一个字体后，NoteEditActivity 会立即在编辑器中应用新的字体样式，为用户提供即时的视觉反馈。这一过程通过 Typeface.createFromAsset 方法实现，根据用户选择的字体，动态加载相应的字体文件并应用到 mNoteEditor（便签编辑器的 Text View，如图 65 所示）上。

```
alertDialog2 = alertBuilder.create();
alertDialog2.show();
```

图 65 小米便签字体显示

2. **字体信息的存储**：一旦用户确认了字体选择，NoteEditActivity 会调用 WorkingNote 类的 setFont(int font)方法，将用户选择的字体样式标识更新到 mFont 变量中。这个标识随后被保存到数据库中，确保用户的选择能够被持久化存储(如图 66 所示)。

```
/**
 * 保存笔记。
 * 更新笔记内容并保存。
 *
 * @return 是否成功保存笔记。
 */
private boolean saveNote() {
    getWorkingText();
    mWorkingNote.setFont(mWorkingNote.getFont()); // 根据实际获取字体选择的逻辑获取字体值

    boolean saved = mWorkingNote.saveNote();
    if (saved) {
        // 设置结果为成功，以便外部调用者知道保存操作的状态
        setResult(RESULT_OK);
    }
    return saved;
}
```

图 66 存储字体信息

3. **数据持久化**：WorkingNote 类的 setFont(int font)方法(如图 67 所示)中，除了更新 mFont 变量外，还会调用 mNote.setNoteValue(NoteColumns.FONT, String.valueOf(font))来更新数据库中的字体信息。这样，无论用户何时再次打开这个便签，应用都能从数据库中读取并应用用户之前选择的字体。

```
public void setFont(int font) {
    if (this.mFont != font) {
        this.mFont = font;
        mNote.setNoteValue(NoteColumns.FONT, String.valueOf(font)); // 更新数据库中的字体信息
    }
}
```

图 67 更新数据库字体信息

4. **初始化与恢复字体设置**：在 NoteEditActivity 的 onCreate 方法中，如果检测到是新便签，会设置默认字体为宋体（字体标识为 5）。在 initNoteScreen 方法中，根据从数据库加载的字体信息(如图 68 所示)，通过 Typeface.createFromAsset 方法加载相应的字体文件，并应用到便签编辑器上，确保用户在打开便签时看到的是他们之前选择的字体样式。

```
// 设置字体
int fontValue = mWorkingNote.getFont();
Typeface typeface = null;
switch (fontValue) {
    case 0:
        typeface = Typeface.createFromAsset(getAssets(), "font/caiyun.ttf");
        break;
    case 1:
        typeface = Typeface.createFromAsset(getAssets(), "font/heiti.ttf");
        break;
    case 2:
        typeface = Typeface.createFromAsset(getAssets(), "font/huati.ttf");
        break;
    case 3:
        typeface = Typeface.createFromAsset(getAssets(), "font/kaiti.ttf");
        break;
    case 4:
        typeface = Typeface.createFromAsset(getAssets(), "font/lishu.ttf");
        break;
    case 5:
        typeface = Typeface.createFromAsset(getAssets(), "font/songti.ttf");
        break;
    case 6:
        typeface = Typeface.createFromAsset(getAssets(), "font/xingkai.ttf");
        break;
    default:
        typeface = Typeface.createFromAsset(getAssets(), "font/songti.ttf"); // 默认字体
        break;
}
mNoteEditor.setTypeface(typeface);
// 显示提醒头部信息（当前禁用，因DateTimePicker未准备好）
showAlertHeader();
convertToImage();
```

图 68 根据字体值加载对应字体

5. **字体选择的确认与保存**：在字体选择对话框的“确定”按钮点击事件中，会调用 saveNote()方法，这不仅保存了便签的内容，也确保了字体选择的持久化。如果保存成功，会设置结果码为 RESULT_OK，以便外部调用者知道保存操作的状态(如图 69 所示)。

```
alertBuilder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int i) {
        alertDialog2.dismiss();
        saveNote(); // 保存字体选择到数据库

    }
});
alertBuilder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int i) {
        alertDialog2.dismiss();
    }
});
```

图 69 用户更改字体的响应

通过这样的设计，小米便签的更改字体功能实现了用户界面的直观操作与后台数据的同步更新，确保了用户个性化设置的持久性和应用的响应性。

3.2.3.9 便签开场动画功能

(一) 功能介绍

小米便签应用中的开场动画功能为用户带来了一个直观且具有吸引力的启动体验。这个动画在应用启动时即刻呈现，以简洁的视觉效果快速吸引用户的注意力。动画结束后，用户将直接进入主界面。

(二) 功能详细设计

小米便签应用中的开场动画功能，以其精心设计的流程和类结构，致力于为用户提供一个既流畅又充满吸引力的启动体验。这个功能由 SplashActivity 类主导，它负责管理动画的展示以及随后的过渡至应用的主界面，从而确保了应用启动时的视觉效果。通过类图和顺序图（如图 70、图 71 所示），我们可以清晰地了解到开场动画的具体实现细节和流程，这些图表共同描绘了这一功能背后的技术架构。

1. **SplashActivity 的初始化**：当用户点击小米便签图标启动应用时，首先初始化 SplashActivity。这个活动负责展示开场动画并设置布局文件 R.layout.activity_splash。

2. **动画效果的实现**：在 SplashActivity 中，通过获取 TextView 引用并创建透明度动画（AlphaAnimation），应用将动画应用到 TextView 上，实现开场动画效果。

3. **延迟跳转**：为了确保动画效果的完整展示，SplashActivity 使用 Handler 类设置了一个 3 秒的延迟，之后触发跳转到应用的主界面。

4. **Intent 的创建与传递**：在延迟结束后，SplashActivity 创建一个 Intent，指定从 SplashActivity 跳转到 NotesListActivity，这是用户进入应用后看到的第一个界面，显示便签列表。

5. **SplashActivity 的销毁**：在跳转完成后，SplashActivity 被销毁，确保不会在应用的后台占用资源。

6. **NotesListActivity 的启动**：NotesListActivity 接收到 Intent 后启动，显示主界面和便签列表，用户可以开始使用便签应用。

图 70 小米便签开场动画顺序图

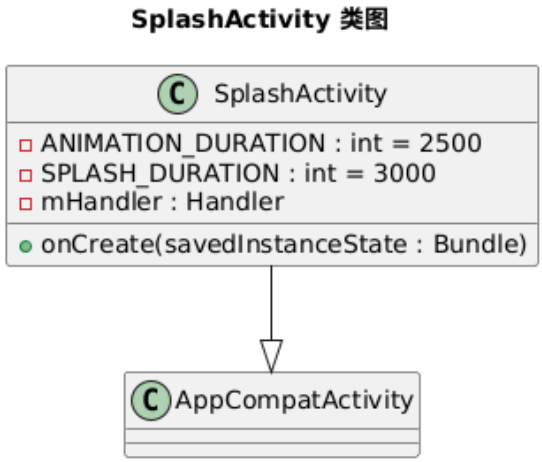


图 71 小米便签开场动画类图

(三) 代码实现

1. **配置文件**：在 AndroidManifest.xml 中，SplashActivity 被配置为应用的启动活动(如图 72 所示)，通过<intent-filter>标签指定了 MAIN action 和 LAUNCHER category，确保这是应用启动时首先显示的活动。

```
<!-- 动画-->
    <activity
        android:name=".ui.SplashActivity"
        android:configChanges="orientation|keyboardHidden|screenSize"
        android:theme="@style/Theme.Notes1.Fullscreen">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

图 72 开场动画配置

2. **动画布局**：SplashActivity 继承自 AppCompatActivity，使用 AndroidX 库以确保兼容性。在 onCreate 方法中，首先通过 setContentView 设置布局文件 activity_splash。接着，获取布局中的 TextView 控件引用。具体实现如图 73 所示。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_splash);

    // 获取 TextView 的引用
    TextView textView = findViewById(R.id.fullscreen_content);
```

图 73 开场动画视图绑定

3. **动画参数设置**：创建一个 AlphaAnimation 对象，实现从完全透明（0f）到不透明的渐变效果，动画持续时间设定为 2500 毫秒。通过调用 startAnimation 方法，将这个透明度变化动画应用到 TextView 上，从而启动动画。具体代码实现如图 74 所示。

```
// 创建透明度动画对象，从完全透明到不透明
AlphaAnimation alphaAnimation = new AlphaAnimation(0f, 1f);
alphaAnimation.setDuration(ANIMATION_DURATION); // 设置动画持续时间
```

图 74 开场动画参数设置

4. **延迟跳转与界面跳转**：为了在动画结束后提供平滑的过渡到应用的主界面，使用 Handler 的 postDelayed 方法设置一个 3000 毫秒（SPLASH_DURATION）的延迟。当延迟结束时，执行一个 Runnable 任务。在这个任务中，创建一个 Intent 对象，指定从 SplashActivity 跳转到 NotesListActivity。然后调用 startActivity 启动这个 Intent，并调用 finish 方法来销毁 SplashActivity，完成从欢迎界面到便签列表界面的跳转。具体代码实现如图 75 所示。

```
// 当计时结束时，跳转至 NotesListActivity
mHandler.postDelayed(new Runnable() {
    @Override
    public void run() {
        Intent intent = new Intent(SplashActivity.this, BeginViewActivity.class);
        startActivity(intent);
        finish(); // 销毁欢迎页
    }
}, SPLASH_DURATION);
```

图 75 开场动画跳转

3.2.3.10 便签插入图片功能

(一) 功能介绍

小米便签应用新推出的图片插入功能，允许用户在便签中添加图片，从而让便签内容更加丰富多彩。用户只需在编辑页面点击“插入图片”按钮，即可从媒体库中选择图片并将其嵌入便签，这一过程简单直观，极大地提升了便签的信息表达能力。

(二) 功能详细设计

小米便签插入图片允许用户在编辑便签时，通过简单的操作将图片嵌入便签内容。设计上，该功能通过一个清晰的用户界面引导用户授权访问媒体库，随后通过一个简洁的对话框让用户选择图片。其设计展示在类图和顺序图（如图 76 所示）中。

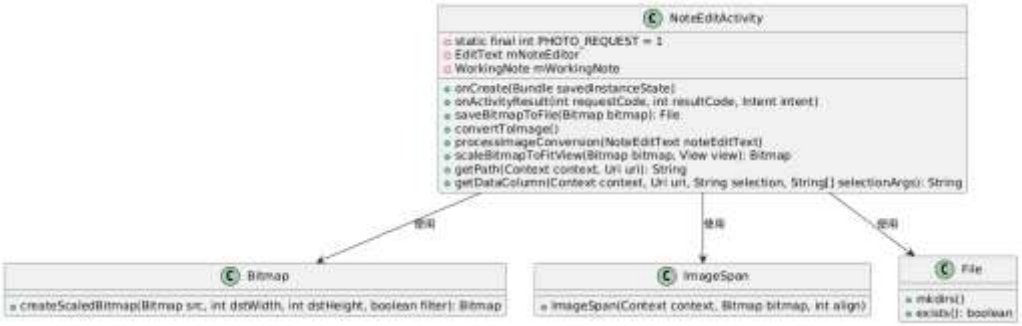


图 76 小米便签插入图片功能类图

功能具体设计如下（图 77）：

1. **用户交互设计**：首先，设计一个用户友好的界面，允许用户通过点击按钮来启动图片选择流程。
2. **图片选择流程**：用户点击图片按钮后，通过 Intent 启动图片选择器，让用户从设备媒体库中选择图片。（如图 79）
3. **图片处理**：用户选择图片后，应用需要处理图片，包括解码、保存到本地、获取图片路径等。
4. **图片显示**：将处理后的图片以合适的方式插入到便签文本中，并在文本编辑器中显示。（如图 78）

5. **数据持久化：**将图片信息（如路径）保存到数据库，以便下次打开便签时能够恢复图片显示。

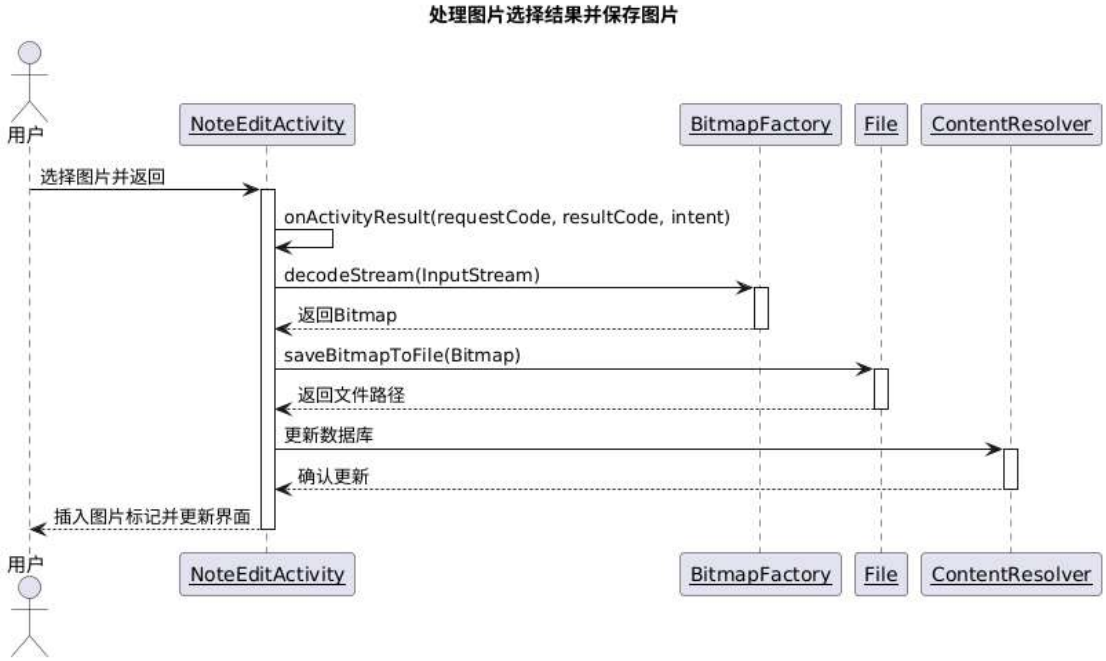


图 77 处理图片选择结果并保存图片顺序图

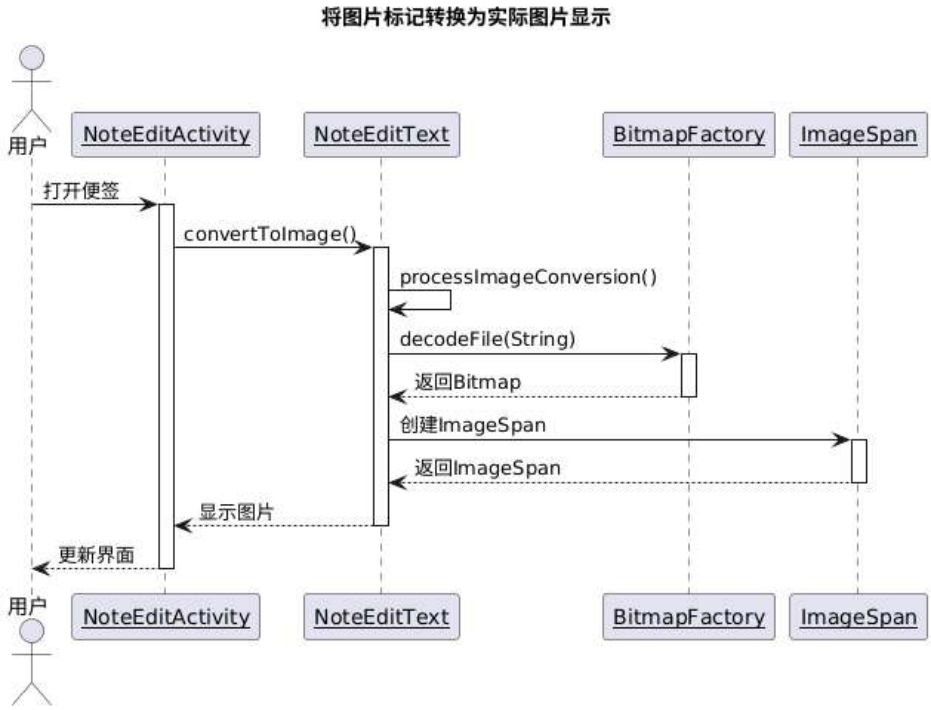


图 78 将图片标记转换为实际图片显示功能顺序图

图 79 用户点击按钮并选择图片功能顺序图

(三) 代码实现

1. 图片选择器的启动：

在 NoteEditActivity 类的 onCreate 方法中，我们设计了一个 ImageButton，其 ID 被设定为 add_img_btn。这个按钮不仅是用户界面的一部分，也是启动图片选择流程的关键。我们为这个按钮配置了一个点击监听器，当用户点击这个按钮时，就会触发图片选择器的启动(如图 80 所示)。图片选择器通过 Intent.ACTION_GET_CONTENT 这一 Intent 动作来实现，它允许用户从他们的设备中灵活选择图片，无论是从相册还是其他图片库中。

```
//根据id获取添加图片按钮
final ImageButton add_img_btn = (ImageButton) findViewById(R.id.add_img_btn);
//为点击图片按钮设置监听器
add_img_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.d(TAG, "onClick: click add image button");
        //ACTION_GET_CONTENT: 允许用户选择特殊种类的数据，并返回（特殊种类的数据：照一张相片或录一段音）
        Intent loadImage = new Intent(Intent.ACTION_GET_CONTENT);
        //Category属性用于指定当前动作（Action）被执行的环境。
        //CATEGORY_OPENABLE: 用来指示一个ACTION_GET_CONTENT的intent
        loadImage.addCategory(Intent.CATEGORY_OPENABLE);
        loadImage.setType("image/*");
        startActivityForResult(loadImage, PHOTO_REQUEST);
    }
});
```

图 80 插入图片按钮实现

2. 处理图片选择结果：

在 onActivityResult 方法中，我们精心处理了图片选择的结果。首先，我们会检查请求码是否与我们预设的 PHOTO_REQUEST 相匹配，并且结果是否为 RESULT_OK。如果这两个条件都满足，那么我们就知道用户已经成功选择了一张图片。接下来，我们通过 getPath、getDataColumn 等方法来获取用户所选图片的 Uri，这是为了确保我们能够准确地定位并处理用户选择的图片。具体代码实现如图 81 所示。

```
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    if (requestCode == PHOTO_REQUEST && resultCode == RESULT_OK && intent != null) {
        Uri originalUri = intent.getData();
        Bitmap bitmap = null;
        try {
            bitmap = BitmapFactory.decodeStream(getContentResolver().openInputStream(originalUri));
        } catch (FileNotFoundException e) {
            Log.d(TAG, "onActivityResult: get file_exception");
            e.printStackTrace();
        }
        if (bitmap != null) {
            File savedFile = saveBitmapToFile(bitmap);
            if (savedFile != null) {
                String filePath = savedFile.getAbsolutePath();
                String img_fragment = "[local]" + filePath + "[/local]";
                final NoteEditText e = (NoteEditText) findViewById(R.id.note_edit_view);


                Editable edit_text = e.getEditableText();
                int index = e.getSelectionStart();
                edit_text.insert(index, img_fragment);
                mWorkingNote.mContent = e.getText().toString();
                // 更新数据库
                ContentResolver contentResolver = getContentResolver();
                ContentValues contentValues = new ContentValues();
                final long id = mWorkingNote.getNoteId();
                contentValues.put("content", mWorkingNote.mContent);
                contentResolver.update(Uri.parse("content://micode_notes/data"), contentValues, "mime_type=? and note_id=?", new Strin
```

图 81 onActivityResult 方法实现

3. 图片的保存与路径标记：

获取到图片的 Uri 后，我们使用 BitmapFactory.decodeStream 方法(如图 82 所示)将其转换为 Bitmap 对象，以便进行进一步的处理。然后，我们调用 saveBitmapToFile 方法，将这个 Bitmap 保存到应用的私有目录中，文件名以当前的时间戳加上.jpg 后缀来命名，确保每个文件都是唯一的。保存图片后，我们会在文本编辑器中生成一个图片路径标记，并更新数据库以保存这些更改，这样用户就可以在便签中看到他们插入的图片了。

```
/**
 * 显示一个Toast消息。
 *
 * @param resId 资源ID，指向要显示的字符串
 */
private void showToast(int resId) {
    showToast(resId, Toast.LENGTH_SHORT);
}

private File saveBitmapToFile(Bitmap bitmap) {
    File file = new File(getFilesDir(), child:"images");
    if (!file.exists()) {
        file.mkdirs();
    }
    String filename = System.currentTimeMillis() + ".jpg";
    File outputFile = new File(file, filename);
    try (FileOutputStream fos = new FileOutputStream(outputFile)) {
        bitmap.compress(Bitmap.CompressFormat.JPEG, 90, fos);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return outputFile;
}
```

图 82 保存图片实现

4. 图片显示与调整：

为了在文本编辑器中显示图片，我们实现了 convertToImage 方法(如图 83 所示)。这个方法首先检查视图的宽度和高度，如果这些值是 0，说明视图还没有完成布局，我们会添加一个全局布局监听器，等待布局完成后再处理图片显示。如果视图的尺寸已经确定，我们就直接调用 processImageConversion 方法。processImageConversion 方法会遍历文本内容，找到图片的标记，然后读取对应的图片文件，并创建 ImageSpan 来在文本中显示图片。为了确保图片能够适应 TextView 的尺寸，我们实现了 scaleBitmapToFitView 方法，它负责调整 Bitmap 的大小，使其完美适配 TextView 的尺寸，从而在用户界面中呈现出最佳的视觉效果。

```
private void convertToImage() {
    final NoteEditText noteEditText = findViewById(R.id.note_edit_view);
    int viewWidth = noteEditText.getWidth();
    int viewHeight = noteEditText.getHeight();

    if (viewWidth > 0 && viewHeight > 0) {
        processImageConversion(noteEditText);
    } else {
        noteEditText.getViewTreeObserver().addOnGlobalLayoutListener(new ViewTreeObserver.OnGlobalLayoutListener() {
            @Override
            public void onGlobalLayout() {
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
                    noteEditText.getViewTreeObserver().removeOnGlobalLayoutListener(this);
                } else {
                    noteEditText.getViewTreeObserver().removeGlobalOnLayoutListener(this);
                }
                processImageConversion(noteEditText);
            }
        });
    }
}
```

图 83 图片转化函数

3.2.3.11 便签更换背景功能

(一) 功能介绍

小米便签应用的更改背景功能为用户提供了一种个性化定制界面的方式。当用户首次启动应用并进入主界面时，系统将自动加载用户预先选择的背景图片，为便签列表增添视觉美感。通过点击应用内的菜单选项，用户可以浏览并选择不同的背景图案，以满足个人审美和心情变化。选择新背景后，界面会即时更新，无需复杂操作，即可让便签界面焕然一新，让记录和回顾便签的过程更加愉悦。这一功能不仅丰富了用户的使用体验，也使得小米便签成为一个更具个性化和情感色彩的记录工具。

(二) 功能详细设计

小米便签在启动时，会自动加载用户选定的背景图片，为用户呈现个性化的视觉体验。用户可以通过简单的操作，在应用内更换便签的背景，使得记录便签的过程更加愉悦和个性化。其设计类图与顺序图如下图 86、图 84 和图 85 所示。

代码实现细节

1. **背景图片加载**：当用户进入小米便签的主界面，NotesListActivity 会通过 onCreate 方法设置布局，并加载预先设定的背景图片，如 R.drawable.feng，确保用户界面的美观性。

2. **菜单界面展示**：用户在便签列表中点击菜单按钮，系统会通过 onPrepare OptionsMenu 方法准备选项菜单，展示可更换的背景图片选项。

3. **背景更换操作**：用户在菜单中选择新的背景图片后，NotesListActivity 会

清除之前的菜单项，并加载新的菜单布局，包括新的背景选项。用户的选择将通过 inflate 方法应用到当前的便签列表界面。

4. **即时视觉反馈**：更换背景后，用户会立即看到界面的变化，这种即时的视觉反馈增强了用户的交互体验。

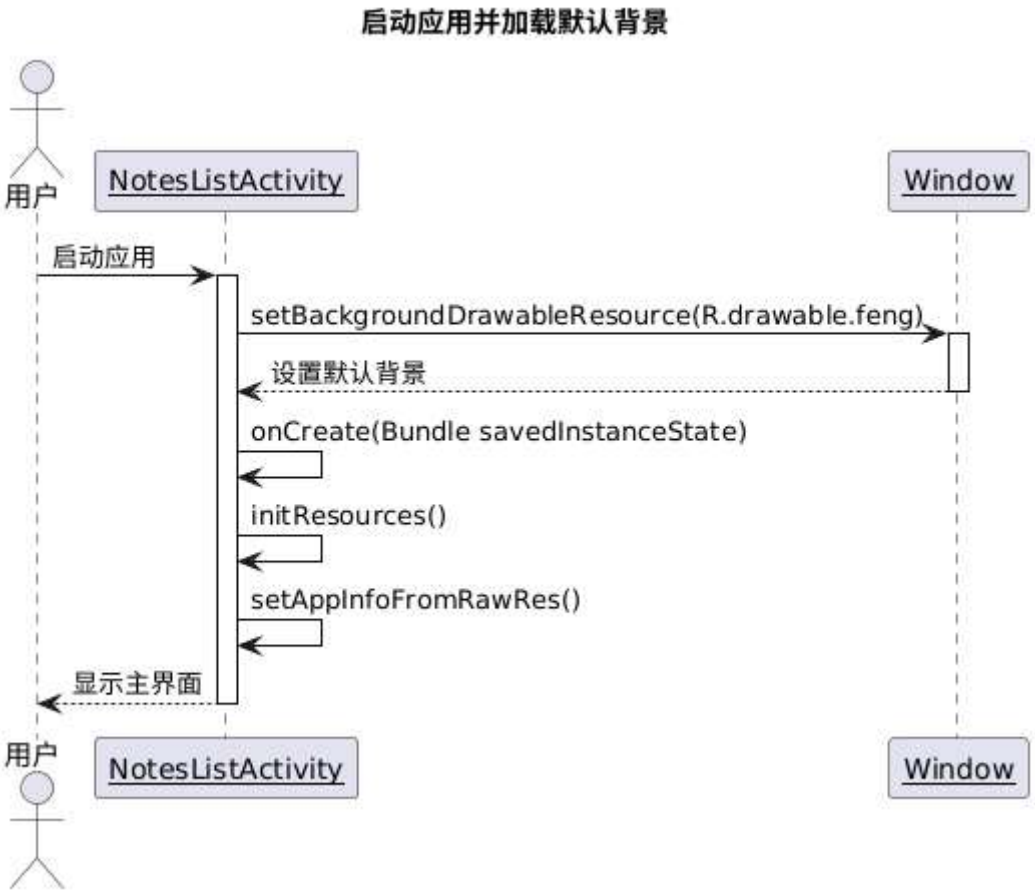


图 84 启动应用并加载默认背景功能顺序图

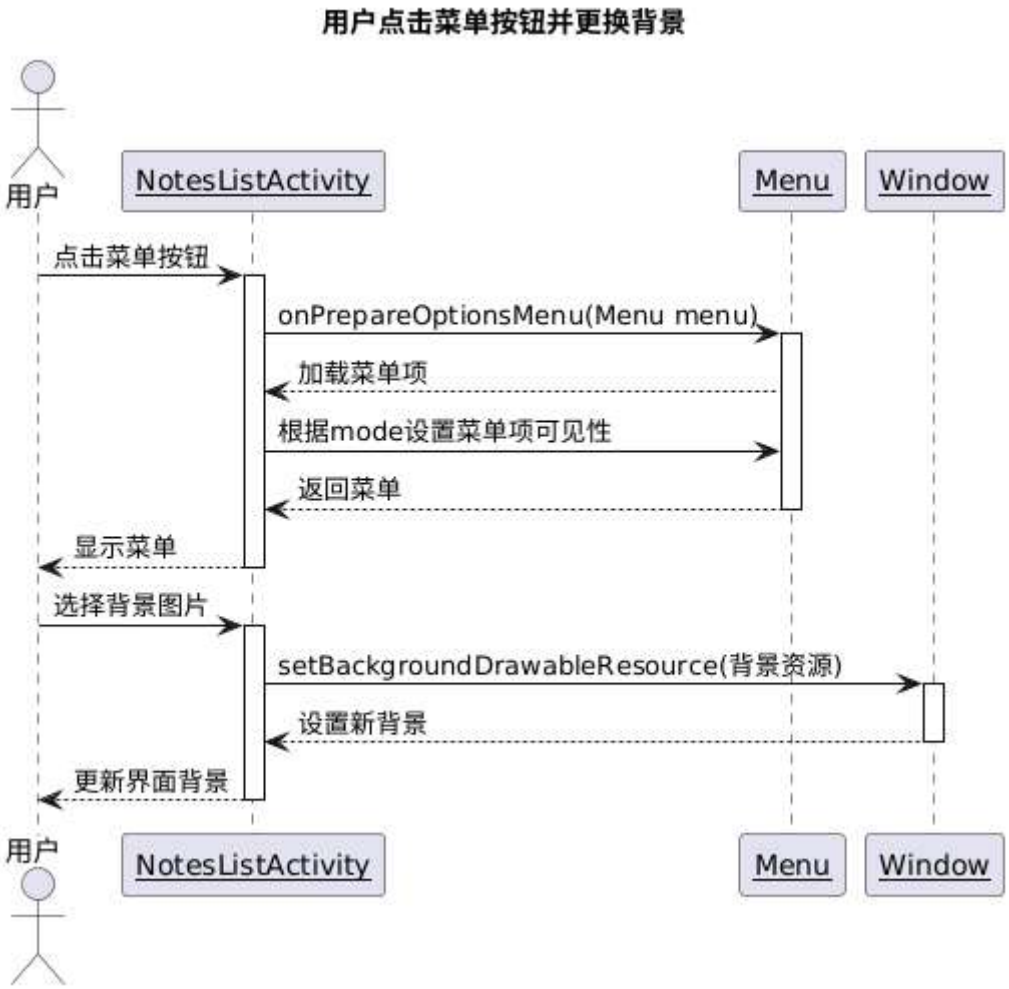


图 85 用户点击菜单按钮并更换背景功能顺序图

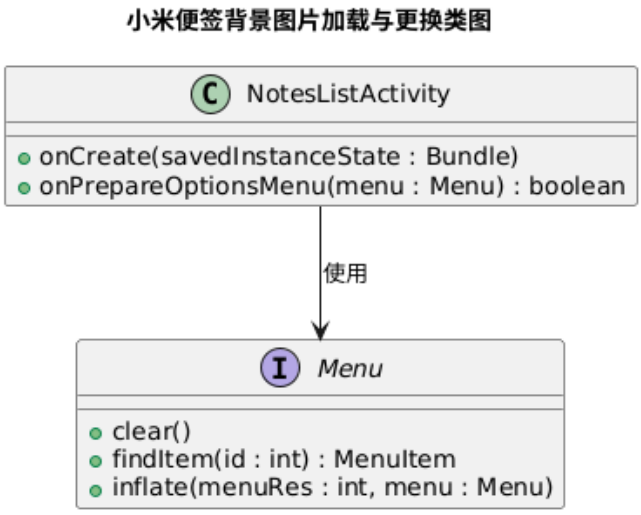


图 86 更换背景相关类类图

(三) 代码实现

小米便签应用中的背景更换功能通过精心设计的界面和灵活的资源管理，为用户提供了一个直观且个性化的背景定制体验。在应用启动时，NotesListActivit

y 通过 onCreate 方法加载预设的背景图片，为用户呈现一个美观的初始界面。当用户通过点击菜单按钮进入选项菜单时，onPrepareOptionsMenu 方法会被触发，它根据当前的编辑状态动态加载相应的菜单布局，并根据用户当前选择的背景模式调整菜单项的可见性，确保用户不会重复选择相同的背景。

代码实现描述：

1. **初始化背景**：在 NotesListActivity 的 onCreate 方法中，应用启动时会加载一个预设的背景图片。通过 setContentView 设置便签列表的布局，并使用 getWindow().setBackgroundDrawableResource 方法应用背景资源(如图 87 所示)，如 R.drawable.feng。

```
/**
 * 在活动创建时调用，用于初始化资源和设置应用信息。
 *
 * @param savedInstanceState 如果活动之前被销毁，这参数包含之前的状态。如果活动没被销毁之前，这参数是null。
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.note_list);

    getWindow().setBackgroundDrawableResource(R.drawable.feng);

    initResources();

    // 用户首次使用时插入介绍信息
    setAppInfoFromRawRes();
}
```

图 87 初始化加载背景

2. **准备选项菜单**：当用户点击菜单按钮时，onPrepareOptionsMenu 方法被调用。此方法首先清除现有的菜单项，然后根据当前的编辑状态（mState），使用 getMenuInflater().inflate 方法加载相应的菜单布局。根据当前的背景模式（由 mode 变量控制），onPrepareOptionsMenu 方法会调整菜单项的可见性。例如，如果当前背景是 feng，则隐藏对应的菜单项，避免用户重复选择相同的背景，如下图 88 所示。

```
// 根据mode变量设置菜单项的可见性
if (mode == -1) {
    menu.findItem(R.id.menu_feng).setVisible(false);
} else if (mode == 0) {
    menu.findItem(R.id.menu_lee).setVisible(false);
} else if (mode == 1) {
    menu.findItem(R.id.menu_zhang).setVisible(false);
}
```

图 88 隐藏按钮

3. **背景更换操作**：用户在菜单中选择一个新的背景选项后，系统会立即应用

新的背景图片。这通过更改 mode 变量的值并重新调用 onPrepareOptionsMenu 方法来实现(如图 89 所示)，确保菜单项的可见性与当前背景相匹配。

```
case R.id.menu_lee:
    mode = 0 ;
    getWindow().setBackgroundDrawableResource(R.drawable.green);
    break;

case R.id.menu_feng:
    mode = -1 ;
    getWindow().setBackgroundDrawableResource(R.drawable.feng);
    break;

case R.id.menu_zhang:
    mode = 1 ;
    getWindow().setBackgroundDrawableResource(R.drawable.shuimo);
    break;
```

图 89 设置不同背景

3.2.3.12 字符统计功能

(一) 功能介绍

字符统计功能为用户提供了实时统计便签内容字符数的功能。当用户在编辑便签时，系统会自动统计当前便签的字符数，并在界面上显示。这一功能帮助用户了解便签的长度，避免超出字符限制，并提升用户的编辑体验。

(二) 功能详细设计

字符统计功能的工作流程是：在便签编辑界面初始化时，设置字符统计功能。通过监听文本变化事件，实时更新字符数。当用户输入或删除字符时，触发文本变化事件，更新字符统计。在统计字符数时，过滤掉不计入字符数的特殊字符（如图片标签、换行符、空格等）。将统计结果显示在界面上，实时反馈给用户。本功能涉及到的类图如图 90 所示。实现功能的顺序图如图 91 所示。

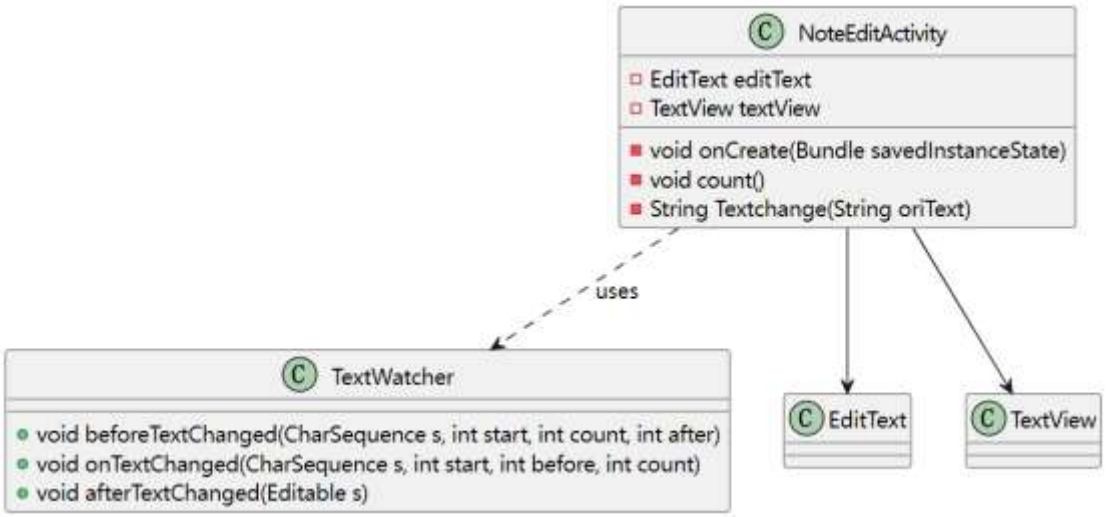


图 90 字符统计功能类图

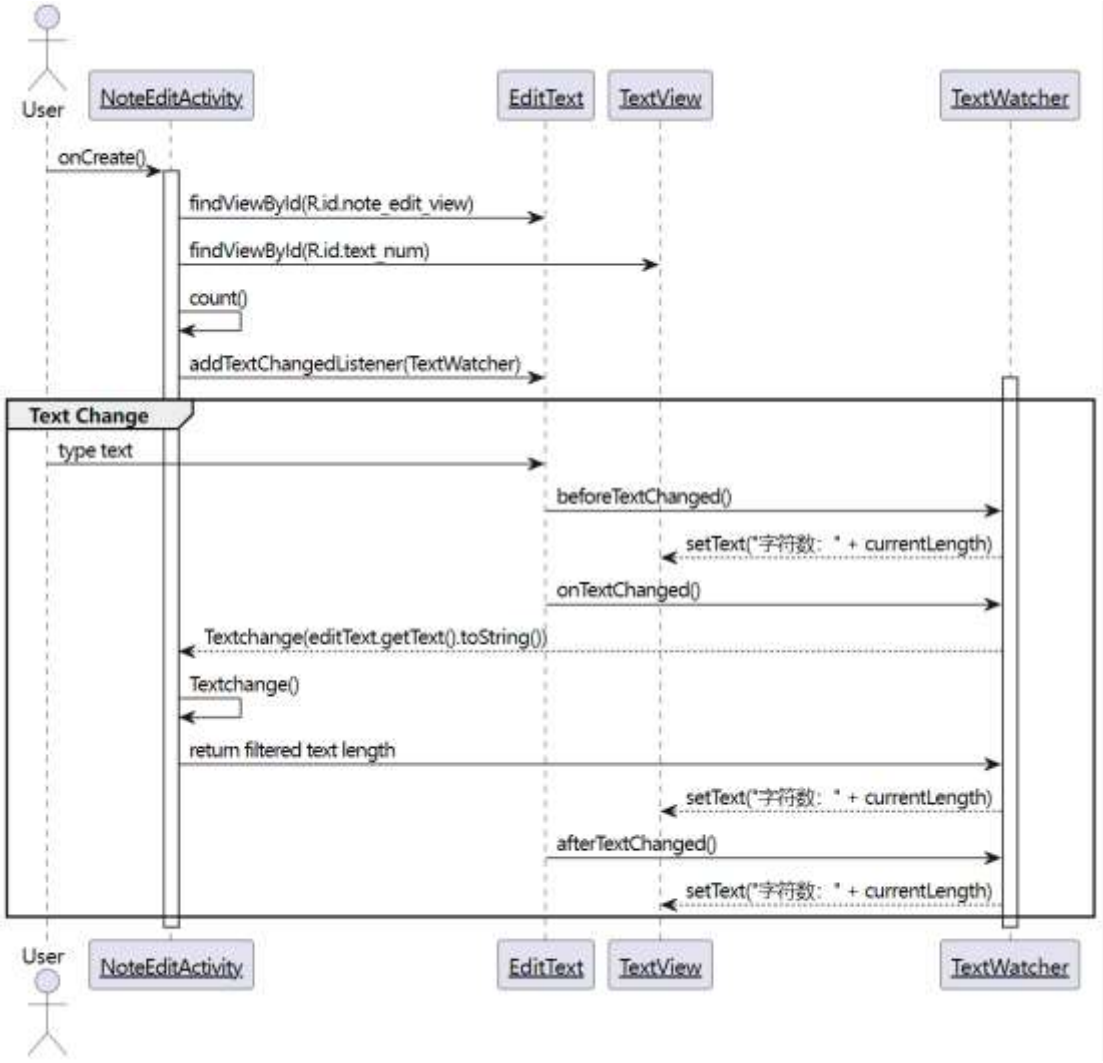


图 91 字符统计功能顺序图

(三) 代码实现

在 NoteEditActivity 类的 onCreate 方法中，初始化字符统计功能。在 NoteEditActivity 类中，添加文本变化监听器，实时更新字符数。在 NoteEditActivity 类中，定义 Textchange 方法，过滤不计入字符数的特殊字符。具体实现字符过滤功能的代码如图 92 所示。

```
private String Textchange(String oriText) {
    StringBuffer stringBuffer = new StringBuffer(oriText);
    int Flag1 = -1;
    int Flag2 = -1;
    do {
        Flag1 = stringBuffer.indexOf("<img");
        Flag2 = stringBuffer.indexOf(">");
        if (Flag1 != -1 && Flag2 != -1) {
            stringBuffer = stringBuffer.replace(Flag1, Flag2 + 1, "");
        }
    } while (Flag1 != -1 && Flag2 != -1);

    do {
        Flag1 = stringBuffer.indexOf("\n");
        if (Flag1 != -1) {
            stringBuffer = stringBuffer.replace(Flag1, Flag1 + 1, "");
        }
    } while (Flag1 != -1);

    do {
        Flag1 = stringBuffer.indexOf(" ");
        if (Flag1 != -1) {
            stringBuffer = stringBuffer.replace(Flag1, Flag1 + 1, "");
        }
    } while (Flag1 != -1);

    return stringBuffer.toString();
}
```

图 92 过滤特殊字符代码实现图

3.2.3.13 文本撤回功能

(一) 功能介绍

文本撤回功能为用户提供了一种便捷的方式来撤回最近一次的文本修改。当用户在编辑便签时，可以通过点击撤回按钮，系统会自动撤回最近一次的文本修改。这一功能帮助用户在编辑过程中快速恢复到之前的状态，提升用户的编辑体验。

(二) 功能详细设计

在便签编辑界面初始化时，设置撤回按钮。通过监听撤回按钮的点击事件，触发撤回操作。当用户输入或删除字符时，触发文本变化事件，保存当前文本到历史记录。当用户点击撤回按钮时，从历史记录中取出最近一次的文本状态，并恢复到编辑框中。使用栈结构保存文本的历史记录，确保撤回操作的顺序性和准确性。本功能涉及到的类图如图 93 所示。实现功能的顺序图如图 94 所示。

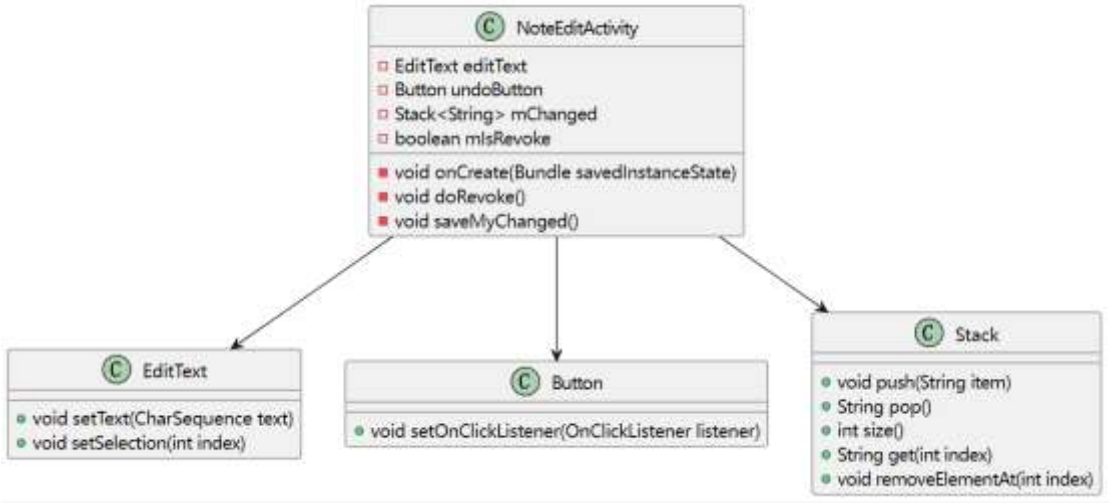


图 93 撤回文本功能类图

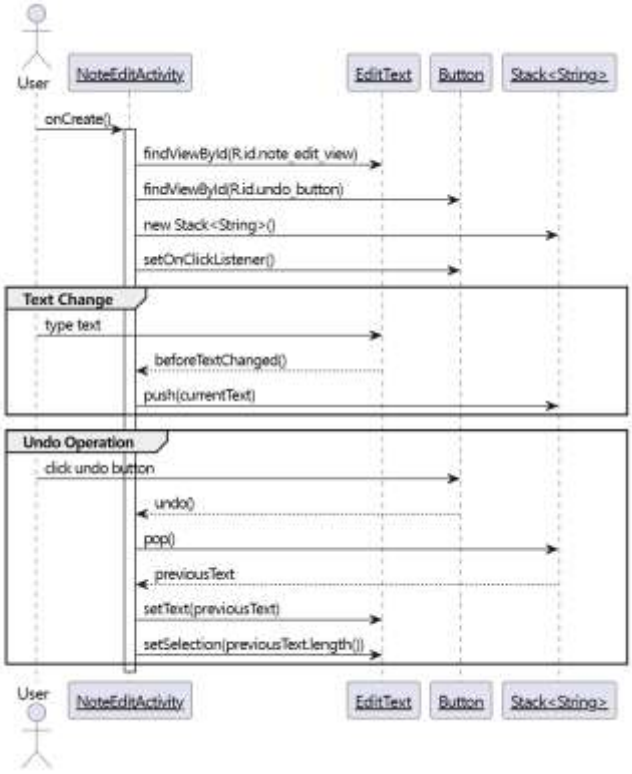


图 94 撤回文本功能顺序图

(三) 代码实现

在 NoteEditActivity.java 类的 onCreate 方法中，初始化撤回按钮并设置点

击事件监听器。修改 XML 布局文件，在 note_edit.xml 文件中添加一个用于撤回操作的按钮。新增 doRevoke（）方法，具体代码实现如图 95 所示。

```
private boolean mIsRevoke = false;
private Vector<SpannableString> mChanged = new Vector<>();
/**
 * 撤回功能主方法
 */
private void doRevoke(){
    int size = mChanged.size();
    AlertDialog.Builder dialog = new AlertDialog.Builder(this);//创建一个alertdialog窗口
    dialog.setTitle(R.string.tips_of_revoke);//设置title信息
    dialog.setCancelable(true);//设置为可取消
    dialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {//只需要设置一个OK键即可
        @Override
        public void onClick(DialogInterface dialog, int which) {
        }
    });
    mIsRevoke = true;//把是否已执行撤销的标记设置为true
    if(size<=1) {//如果栈中元素过少，打印提示信息
        dialog.setMessage(R.string.have_not_input_anything);//提示用户您还没有输入任何信息
        dialog.show();//显示当前alertdialog
        return;
    }else {
        mNoteEditor.setText((CharSequence) mChanged.get(size-2));//将栈中倒数第二个元素取出并显示在文本框中
        mNoteEditor.setSelection(mNoteEditor.length());
        mChanged.removeElementAt(size-1);//将栈中最后一个元素删除
        if(size == 2){
            dialog.setMessage(R.string.can_not_revoke);//提示用户您还没有输入任何信息
            dialog.show();//显示当前alertdialog
        }
    }
}
```

图 95 撤回文本功能代码实现图

3.2.3.14 朗读文本功能

(一) 功能介绍

朗读文本功能为用户提供了一种便捷的方式，将便签中的文本内容转换为语音并自动朗读。用户可以在便签编辑界面中点击朗读按钮，系统会自动调用 Android 的 Text-to-Speech（TTS）引擎，将文本框中的内容转换为语音并播放。在朗读过程中，屏幕底部会显示两个按钮：一个用于暂停或继续朗读，另一个用于退出朗读模式。用户可以通过点击暂停按钮暂停当前的朗读，点击继续按钮恢复朗读，或者点击退出按钮直接退出朗读模式。该功能提升了便签的可访问性和用户体验，特别适用于需要语音辅助的场景。

(二) 功能详细设计

在便签编辑界面初始化时，系统会加载朗读功能所需的资源，并初始化 TTS 引擎。用户点击朗读按钮后，系统会获取文本框中的内容，并将其传递给 TTS 引擎进行语音合成。TTS 引擎会将文本转换为语音并开始播放。在朗读过程中，系统会实时监听朗读状态，并根据用户的操作（如暂停、继续或退出）调整朗读

行为。

朗读功能的实现依赖于 Android 的 TextToSpeech 类，通过该类可以实现文本到语音的转换。系统还实现了 UtteranceProgressListener 接口，用于监听朗读的开始、暂停、完成和错误事件。为了确保朗读的流畅性，系统还实现了暂停和继续功能，通过记录已朗读的时间来计算剩余文本，并在恢复朗读时从剩余文本开始播放。

本功能涉及到的类图如图 96 所示。实现功能的顺序图如图 97 图 98 图 99 所示。

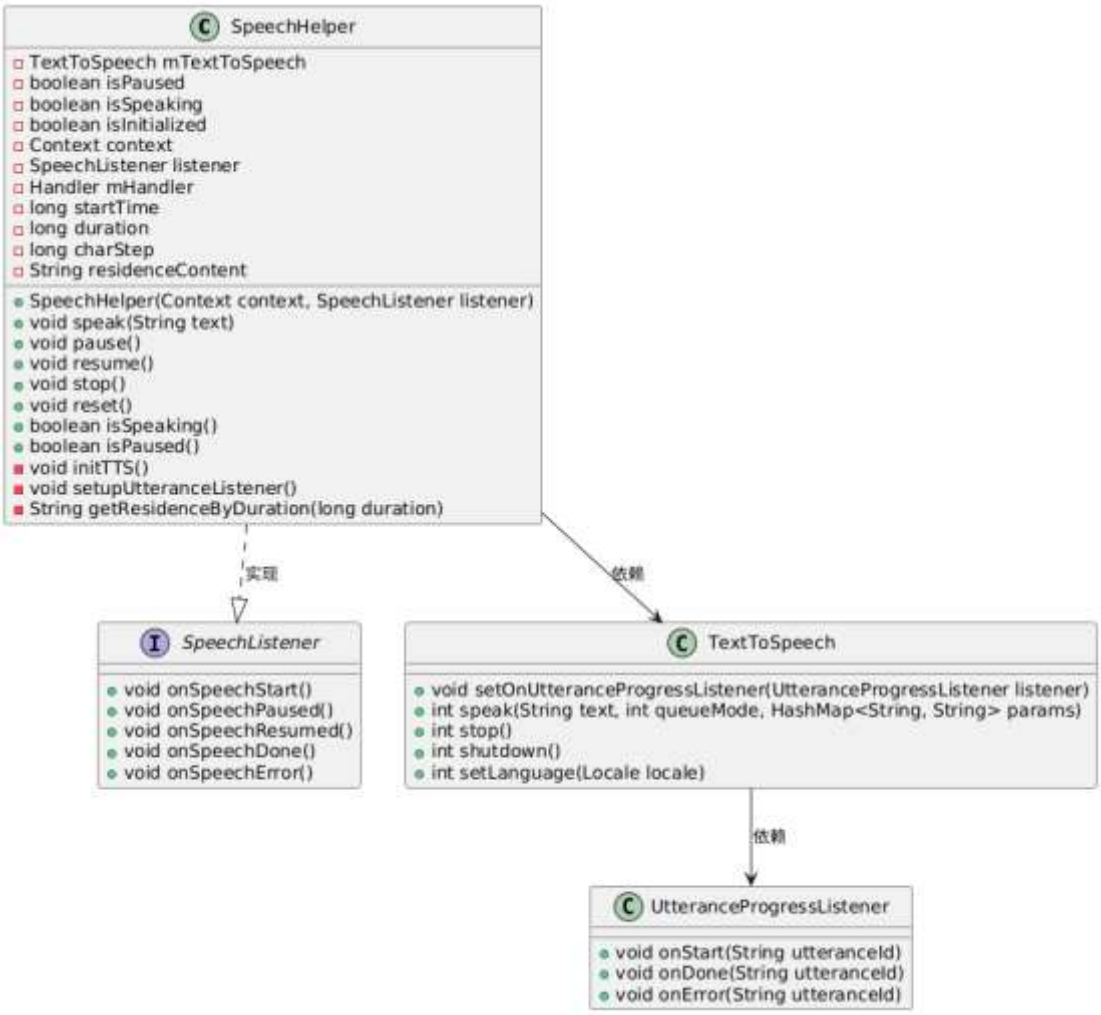


图 96 朗读功能类图

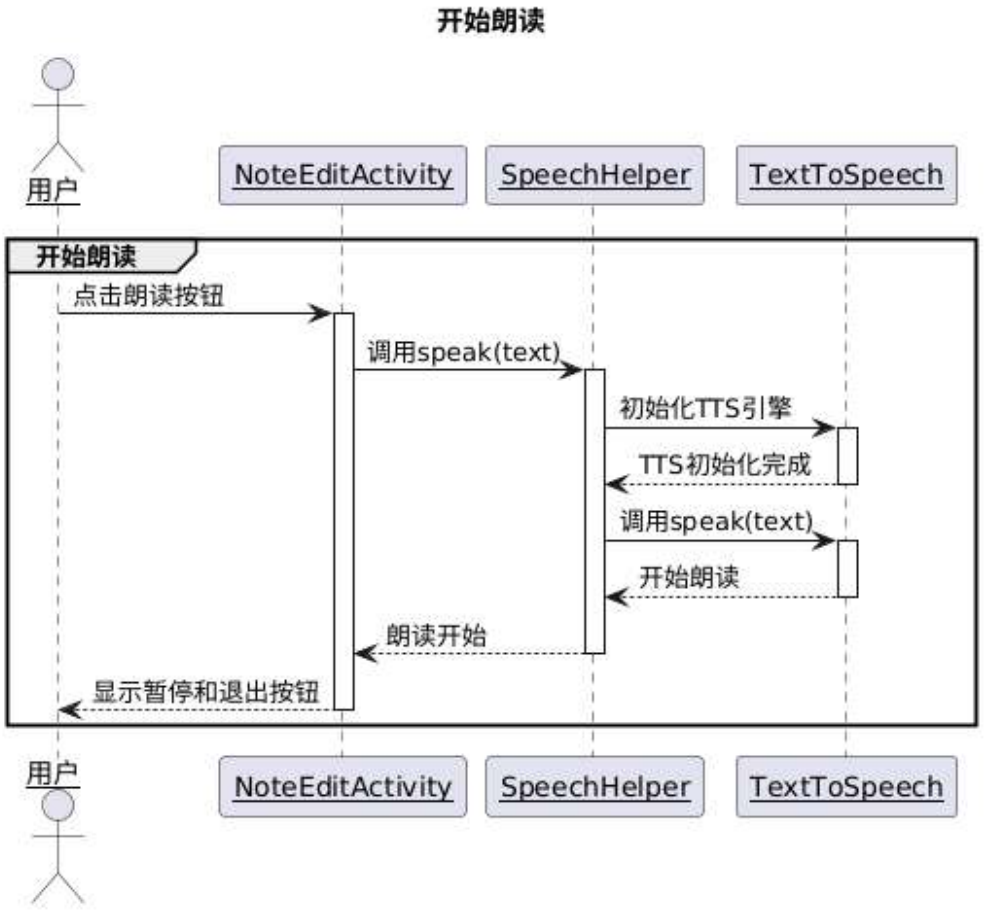


图 97 开始朗读功能顺序图

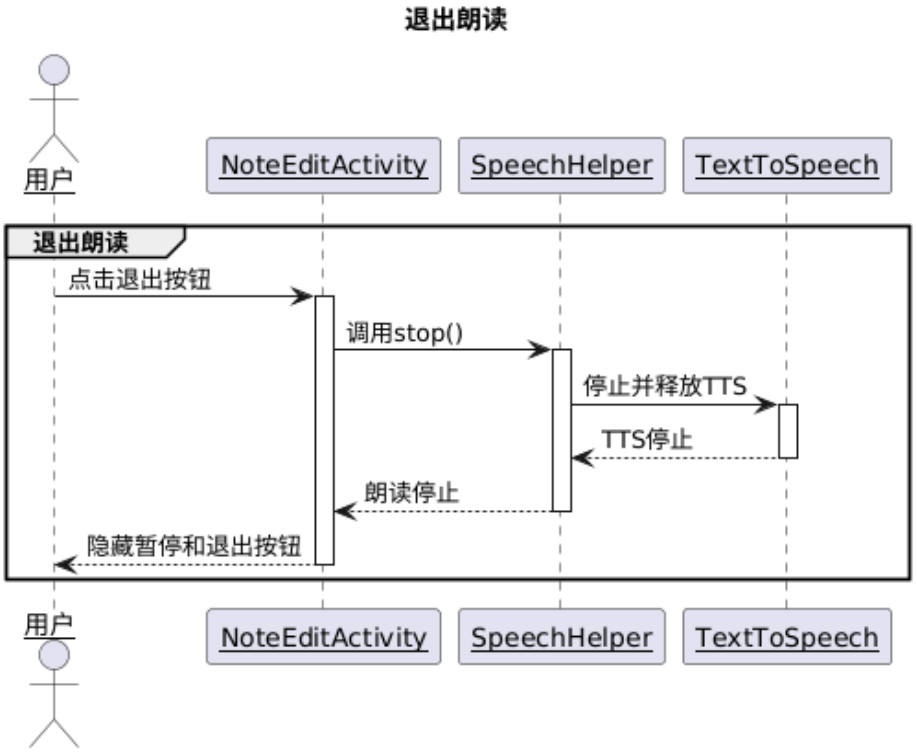


图 98 退出朗读功能顺序图

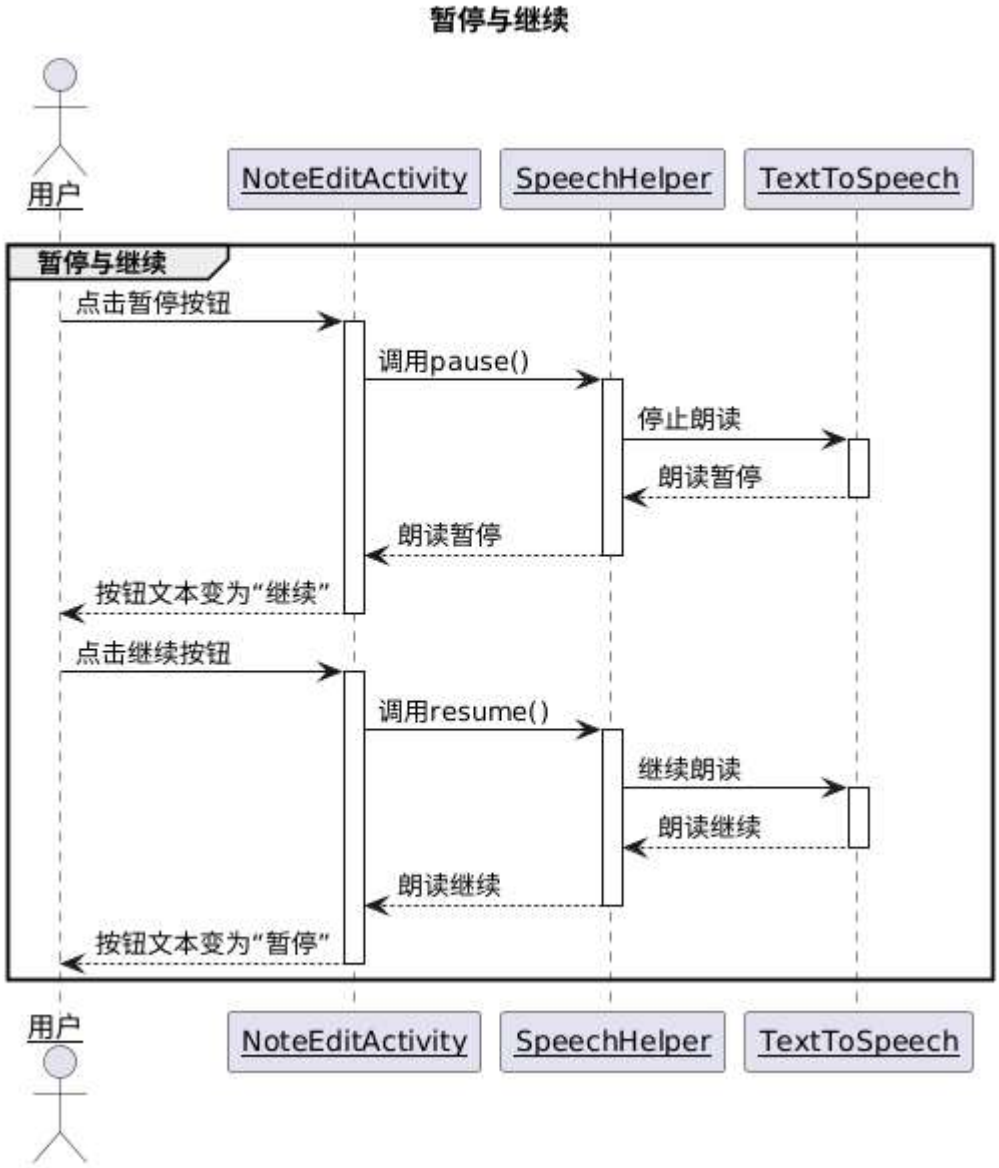


图 99 暂停与继续功能顺序图

(三) 代码实现

在 NoteEditActivity.java 类中，朗读文本功能的实现主要分为以下几个步骤：

1. 初始化 TTS 引擎：在 onCreate 方法中，初始化 TTS 引擎，并设置朗读状态的回调监听器。当朗读开始、暂停、继续或完成时，系统会通过回调函数更新 UI 界面。
2. 开始朗读：当用户点击朗读按钮时，系统会获取文本框中的内容，并调用 SpeechHelper 的 speak 方法开始朗读。具体代码如图 100 所示。

```
case R.id.speak_out:
    // 获取文本框内容并朗读
    String text = mNoteEditor.getText().toString();
    if (!text.isEmpty()) {
        speechHelper.speak(text); // 开始朗读
        btnPauseResume.setVisibility(View.VISIBLE); // 显示暂停按钮
        btnStop.setVisibility(View.VISIBLE); // 显示退出朗读按钮
        updatePauseButtonText(); // 更新按钮文本
    } else {
        Toast.makeText( context: this,  text: "文本框内容为空", Toast.LENGTH_SHORT).show();
    }

    break;
```

图 100 开始朗读功能代码实现图

3. 暂停和继续朗读：用户可以通过点击暂停按钮暂停当前的朗读，点击继续按钮恢复朗读。系统会通过 SpeechHelper 的 pause 和 resume 方法实现暂停和继续功能。具体代码实现如图 101 所示。

```
// 设置暂停/继续按钮点击事件
btnPauseResume.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_pause_resume) {
            if (speechHelper.isPaused()) {
                // 恢复朗读
                String text = mNoteEditor.getText().toString();
                speechHelper.resume();
                btnPauseResume.setText("暂停"); // 更新按钮文本

            } else {
                // 暂停朗读
                speechHelper.pause();
                btnPauseResume.setText("继续"); // 更新按钮文本
            }
        } else if (v.getId() == R.id.btn_stop) {
            // 停止朗读
            speechHelper.stop();
            btnPauseResume.setVisibility(View.GONE); // 隐藏暂停按钮
            btnStop.setVisibility(View.GONE); // 隐藏退出朗读按钮
        }
    }
});
```

图 101 暂停播放功能代码实现图

4. 退出朗读模式：用户可以通过点击退出按钮直接退出朗读模式。系统会

调用 SpeechHelper 的 stop 方法停止当前的朗读，并隐藏暂停和退出按钮。具体代码实现如图 102 所示。

```
// 设置退出朗读按钮点击事件
btnStop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        speechHelper.stop(); // 停止朗读
        btnPauseResume.setVisibility(View.GONE); // 隐藏暂停按钮
        btnStop.setVisibility(View.GONE); // 隐藏退出朗读按钮
    }
});
```

图 102 退出朗读功能实现图

5. 实现 SpeechHelper 类：SpeechHelper 类封装了 TTS 引擎的初始化和操作逻辑，包括开始、暂停、继续和停止朗读等功能。具体代码实现如图 103 所示。

```
public class SpeechHelper implements OnInitListener {
    18 usages
    private TextToSpeech mTextToSpeech; // TTS 引擎
    8 usages
    private boolean isPaused = false; // 是否暂停
    10 usages
    private boolean isSpeaking = false; // 是否正在朗读
    5 usages
    private boolean isInitialized = false; // TTS 是否初始化完成
    2 usages
    private Context context;
    11 usages
    private SpeechListener listener; // 回调接口
    5 usages
    private Handler mHandler = new Handler(Looper.getMainLooper()); // 用于切换到 UI 线程
```

图 103 TTS 引擎初始化代码实现图

3.2.3.15 大模型交互功能

(一) 功能介绍

我们利用讯飞星火的开源接口，在小米便签中集成了讯飞星火大模型的多种能力，主要包括以下两个核心功能：

用户可以在便签编辑界面中点击菜单中的“大模型对话”按钮，进入与大模型的交互模式。界面会从全屏编辑模式切换为分屏模式，左侧显示与大模型的聊天界面，右侧显示缩放后的编辑界面。用户可以通过左侧的聊天界面与大模型进行实时对话，并将对话中有用的内容复制到右侧的编辑界面中。该功能利用讯飞

星火大模型的自然语言处理能力，为用户提供智能化的文本生成和信息检索服务。

用户还可以在便签编辑界面中点击菜单中的“语音听写”按钮，进入讯飞星火提供的语音听写功能主界面。该界面提供了多种语音相关功能的入口，包括语音识别（语音转文字）、语音合成（文字转语音）、人像识别、语音评测等。用户可以通过这些功能体验语音技术的实际应用，例如将语音实时转换为文字、将文字合成为语音、进行人脸识别操作，以及对语音进行评测和打分。

通过这两个功能，用户不仅可以与大模型进行智能对话，还可以体验讯飞星火提供的多种语音技术，提升便签应用的智能化水平和用户体验。

(二) 功能详细设计

在便签编辑界面中，用户点击“大模型对话”按钮后，界面从全屏编辑模式切换为分屏模式。左侧为聊天界面，包含一个 RecyclerView 用于显示聊天记录，一个 EditText 用于输入问题，以及一个“发送”按钮。右侧为缩放后的编辑界面，用户可以在聊天界面中与大模型对话，并将结果复制到编辑界面。用户在聊天界面中输入问题后，点击“发送”按钮，系统将问题发送至讯飞星火大模型 API。大模型返回响应后，系统将问题和响应添加到聊天记录中，并滚动到最新消息。用户可以选择聊天记录中的内容，复制到右侧的编辑界面。通过 isChatVisible 变量管理聊天界面的显示状态，点击“退出对话”按钮后，恢复全屏编辑界面。

用户点击“语音听写”按钮后，跳转至讯飞星火提供的语音听写功能主界面。该界面提供了多种语音相关功能的入口，包括语音识别、语音合成、人像识别、语音评测等。用户选择具体功能后，进入相应的功能界面。例如，选择“语音转写”功能后，用户可以实时录制语音，系统将语音转换为文字并显示在界面上。在进入功能界面前，系统会检查用户是否同意隐私政策，并请求必要的权限（如录音权限、存储权限、摄像头权限等）。大模型对话功能顺序图如所示，该功能相关的类图如所示。语音听写功能的顺序图如图 106 所示，该功能相关的类图如图 107 所示。

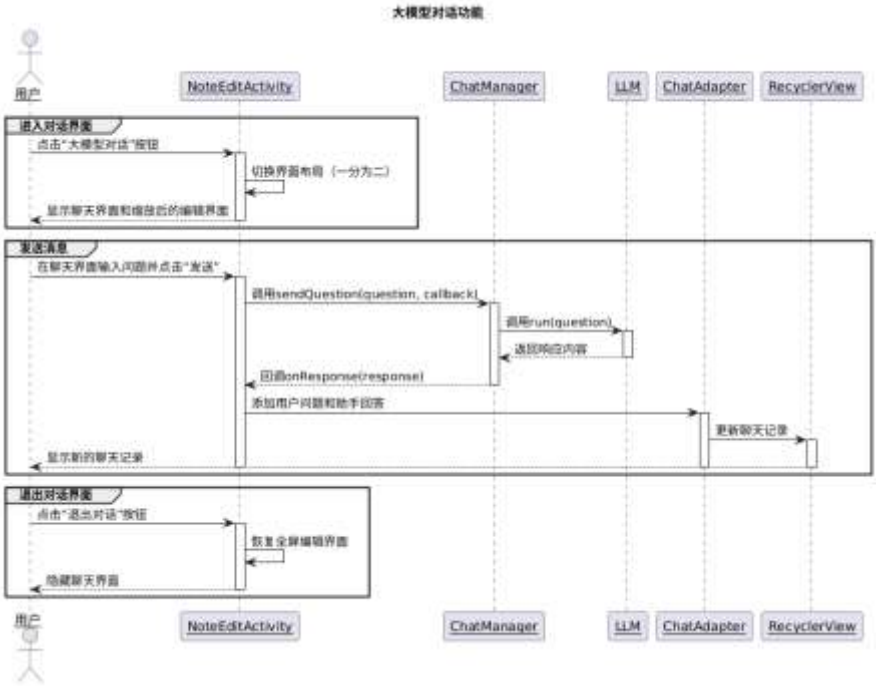


图 104 大模型对话功能顺序图

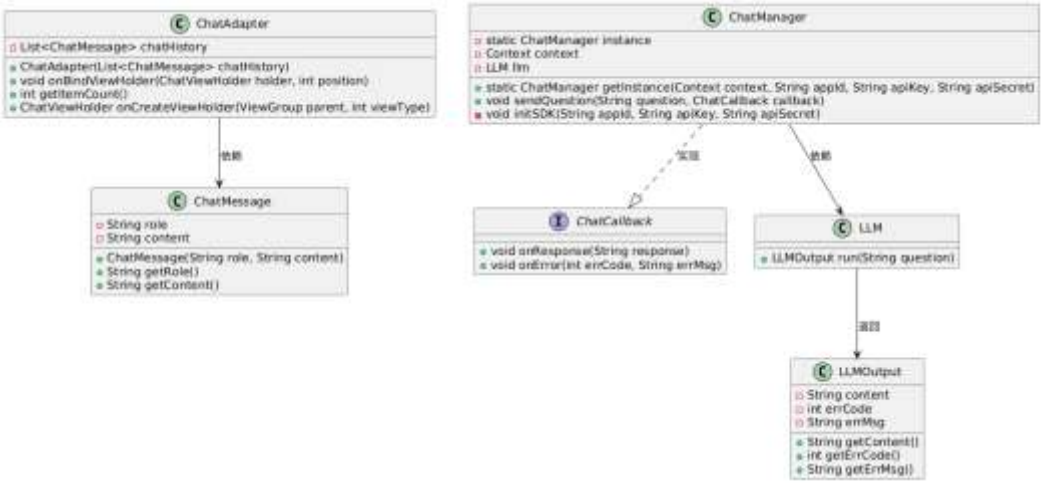


图 105 大模型对话功能类图

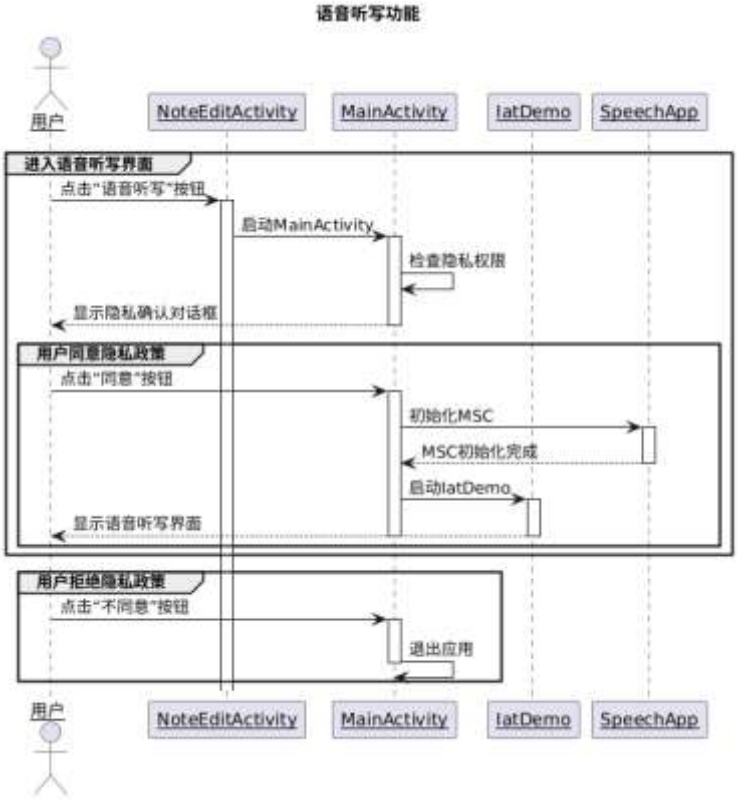


图 106 语音听写功能顺序图

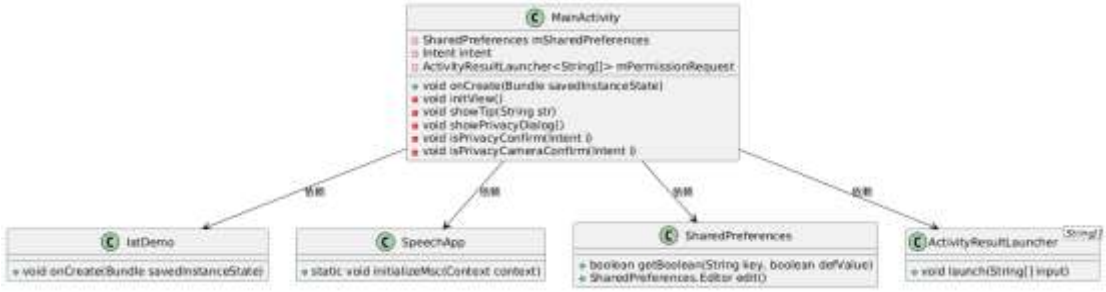


图 107 语音听写功能类图

(三) 代码实现

1. 界面切换与布局管理：

当用户点击“大模型对话”按钮时，界面从全屏编辑模式切换为分屏模式。左侧显示聊天界面，右侧显示缩放后的编辑界面。通过 toggleChatView 方法实现界面切换。具体代码实现如图 108 所示。

```
1 usage
private void toggleChatView() {
    if (isChatVisible) {
        // 隐藏聊天界面，恢复全屏编辑界面
        chatContainer.setVisibility(View.GONE);
        noteContainer.setLayoutParams(new LinearLayout.LayoutParams(
                width: 0, LinearLayout.LayoutParams.MATCH_PARENT,  weight: 1));
    } else {
        // 显示聊天界面，缩放编辑界面
        chatContainer.setVisibility(View.VISIBLE);
        noteContainer.setLayoutParams(new LinearLayout.LayoutParams(
                width: 0, LinearLayout.LayoutParams.MATCH_PARENT,  weight: 1));
    }
    isChatVisible = !isChatVisible;

    // 更新菜单按钮标题
    invalidateOptionsMenu();
}
```

图 108 界面切换与布局管理代码实现图

2. 发送问题并显示响应：

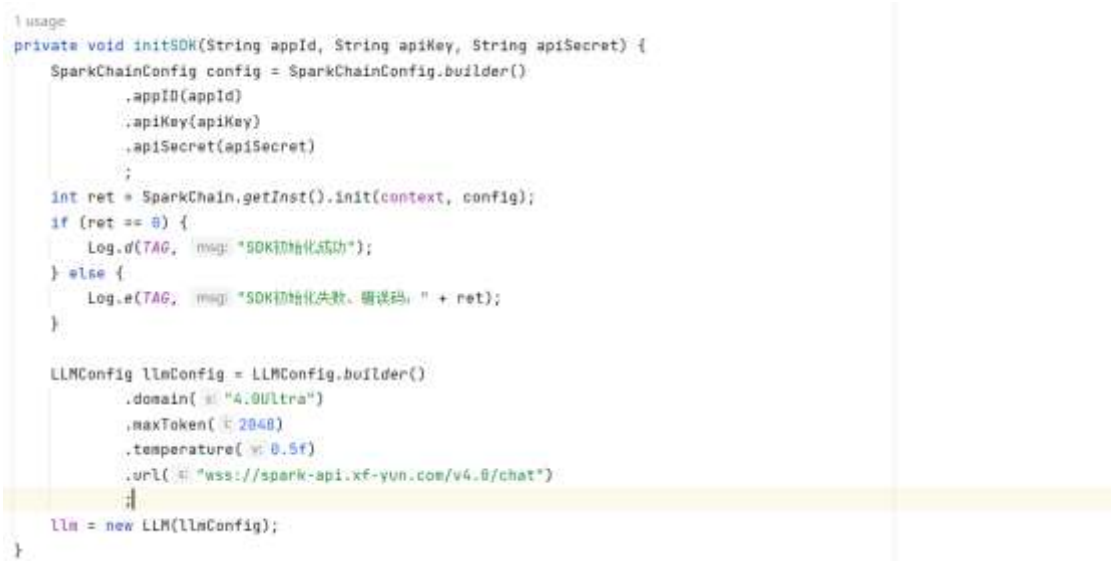
用户在聊天界面中输入问题后，点击“发送”按钮，系统将问题发送至讯飞星火大模型 API，并将响应显示在聊天记录中。具体代码实现如图 109 所示。

```
// 发送按钮点击事件
sendButton.setOnClickListener(v -> {
    String question = inputEditText.getText().toString();
    if (!question.isEmpty()) {
        // 发送问题并处理响应
        chatManager.sendQuestion(question, new ChatManager.ChatCallback() {
            1 usage
            @Override
            public void onResponse(String response) {
                runOnUiThread(() -> {
                    // 将用户的问题和助手的回答添加到聊天记录中
                    chatHistory.add(new ChatMessage( role: "用户", question));
                    chatHistory.add(new ChatMessage( role: "助手", response));
                    chatAdapter.notifyDataSetChanged();
                    chatRecyclerView.scrollToPosition(chatHistory.size() - 1);
                });
            }

            1 usage
            @Override
            public void onError(int errCode, String errMsg) {
                runOnUiThread(() -> {
                    Toast.makeText( context: NoteEditActivity.this,  text: "错误: " + errMsg, Toast.LENGTH_SHORT).show();
                });
            }
        });
        inputEditText.setText("");
    }
});
```

图 109 发送问题并显示响应代码实现图

3. 初始化与调用：

通过 ChatManager 类封装讯飞星火大模型 API 的调用逻辑，包括初始化 SDK、发送问题、处理响应等。具体代码实现如图 110 所示。



```
1 usage
private void initSDK(String appId, String apiKey, String apiSecret) {
    SparkChainConfig config = SparkChainConfig.builder()
            .appID(appId)
            .apiKey(apiKey)
            .apiSecret(apiSecret)
            ;
    int ret = SparkChain.getInst().init(context, config);
    if (ret == 0) {
        Log.d(TAG,  msg: "SDK初始化成功");
    } else {
        Log.e(TAG,  msg: "SDK初始化失败、错误码: " + ret);
    }

    LLMConfig llmConfig = LLMConfig.builder()
            .domain( v: "4.0Ultra")
            .maxToken( v: 2048)
            .temperature( v: 0.5f)
            .url( v: "wss://spark-api.xf-yun.com/v4.0/chat")
            ;
    llm = new LLM(llmConfig);
}
```

图 110 初始化与调用代码实现图

1. 跳转至语音听写界面：

用户点击“语音听写”按钮后，跳转至讯飞星火提供的语音听写功能主界面。具体代码实现如图 111 所示。

```
case R.id.speech_and_listen:
    /*
    your work here
     */
    // 创建 Intent 对象，指定目标应用的包名和 MainActivity 的类名
    Intent intent = new Intent();
    intent.setComponent(new ComponentName( pkg: "com.iflytek.voicedemo",  cls: "com.iflytek.voicedemo.MainActivity"));

    // 启动目标应用的 MainActivity
    startActivity(intent);
    break;
```

图 111 跳转至语音听写界面代码实现图

2. 权限检查与隐私政策确认：

在进入语音听写功能前，系统会检查用户是否同意隐私政策，并请求必要的权限。具体代码实现如图 112 所示。

```
private ActivityResultLauncher<String[]> mPermissionRequest = registerForActivityResult(
        new ActivityResultContracts.RequestMultiplePermissions(),
        result -> {
            //判断所有权限都通过才能执行后续操作
            boolean isAllGranted = true;
            for (Boolean granted : result.values()) {
                if (!granted) {
                    isAllGranted = false;
                    break;
                }
            }
            if (isAllGranted) {
                if (intent != null) {
                    MainActivity.this.startActivity(intent);
                }
                SpeechApp.initializeMsc(MainActivity.this);
            }
        });
```

图 112 权限检查与隐私政策确认代码实现图

3.2.3.16 富文本编辑功能

(一) 功能介绍

富文本编辑功能允许用户在便签中对选中的文字应用多种格式，例如加粗、倾斜、删除线、高亮和下划线等。当用户在便签中长按选中文字时，屏幕底部会浮现出格式工具栏，用户可以通过点击工具栏中的按钮对选中的文字应用相应的格式。便签退出时，系统会自动将附加的格式信息存储到数据库中，下次打开便签时会自动加载并恢复这些格式。该功能提升了便签的文本编辑能力，使用户能够更灵活地处理文本内容。

(二) 功能详细设计

1. 格式工具栏的显示与隐藏

当用户在便签中长按选中文字时，系统会检测到选中事件，并显示底部的格式工具栏。工具栏包含多个按钮，分别对应加粗、倾斜、删除线、高亮和下划线等格式。用户点击某个按钮后，系统会对选中的文字应用相应的格式，并实时更新显示。

2. 格式的存储与加载

便签退出时，系统会将当前文本的富文本样式信息（如加粗、倾斜等）转换为字符串格式，并存储到数据库的 RICH_TEXT_STYLE 字段中。下次打开便签时，系统会从数据库中读取该字段的内容，并将其转换为富文本样式，应用到文本中，恢复之前的格式。

3. 格式的转换与解析

富文本样式信息以 JSON 格式存储，包含每个格式的范围（起始位置和结束位置）和类型（如加粗、倾斜等）。系统通过 SpannableString 和 Span 对象实现文本格式的实时渲染，并通过 convertSpannableToString 和 convertStringToSpannable 方法实现格式信息与字符串之间的转换。

其类图和顺序图如图 113 图 114 图 115 图 116 图 117 图 118 所示

图 113 富文本样式转换顺序图

图 114 加载富文本样式顺序图

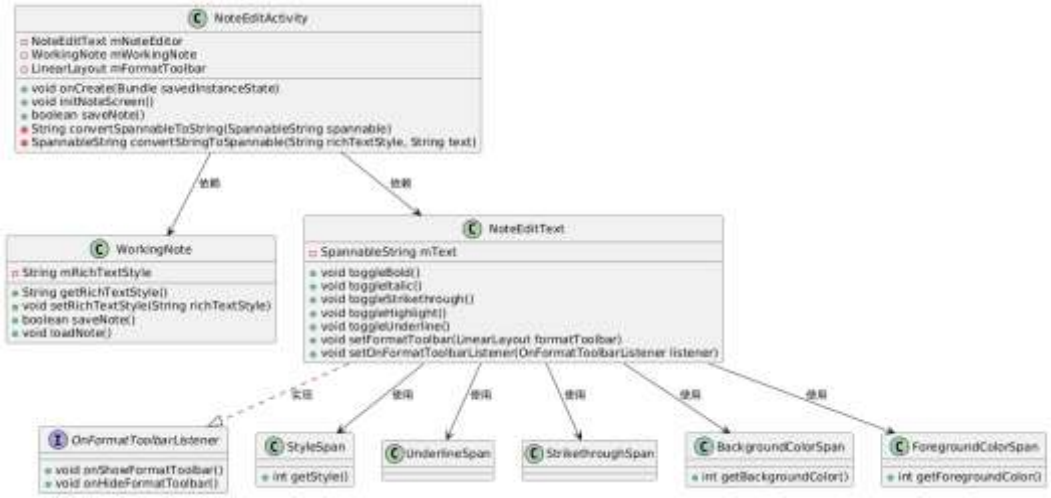


图 115 富文本功能类图

图 116 显示工具栏功能顺序图

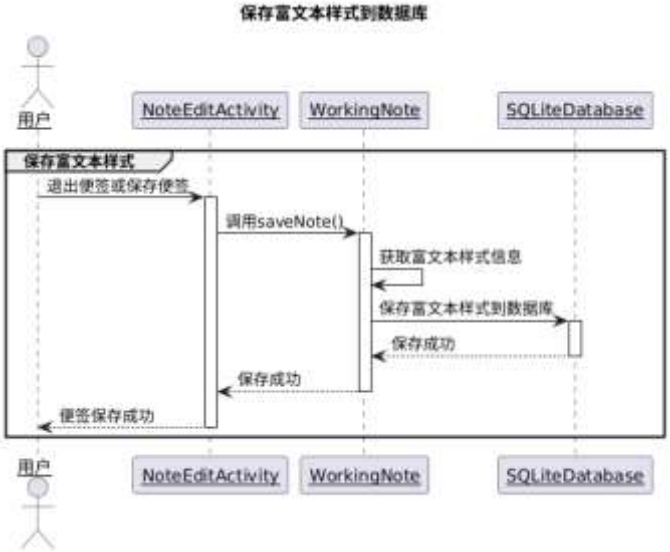


图 117 保存富文本功能顺序图

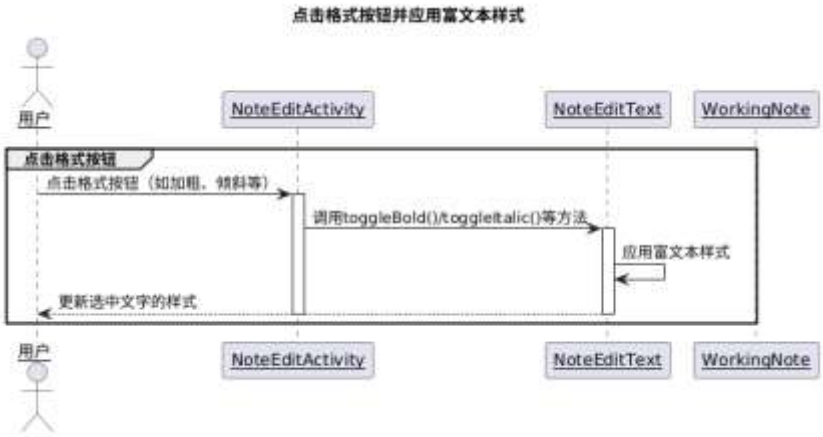


图 118 应用富文本功能顺序图

(三) 代码实现

1. 格式工具栏的初始化与事件绑定

在 NoteEditActivity 的 onCreate 方法中，初始化格式工具栏并绑定按钮的点击事件。每个按钮对应一种格式，点击按钮时调用 mNoteEditor 的相应方法对选中的文字应用格式。具体代码实现如图 119 所示。

```
// 初始化格式工具栏按钮
ImageButton btnBold = findViewById(R.id.btn_format_bold);
ImageButton btnItalic = findViewById(R.id.btn_format_italic);
ImageButton btnStrikethrough = findViewById(R.id.btn_format_strikethrough);
ImageButton btnHighlight = findViewById(R.id.btn_format_highlight);
ImageButton btnUnderline = findViewById(R.id.btn_format_underline); // 下划线按钮

// 加粗按钮点击事件
btnBold.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { mNoteEditor.toggleBold(); // 调用加粗方法 }
});

// 倾斜按钮点击事件
btnItalic.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { mNoteEditor.toggleItalic(); // 调用倾斜方法 }
});

// 删除线按钮点击事件
btnStrikethrough.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { mNoteEditor.toggleStrikethrough(); // 调用删除线方法 }
});
```

图 119 工具栏的初始化与事件绑定代码实现图

2. 格式的存储与加载

在保存便签时，将当前文本的富文本样式信息转换为字符串并存储到数据库中。具体代码实现如图 120 所示。

```
private boolean saveNote() {
    getWorkingText();
    mWorkingNote.setFont(mWorkingNote.getFont()); // 根据实际获取字体选择的逻辑获取字体值

    // 保存富文本样式信息
    SpannableString spannable = new SpannableString(mNoteEditor.getText());
    String richTextStyle = convertSpannableToString(spannable);
    mWorkingNote.setRichTextStyle(richTextStyle);

    boolean saved = mWorkingNote.saveNote();
    if (saved) {
        // 设置结果为成功，以便外部调用者知道保存操作的状态
        setResult(RESULT_OK);
    }
    return saved;
}
```

图 120 格式的存储与加载代码实现图

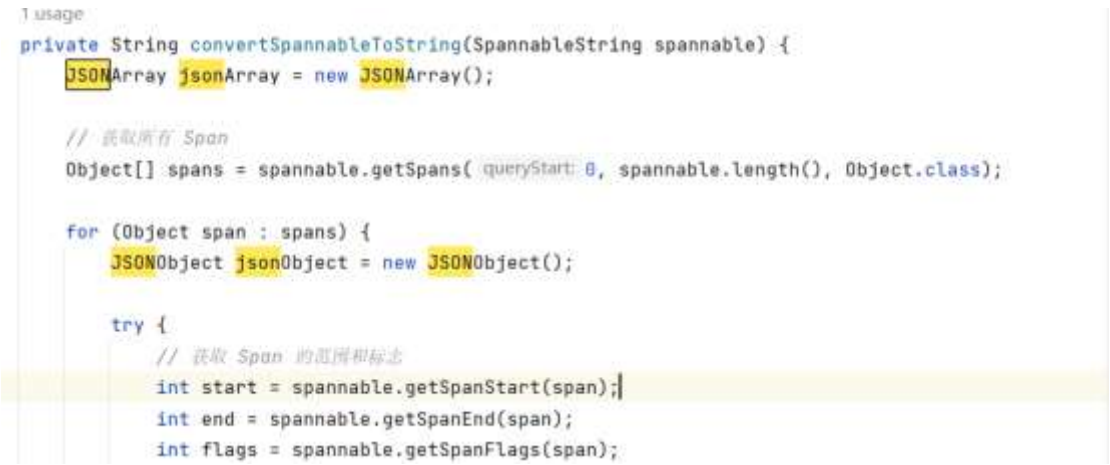
在加载便签时，从数据库中读取富文本样式信息并应用到文本中。具体代码实现如图 121 所示。

```
// 获取文本内容
String text = mWorkingNote.getContent();
// 获取富文本样式信息
String richTextStyle = mWorkingNote.getRichTextStyle();

// 将样式信息应用到文本
if (richTextStyle != null && !richTextStyle.isEmpty()) {
    SpannableString spannable = convertStringToSpannable(richTextStyle, text);
    mNoteEditor.setText(spannable);
} else {
    // 如果没有样式信息，直接显示普通文本
    mNoteEditor.setText(getHighlightQueryResult(text, mUserQuery));
}

// 设置光标位置
mNoteEditor.setSelection(mNoteEditor.getText().length());
```

图 121 获取富文本格式代码实现图

3. 格式的转换与解析

将 SpannableString 转换为 JSON 字符串以便存储。具体代码实现如图 122 所示。



```
1 usage
private String convertSpannableToString(SpannableString spannable) {
    JSONArray jsonArray = new JSONArray();

    // 获取所有 Span
    Object[] spans = spannable.getSpans( queryStart: 0, spannable.length(), Object.class);

    for (Object span : spans) {
        JSONObject jsonObject = new JSONObject();

        try {
            // 获取 Span 的范围和标志
            int start = spannable.getSpanStart(span);
            int end = spannable.getSpanEnd(span);
            int flags = spannable.getSpanFlags(span);
```

图 122 格式的转换代码实现图

将 JSON 字符串转换为 SpannableString 以便加载。具体代码实现如图 123 所示。

```
1 usage
private SpannableString convertStringToSpannable(String richTextStyle, String text) {
    SpannableString spannable = new SpannableString(text);

    if (richTextStyle == null || richTextStyle.isEmpty()) {
        return spannable; // 如果没有样式信息，直接返回普通文本
    }

    try {
        // 解析 JSON 字符串
        JSONArray jsonArray = new JSONArray(richTextStyle);

        // 遍历 JSON 数组，逐个应用样式
        for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject jsonObject = jsonArray.getJSONObject(i);
```

图 123 格式解析代码实现图

3.2.3.17 百度翻译功能

(一) 功能介绍

百度翻译功能为用户提供了一种便捷的方式，将便签中的文本内容翻译成多种语言。用户可以在便签编辑界面中点击翻译按钮，弹出翻译选项对话框，选择源语言和目标语言后，系统会自动调用百度翻译 API，将文本框中的内容翻译成目标语言并显示在编辑框中。该功能支持多种语言之间的互译，如中文与英文、中文与日语、中文与韩语等。通过这一功能，用户可以轻松实现跨语言的文本转换，提升便签的多语言处理能力。

(二) 功能详细设计

在便签编辑界面初始化时，系统会加载百度翻译功能所需的资源，并初始化百度翻译 API 的相关配置。用户点击翻译按钮后，系统会弹出翻译选项对话框，

用户可以选择源语言和目标语言。选择完成后，系统会调用百度翻译 API，将文本框中的内容发送至百度翻译服务器进行翻译。翻译完成后，系统会将翻译结果显示在编辑框中，并提示用户翻译成功。若翻译过程中出现错误，系统会提示用户翻译失败并显示错误信息。

百度翻译功能的实现依赖于百度翻译 API，通过 HTTP 请求将待翻译的文本发送至百度翻译服务器，并接收服务器返回的翻译结果。系统使用 OkHttp 库进行网络请求，并通过 Gson 库解析服务器返回的 JSON 格式数据。翻译结果的显示通过 UI 线程更新 EditText 控件的内容实现。为了确保翻译功能的准确性和高效性，系统还实现了签名生成机制，确保每次请求的安全性。

本功能涉及到的类图如图 124 所示。实现功能的顺序图如图 125 所示。

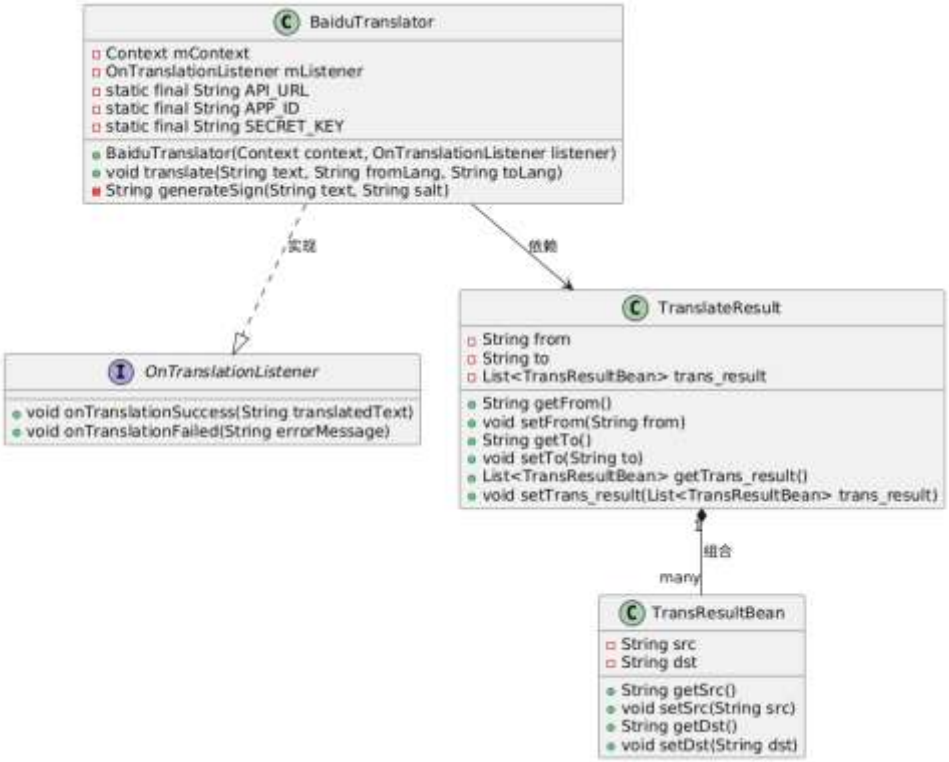


图 124 百度翻译功能类图

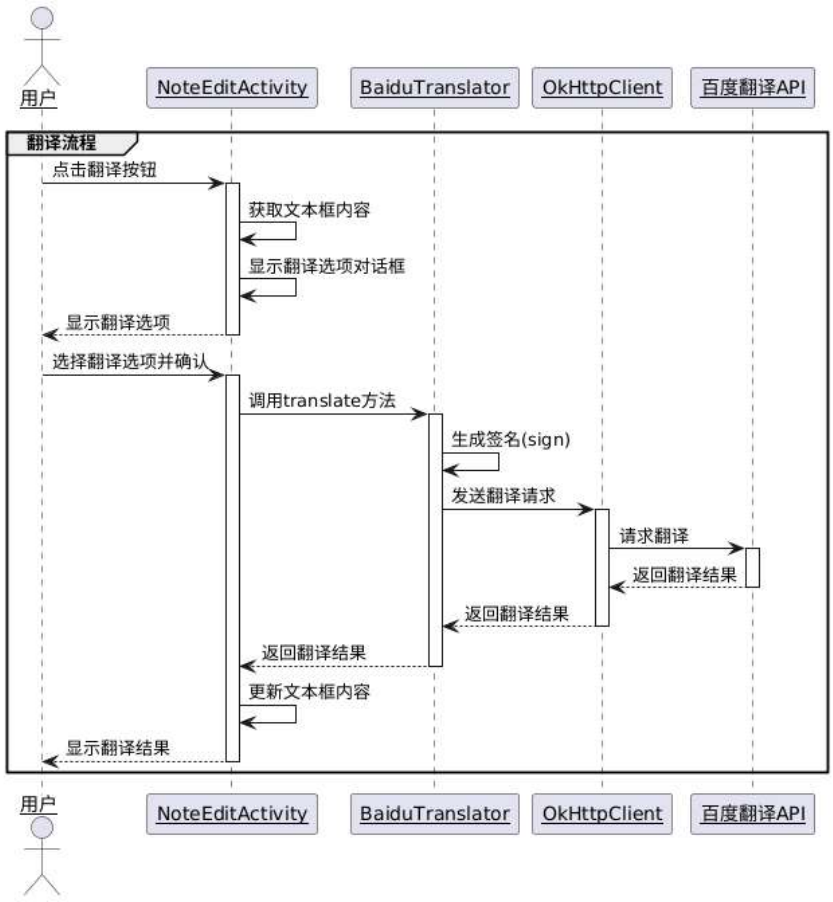


图 125 百度翻译功能顺序图

(三) 代码实现

在 NoteEditActivity.java 类中，百度翻译功能的实现主要分为以下几个步骤：

1. 初始化百度翻译功能：在 onCreate 方法中，初始化百度翻译功能的资源，包括创建 BaiduTranslator 对象，并设置翻译结果的回调监听器。当翻译成功或失败时，系统会通过回调函数更新 UI 界面。具体代码实现如所示。

```
//初始化翻译需要的实例
mBaiduTranslator = new BaiduTranslator( context: this, new BaiduTranslator.OnTranslationListener() {
    1 usage
    @Override
    public void onTranslationSuccess(String translatedText) {
        runOnUiThread(() -> {
            // 将翻译结果显示在 EditText 中
            mNoteEditor.setText(translatedText);
            Toast.makeText( context: NoteEditActivity.this,  text: "翻译成功", Toast.LENGTH_SHORT).show();
        });
    }

    3 usages
    @Override
    public void onTranslationFailed(String errorMessage) {
        runOnUiThread(() -> {
            Toast.makeText( context: NoteEditActivity.this,  text: "翻译失败: " + errorMessage, Toast.LENGTH_SHORT).show();
        });
    }
});
```

图 126 初始化百度翻译功能代码实现图

2. 显示翻译选项对话框：当用户点击翻译按钮时，系统会弹出翻译选项对话框，用户可以选择源语言和目标语言。对话框中的选项包括“中文 → 英文”、“英文 → 中文”、“中文 → 日语”等。具体代码实现如所示。

```
private void showTranslationDialog(String text) {
    // 翻译选项
    final String[] translationOptions = {
            "中文 → 英文",
            "英文 → 中文",
            "中文 → 日语",
            "日语 → 中文",
            "中文 → 韩语",
            "韩语 → 中文"
    };

    // 创建对话框
    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setTitle("选择翻译选项");
    builder.setItems(translationOptions, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // 根据用户选择的选项执行翻译
            String fromLang, toLang;
            switch (which) {
                case 0: // 中文 → 英文
                    fromLang = "zh";
                    toLang = "en";
                    break;
```

图 127 显示翻译选项对话框代码实现图

3. 调用百度翻译 API：在 BaiduTranslator 类中，系统通过 OkHttp 库发送 HTTP 请求至百度翻译 API，并处理服务器返回的翻译结果。翻译请求的签名通过 MD5 算法生成，确保请求的安全性。具体代码实现如所示。

```
public void translate(String text, String fromLang, String toLang) {
    String salt = String.valueOf(new Random().nextInt( bound: 10000));
    String sign = generateSign(text, salt);

    String url = API_URL +
            "?q=" + text +
            "&from=" + fromLang +
            "&to=" + toLang +
            "&appid=" + APP_ID +
            "&salt=" + salt +
            "&sign=" + sign;

    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder()
            .url(url)
            .build();

    client.newCall(request).enqueue(new Callback() {
        4 usages
        @Override
        public void onFailure(Call call, IOException e) {
            Log.e(TAG,  msg: "Translation failed: " + e.getMessage());
            mListener.onTranslationFailed(e.getMessage());
        }

        4 usages
        @Override
        public void onResponse(Call call, Response response) throws IOException {
            if (response.isSuccessful()) {
                String responseBody = response.body().string();
```

图 128 调用百度翻译 API 代码实现图

3.2.3.18 好友管理功能

(一) 功能介绍

管理好友功能是小米便签应用的核心社交功能之一，允许用户管理其社交关系网络。用户可以通过账号添加新好友，查看已添加的好友列表，以及删除不再需要的好友关系。当用户添加好友时，系统会自动验证目标账号的有效性，确保添加的是真实存在的用户。用户可以通过好友列表快速查看所有好友，并通过长按操作删除特定好友。该功能增强了应用的社交属性，为用户提供了便捷的社交关系管理方式。

(二) 功能详细设计

当用户点击"添加好友"按钮时，系统会弹出输入对话框，用户输入目标好友的账号。系统首先验证该账号是否存在，如果存在则将该用户添加为好友并更新好友列表，如果不存在则提示用户账号无效。添加成功后，系统会自动刷新好友列表以显示新添加的好友。

系统在主界面维护并显示当前用户的好友列表。每次进入应用或进行好友相关操作后，系统都会从数据库获取最新的好友列表并更新显示。好友列表以垂直线性布局的形式展示，每个好友项包含好友的账号信息。

用户可以通过长按好友列表中的好友项触发删除操作。系统会弹出确认对话框，用户确认后系统将从数据库中删除该好友关系，并自动更新好友列表。删除操作采用异步方式处理，确保主线程的流畅响应。

好友关系数据存储在远程数据库中，通过 DBConnection 类管理数据访问。主要包含好友关系表，存储用户账号与好友账号的对应关系，支持好友的添加、查询和删除操作。管理好友功能的类图如图 129 所示，该功能的顺序图如图 130 所示。

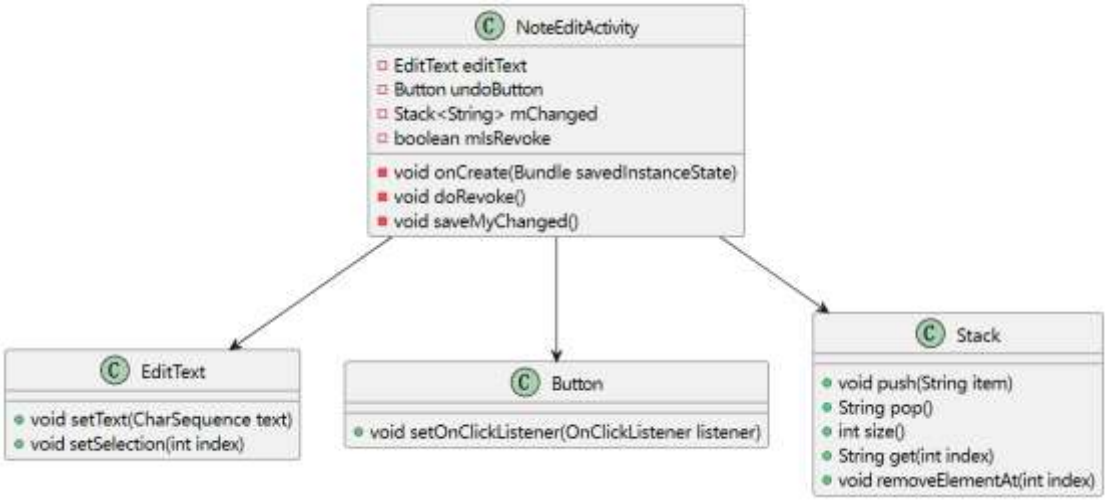


图 129 管理好友功能类图

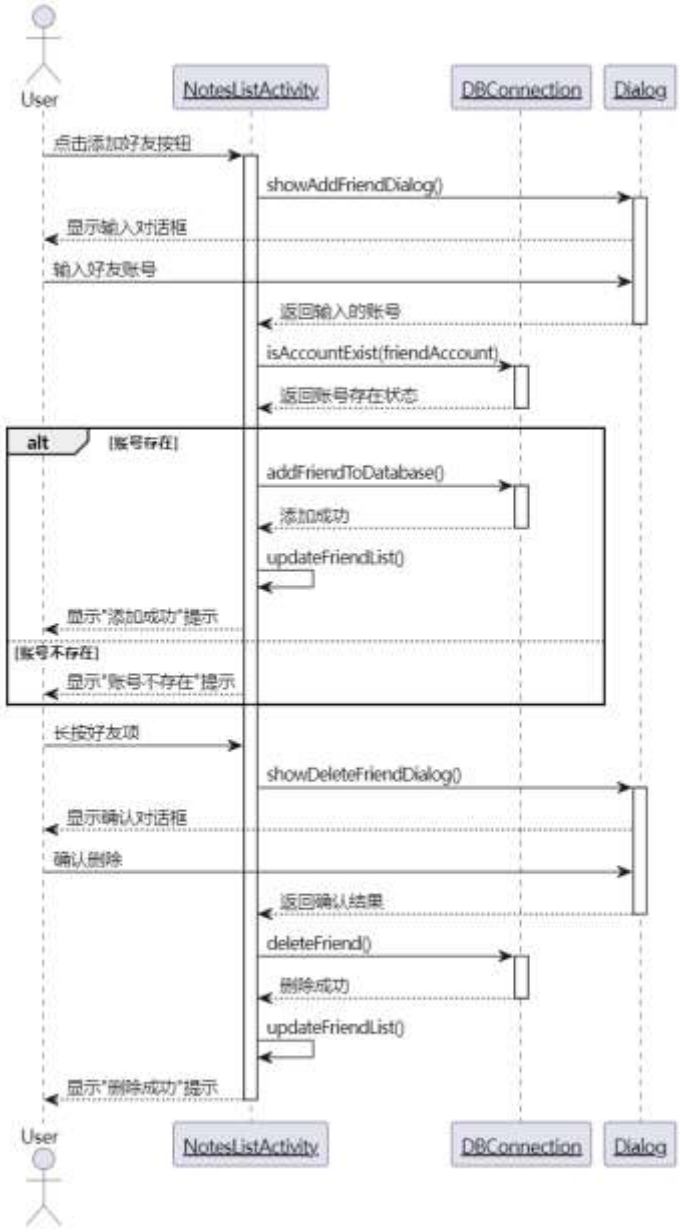


图 130 管理好友功能顺序图

(三) 代码实现

管理好友功能的代码实现包括添加好友，删除好友，以及列表更新这几个核心的功能。下面我将对管理好友功能的代码实现进行详细的介绍。

在好友界面展示管理：在 NotesListActivity 中，系统通过 showAddFriendDialog() 方法创建并显示添加好友对话框。该方法使用 AlertDialog.Builder 构建一个包含输入框和确认/取消按钮的对话框，如图 131 所示。当用户点击确认按钮时，系统调用 addFriend() 方法处理添加好友请求。如图 132 所示。

```
public void showAddFriendDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("添加好友");

    // 设置输入框
    final EditText input = new EditText(this);
    input.setInputType(InputType.TYPE_CLASS_TEXT);
    builder.setView(input);

    // 设置按钮
    builder.setPositiveButton("添加", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            String friendAccount = input.getText().toString();
            addFriend(friendAccount);
        }
    });
    builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
        }
    });

    builder.show();
}
```

图 131 添加好友功能系统提示代码实现图

好友关系验证与添加：在添加好友过程中，系统首先通过 DBConnection 的 isAccountExist() 方法验证目标账号是否存在。如果账号存在，则调用 addFriendToDatabase() 方法将好友关系添加到数据库，并通过 Toast 提示添加成功，如图 132 所示。

```
public void addFriend(String friendAccount) {
    DBConnection.isAccountExist(friendAccount, new DBConnection.AccountExistCallback() {
        @Override
        public void onResult(boolean exists) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    if (exists) {
                        DBConnection.addFriendToDatabase(USERACCOUNT, friendAccount, new Runnable() {
                            @Override
                            public void run() {
                                runOnUiThread(new Runnable() {
                                    @Override
                                    public void run() {
                                        Toast.makeText(NotesListActivity.this, "好友"+friendAccount+"已添加", Toast.LENGTH_SHORT).show();
                                        updateFriendList(); // Update the friend list
                                    }
                                });
                            }
                        });
                    } else {
                        Toast.makeText(NotesListActivity.this, "账户不存在", Toast.LENGTH_SHORT).show();
                    }
                }
            });
        }
    });
}
```

图 132 添加好友功能代码实现图

好友删除处理：当用户长按好友项时，系统通过 showDeleteFriendDialog()

方法显示删除确认对话框。用户确认后，调用 DBConnection 的 deleteFriend() 方法从数据库中删除好友关系，并刷新好友列表，如图 133 所示。

```
public void showDeleteFriendDialog(final String friendAccount) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("删除好友");
    builder.setMessage("确定要删除好友 " + friendAccount + " 吗？");

    builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            DBConnection.deleteFriend(USERACCOUNT, friendAccount, new Runnable() {
                @Override
                public void run() {
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            updateFriendList();
                            Toast.makeText(NotesListActivity.this, "好友 " + friendAccount + "删除成功", Toast.LENGTH_SHORT).show();
                        }
                    });
                }
            });
        }
    });
    builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
        }
    });

    builder.show();
}
```

图 133 删除好友功能代码实现图

好友列表管理：updateFriendList() 方法负责从数据库获取并更新好友列表显示。该方法通过 DBConnection 的 getFriendList() 获取好友数据，然后在 UI 线程中动态创建和更新好友列表视图，如图 134 所示。每个好友项都支持长按删除操作。

```
public void updateFriendList() {
    DBConnection.getFriendList(USERACCOUNT, new DBConnection.FriendListCallback() {
        @Override
        public void onResult(final List<String> friendList) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    LinearLayout friendListLayout = findViewById(R.id.friend_list_layout);
                    friendListLayout.removeAllViews();

                    for (String friend : friendList) {
                        TextView friendTextView = new TextView(NotesListActivity.this);
                        friendTextView.setText(friend);
                        friendTextView.setTextSize(18); // Set font size
                        friendTextView.setGravity(Gravity.CENTER); // Center text
                        friendTextView.setPadding(0, 20, 0, 20); // Add padding for better spacing
                        friendTextView.setLayoutParams(new LinearLayout.LayoutParams(
                                LinearLayout.LayoutParams.MATCH_PARENT,
                                LinearLayout.LayoutParams.WRAP_CONTENT
                        ));
                        friendTextView.setOnLongClickListener(new View.OnLongClickListener() {
                            @Override
                            public boolean onLongClick(View v) {
                                showDeleteFriendDialog(friend);
                                return true;
                            }
                        });
                        friendListLayout.addView(friendTextView);
                    }
                }
            });
        }
    });
}
```

图 134 更新好友列表代码实现图

数据库操作实现：DBConnection 类负责管理与 MySQL 数据库的连接和数据操作。通过 getConnection() 方法建立数据库连接，并提供了添加好友、删除好友、查询好友列表等方法，确保数据的安全存储和访问，如图 135 图 136 图 137 所示。

```
// 添加好友
public static void addFriendToDatabase(final String userAccount, final String friendAccount, final R
    new Thread(new Runnable() {
        @Override
        public void run() {
            Connection conn = null;
            PreparedStatement preparedStatement = null;
            String sqlUpdate = "UPDATE xiaomi_serve.note SET friend_account = CONCAT(IFNULL(friend_a

            try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);

                preparedStatement = conn.prepareStatement(sqlUpdate);
                preparedStatement.setString(1, friendAccount + ",");
                preparedStatement.setString(2, userAccount);
                preparedStatement.executeUpdate();
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                closeResources(conn, preparedStatement, null);
                if (callback != null) {
                    callback.run();
                }
            }
        }
    }).start();
}
```

图 135 添加好友功能数据库操作代码实现图

```
// 获取好友列表
public static void getFriendList(final String userAccount, final FriendListCallback callback) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            Connection conn = null;
            PreparedStatement preparedStatement = null;
            ResultSet resultSet = null;
            List<String> friendList = new ArrayList<>();
            String sqlSelect = "SELECT friend_account FROM xiaomi_serve.note WHERE account = ?";

            try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);

                preparedStatement = conn.prepareStatement(sqlSelect);
                preparedStatement.setString(1, userAccount);
                resultSet = preparedStatement.executeQuery();

                if (resultSet.next()) {
                    String friendAccounts = resultSet.getString("friend_account");
                    if (friendAccounts != null && !friendAccounts.isEmpty()) {
                        String[] friends = friendAccounts.split(",");
                        Collections.addAll(friendList, friends);
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                closeResources(conn, preparedStatement, resultSet);
                if (callback != null) {
                    callback.onResult(friendList);
                }
            }
        }
    }).start();
}
```

图 136 获取好友列表数据库操作代码实现图

```
// 删除好友
public static void deleteFriend(final String userAccount, final String friendAccount, final Runnable
    new Thread(new Runnable() {
        @Override
        public void run() {
            Connection conn = null;
            PreparedStatement preparedStatement = null;
            ResultSet resultSet = null;
            String sqlSelect = "SELECT friend_account FROM xiaomi_serve.note WHERE account = ?";
            String sqlUpdate = "UPDATE xiaomi_serve.note SET friend_account = ? WHERE account = ?";

            try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);

                // 获取当前好友列表
                preparedStatement = conn.prepareStatement(sqlSelect);
                preparedStatement.setString(1, userAccount);
                resultSet = preparedStatement.executeQuery();

                if (resultSet.next()) {
                    String friendAccounts = resultSet.getString("friend_account");
                    if (friendAccounts != null && !friendAccounts.isEmpty()) {
                        // 移除要删除的好友
                        String updatedFriends = friendAccounts.replace(friendAccount + ",", "");

                        // 更新好友列表
                        preparedStatement = conn.prepareStatement(sqlUpdate);
                        preparedStatement.setString(1, updatedFriends);
                        preparedStatement.setString(2, userAccount);
                        preparedStatement.executeUpdate();
                    }
                }
```

图 137 删除好友功能数据库操作代码实现图

3.2.3.19 转发便签功能

(一) 功能介绍

转发便签功能允许用户将当前编辑的便签内容分享到其他用户。这一功能增强了便签的协作和分享能力，使用户能够更方便地与他人交流和分享重要信息。用户可以通过选择目标好友，将便签内容发送出去，实现信息的快速传播和共享。该功能提升了应用的交互性和实用性，使便签不仅仅是个人记录的工具，也成为社交互动的媒介。

(二) 功能详细设计

对于转发便签功能，用户在便签列表中选择一条便签，并点击转发菜单项（menu_forward）。当用户点击转发按钮时，系统弹出一个好友选择对话框，显示当前用户的好友列表。用户选择一个作为转发目标，之后，系统将自动执行转发，系统调用 DBConnection.forwardNoteToFriend 方法，将便签内容发送给所选好友。转发过程中，系统会显示处理转发结果，给出反馈。转发成功后，系统显示“便签已成功转发”消息。如果转发失败，系统提示相应的错误信息。转发便签功能的顺序图如图 138 所示，该功能的类图如图 139 所示。

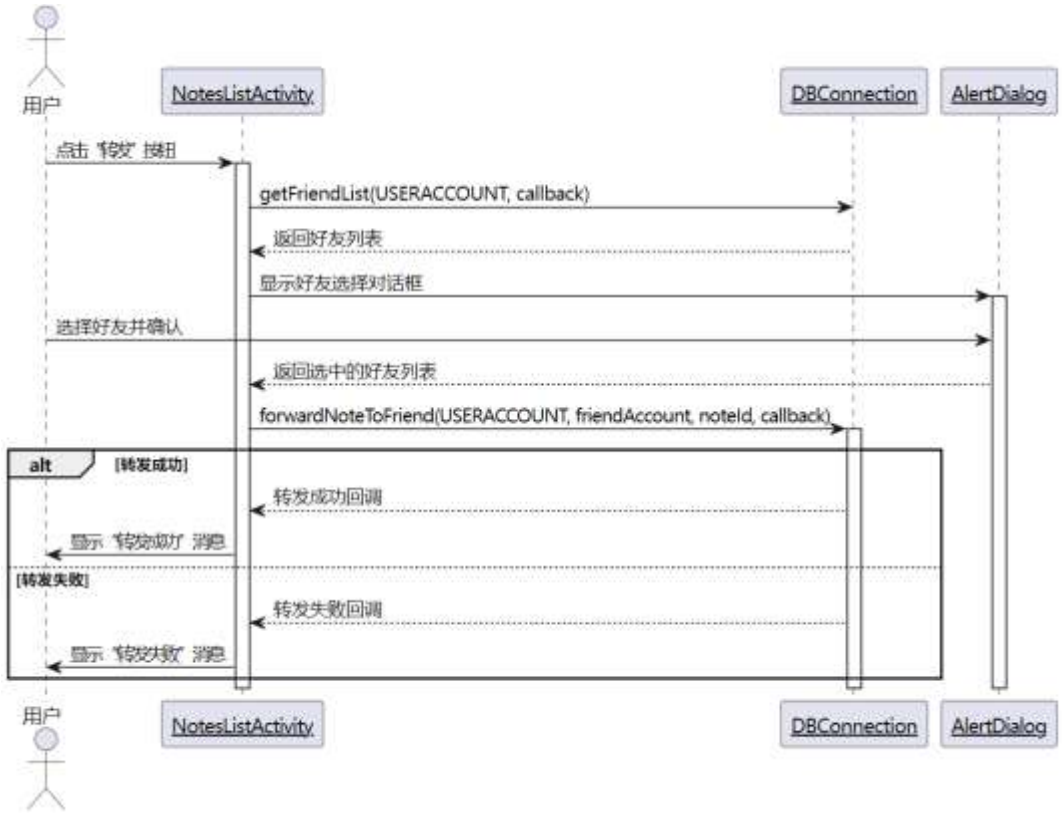


图 138 转发便签功能顺序图

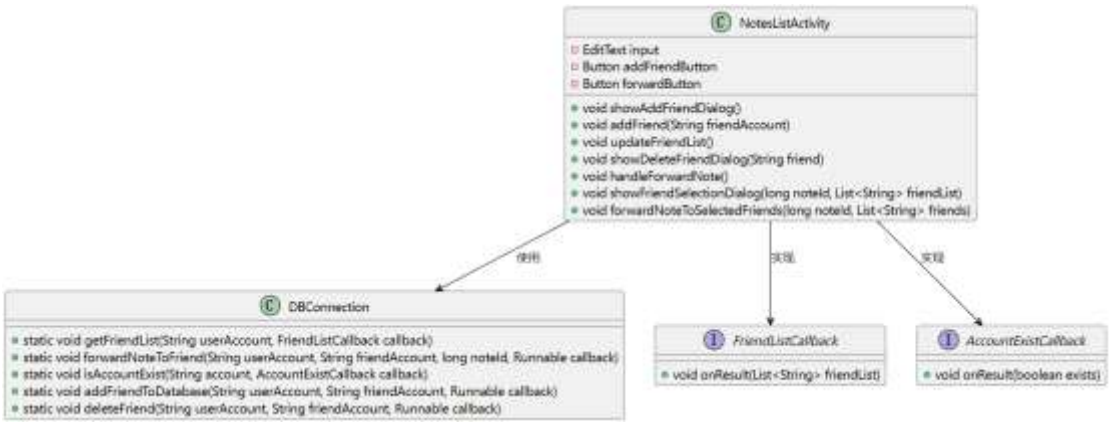


图 139 转发便签功能类图

(三) 代码实现

在转发便签功能的实现过程中，我们将重点介绍以下几个关键部分的代码实现

转发便签界面交互实现：在 NotesListActivity 中，系统通过菜单项触发转发操作。当用户点击转发按钮时，会调用 handleForwardNote()方法，该方法负责获取好友列表并显示好友选择对话框，如图 140 所示。该实现包含了对话框的创建、好友列表的展示以及用户选择的处理逻辑。

```
public class NoteEditActivity extends Activity implements OnClickListener,
    public boolean onOptionsItemSelected(MenuItem item) {
                break;
            case R.id.menu_forward:
                //转发
                forward();
                break;
            default:
                break;
        }
        return true;
    }


    private void forward() {
        System.out.println(mWorkingNote.getNoteId());
        showForwardDialog(mWorkingNote.getNoteId());

    }

    public void showForwardDialog(final long noteId) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("选择好友");

        // Get the list of friends
        DBConnection.getFriendList(USERACCOUNT, new DBConnection.FriendListCallback() {
            @Override
            public void onResult(final List<String> friendsList) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        final String[] friendsArray = friendsList.toArray(new String[0]);

                        builder.setItems(friendsArray, new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                String selectedFriend = friendsArray[which];
                                forwardNoteToFriend(noteId, selectedFriend);
                            }
                        });

                        builder.setNegativeButton("取消", new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                dialog.cancel();
                            }
                        });
```

图 140　转发便签界面交互代码实现图

转发结果系统响应：在完成转发操作后，系统会通过 Toast 消息向用户提供即时反馈。在 NotesListActivity 中，forwardNoteToSelectedFriends 方法会根据转发操作的结果显示相应的提示信息，如图 141 所示。该实现确保了用户能够清楚地知道转发操作的结果。

```
public void forwardNoteToFriend(final long noteId, final String friendAccount) {
    new Thread(new Runnable() {
        @Override
        public void run() {
            DBConnection.forwardNoteToFriend(USERACCOUNT, friendAccount, noteId, new Runnable() {
                @Override
                public void run() {
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            Toast.makeText(NoteEditActivity.this, "便签已转发给 " + friendAccount, Toast.LENGTH_SHORT).show();
                        }
                    });
                }
            });
        }
    }).start();
}
```

图 141 转发结果系统响应提示代码实现图

数据库转发逻辑：在 DBConnection 类中，forwardNoteToFriend 方法负责处理便签转发的具体数据库操作，如图 142 所示。该方法首先从源用户的数据表中查询便签内容，然后将内容插入到目标好友的数据表中。实现包括：查询原始便签数据、生成新的便签 ID、复制便签内容到好友的数据表、复制相关的数据记录、处理异常情况。

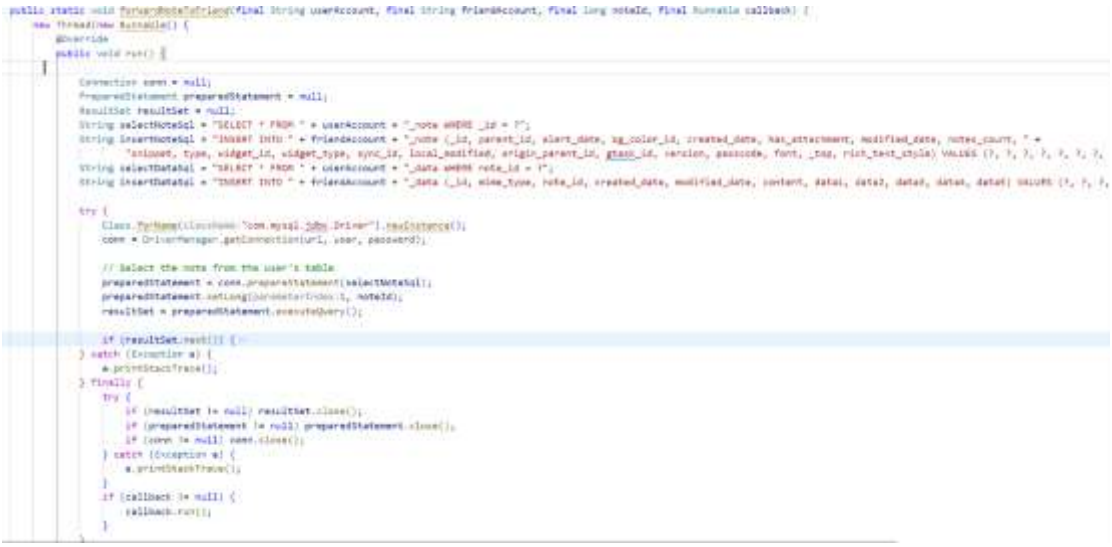


图 142 数据库转发逻辑代码实现图

3.3 维护代码数量及其质量情况

在小米便签应用的维护过程中，我们新增了多个功能，以增强应用的功能性和用户体验。表 5 详细列出了这些新增功能的名称及其对应的代码行数。

通过这些新增功能，我们不仅提升了小米便签应用的实用性和便捷性，还注重了代码的质量和可维护性。在编写代码过程中，我们遵循了良好的编程规范和命名约定，进行了代码审查和测试，以确保代码的质量和稳定性。

表 5. 开源软件维护的代码分析

序号	维护类别	名称	受影响的代码行数
1	新增功能	登录/注册	592 行
2	新增功能	云端存储	100 行
3	新增功能	上传云端	300 行
4	新增功能	载入本地	200 行
5	新增功能	置顶	132 行

6	新增功能	更换背景	35 行
7	新增功能	插入图片	232 行
8	新增功能	修改字体	133 行
9	新增功能	开场动画	117 行
10	新增功能	便签搜索	206 行
11	新增功能	九宫格密码锁	780 行
12	新增功能	字符统计	67 行
13	新增功能	文本撤回	116 行
14	新增功能	朗读文本	328 行
15	新增功能	大模型交互	509 行
16	新增功能	富文本编辑	314 行
17	新增功能	百度翻译	255 行
18	新增功能	好友管理	479 行
19	新增功能	转发便签	130 行

3.4 维护后的软件原型

3.4.1 登录和注册功能

本小组通过维护小米便签，新增了登录和注册的功能。登录功能具体的操作界面如图 81 所示。对于登录功能，用户点击小米便签后，在通过欢迎界面之后进入到 begin_viewpro.xml 的界面，该界面包含两个信息输入框和两个按钮。用户欲使用登录功能的时候，如果用户存在账户，则只需在第一个输入框中写入账户，在第二个输入框中写入密码，点击登录按钮即可进入便签主页面。如果用户输入账户后，系统在数据库中找不到该账户，系统会自动对用户进行系统提示，提示用户的账户不存在，并询问是否需要进行注册。如果用户选择确认按钮，则系统自动切换到注册功能模块。如果用户是密码输入错误，选择取消按钮，则会回到登录界面。在登录功能中，我们精心设计了详细的系统提示，以便用户能够方便的使用我们的软件系统。不仅如此，在密码的输入框，为了保障用户的个人隐私，我们对用户的输入置为“*”由此来防止用户的密码泄露，引发安全的问

题。具体界面如图 143 所示。

图 143 登录功能界面图

对于注册功能，用户欲使用注册功能，可以点击注册按钮，系统会自动跳到注册模块。或者在登录失败后，用户也可以选择进入注册界面进行注册账户。在注册界面有三个输入框和两个按钮部件。用户在账户输入框中输入要注册的账户，在密码输入框输入账户的密码，并在第三个确认密码输入框中输入重复的密码。最后点击注册按钮，用户便可完成注册账户。在整个注册的过程，我们为了保障用户的隐私，对密码部分进行加密处理，将输入的值转换成“*”，来防止用户的隐私泄露。点击注册之后，如果注册成功，系统会自动跳转到新的登录界面，用户将刚刚注册好的账户输入信息输入框中，完成登录账户，即可进入主界面，使用用户个性化的便签。具体界面如图 144 所示。

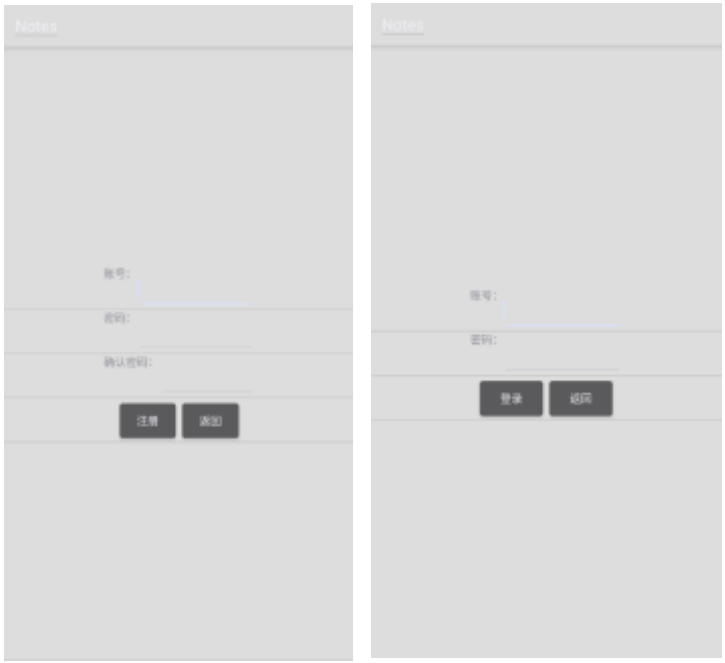


图 144 注册功能界面设计图

3.4.2 云端存储功能

本功能主要是面向系统管理者，系统管理者为了管理软件系统的数据信息，可以通过连接远程的云数据库（xiaomi_serve）进行用户信息管理，在数据库中，我们保存了整个系统的用户基本信息表（note 表），并且在用户注册的时候，自动创建用户的“账户_note”和“账户_data”表。用户的数据信息将主要保存在这两个表中。开发者也可以通过对列表项的增删改进行功能的维护。具体界面如图 145 所示。

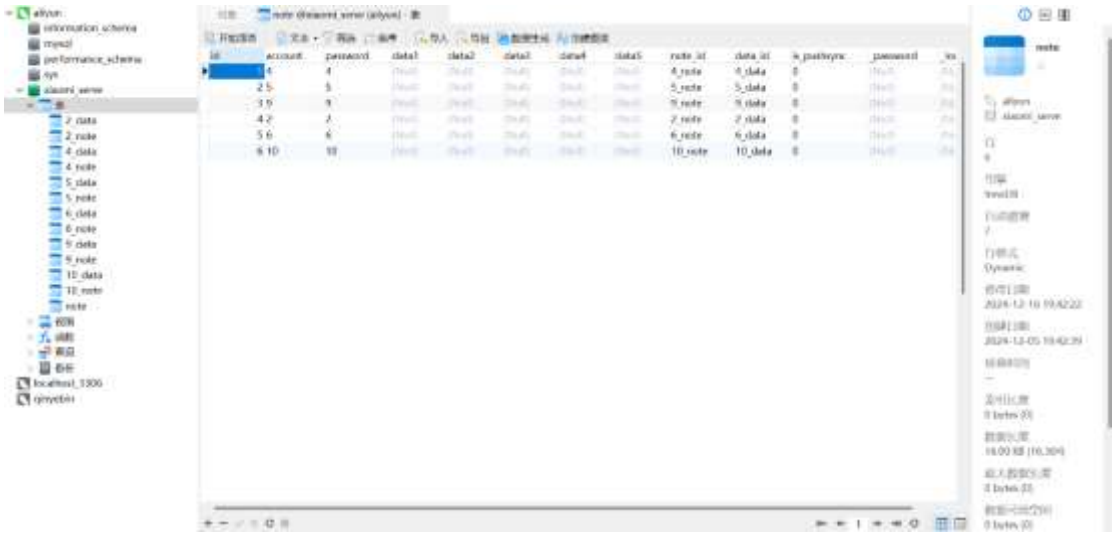


图 145 云端存储功能管理界面图

3.4.3 上传云端功能

对于本功能，本小组在菜单栏部分添加了上传云端按钮，用户通过手动点击上传按钮，系统自动将用户的 note 表信息和 data 表信息复制到云数据库系统中。完成数据的上传云端工作。具体界面如图 146 所示。

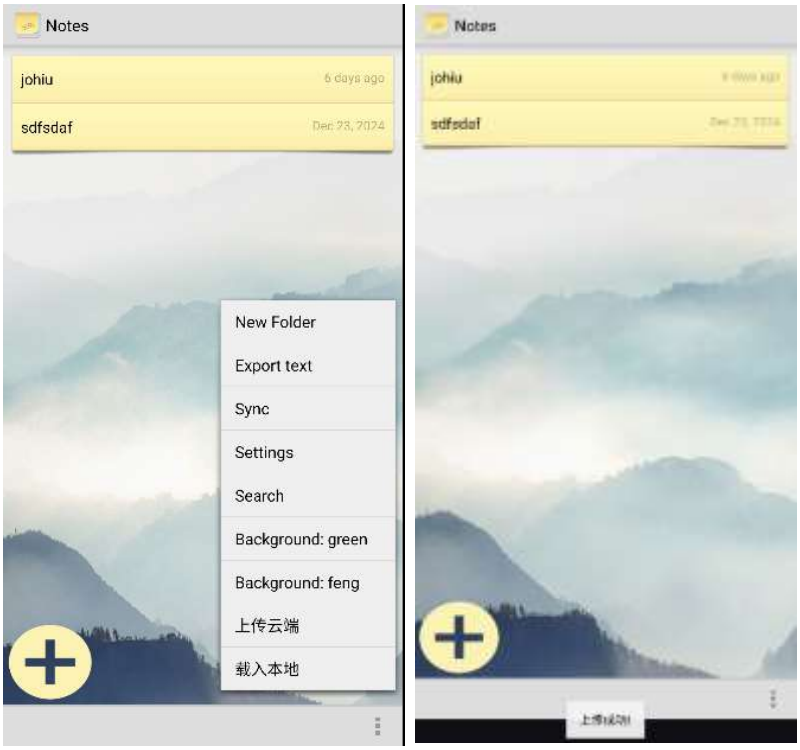


图 146 上传云端功能界面图

3.4.4 载入本地功能

载入本地功能，首先在用户登录成功之后，系统会自动地运行载入本地功能模块。除此之外，本小组也在菜单栏部分添加了手动载入本地的按钮，用户通过点击“载入本地”，系统会弹出系统提示，提示用户载入本地的警告信息。点击确认后，系统自动将云端的数据，载入本地，实现数据的同步。具体功能界面图如图 77。

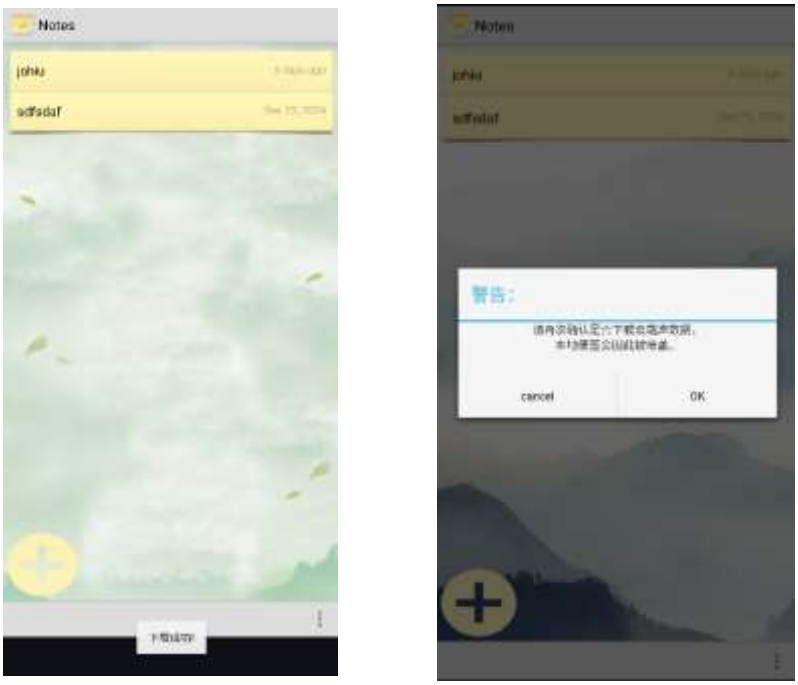


图 147 载入本地功能界面图

3.4.5 置顶功能

本小组在便签的菜单中添加置顶功能模块，用户欲使用置顶功能，只需要点击“置顶”按钮，系统自动将便签置顶，并在便签列表中将所设置的便签置顶，同时显示置顶图标。如果该便签已经置顶，该位置会显示“取消置顶”，如果用户希望取消置顶便签，点击“取消置顶”，系统则会恢复便签位置。具体的界面如图 148 所示。

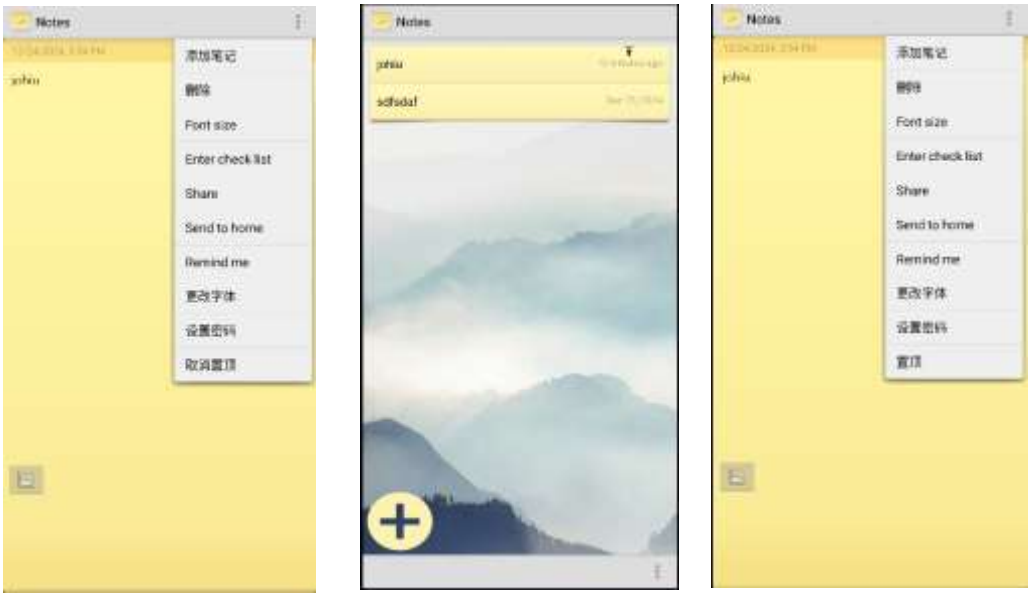


图 148 置顶和取消置顶功能界面图

3.4.6 更改背景功能

在小米便签应用中，用户界面设计注重用户体验与直观操作。应用启动时，主界面默认展示预设背景图。在`NotesListActivity`界面，用户通过点击菜单按钮即可访问背景更换功能。该功能提供了一组按钮，每个按钮代表不同的背景选项，。用户只需直接点击相应的按钮，即可切换至所选背景。如图 149 所示。

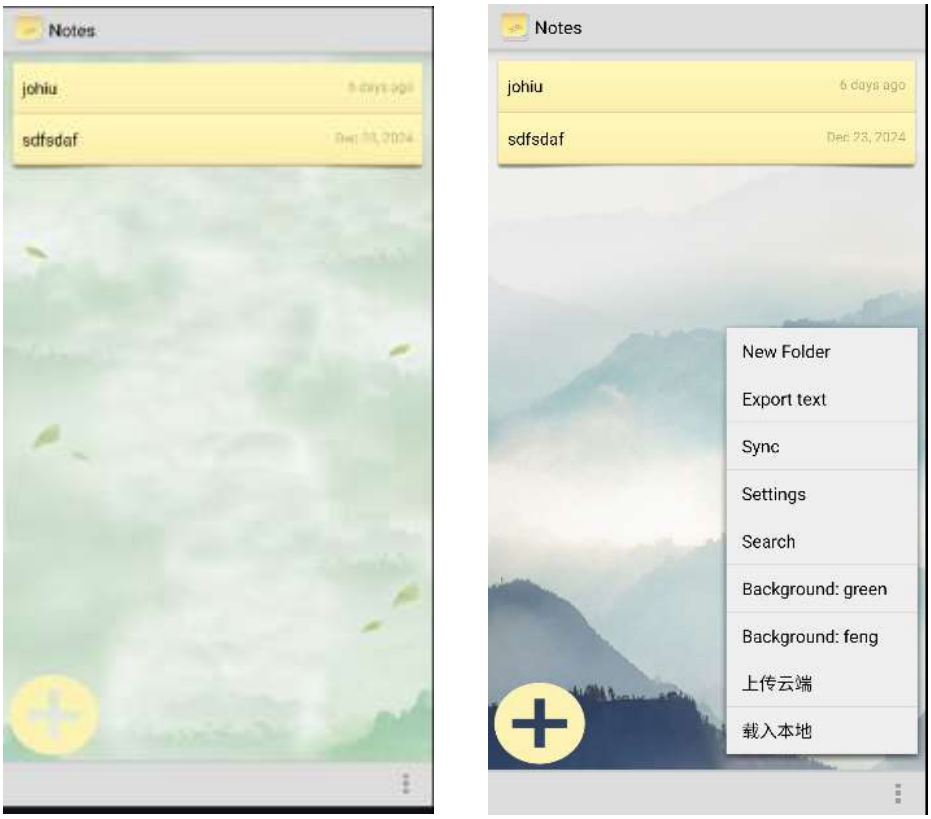


图 149 更改背景功能界面图

3.4.7 便签搜索功能

小米便签应用的便签搜索功能为用户提供了一种高效的便签定位机制。用户在`NotesListActivity`界面通过点击搜索按钮激活搜索对话框，输入关键字后提交，系统随即触发一个包含搜索关键字的 Intent。依据应用配置，`SearchActivity`被启动以处理该搜索请求。在此活动中，系统通过提取 Intent 中的搜索关键字，对`NotesProvider`数据库进行查询，筛选出含有关键字的便签项。查询结果以 Cursor 对象形式返回，并绑定至 ListView 控件，由`SearchAdapter`负责将搜索结果呈现给用户。用户点击搜索结果中的特定便签，即可直接跳转至`NoteEditActivity`进行查看或编辑。整个流程优化了用户查找便签的效率，并增强了应用的交互性和实用性。界面设计如图 150 所示。

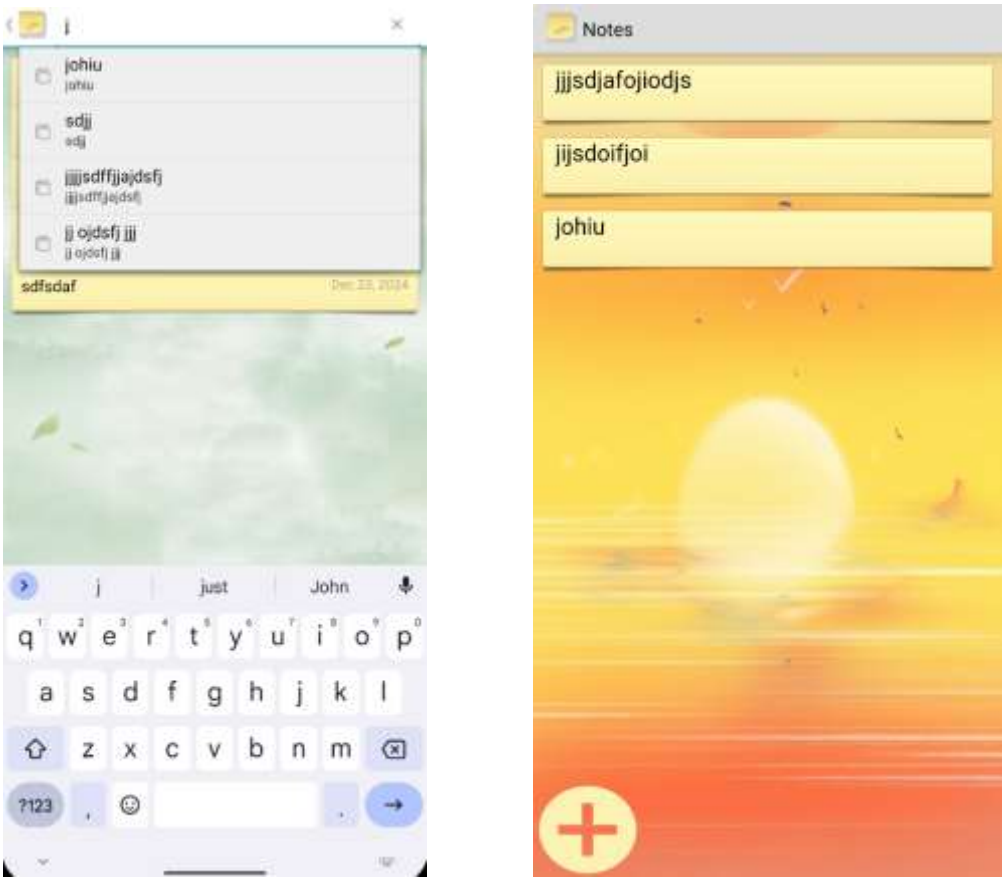


图 150 便签搜索功能界面图

3.4.8 开场动画功能

我们给小米便签增加了一个开场动画功能，它在用户启动应用时立即呈现，以一个持续 3 秒的动态展示迎接用户。动画结束后，用户将被引导至主界面。界面设计如图 151 所示。

图 151 便签开场动画功能界面图

3.4.9 字体更改功能

在小米便签应用中，用户个性化字体设置的功能允许用户根据个人喜好调整便签的字体样式。默认情况下，新建便签时应用采用宋体字体，但用户可以通过在`NoteEditActivity`编辑界面点击菜单按钮并选择“更改字体”选项来访问字体选择对话框。在此对话框中，用户可以选择多种预设字体样式，选择后，所选字体的标识信息将被存储于数据库的`font`列中。当便签再次打开时，应用将自动从数据库读取`font`列的值，并应用相应的字体样式，从而确保用户的个性化选择得到持续的体现。界面设计如图 152 所示。

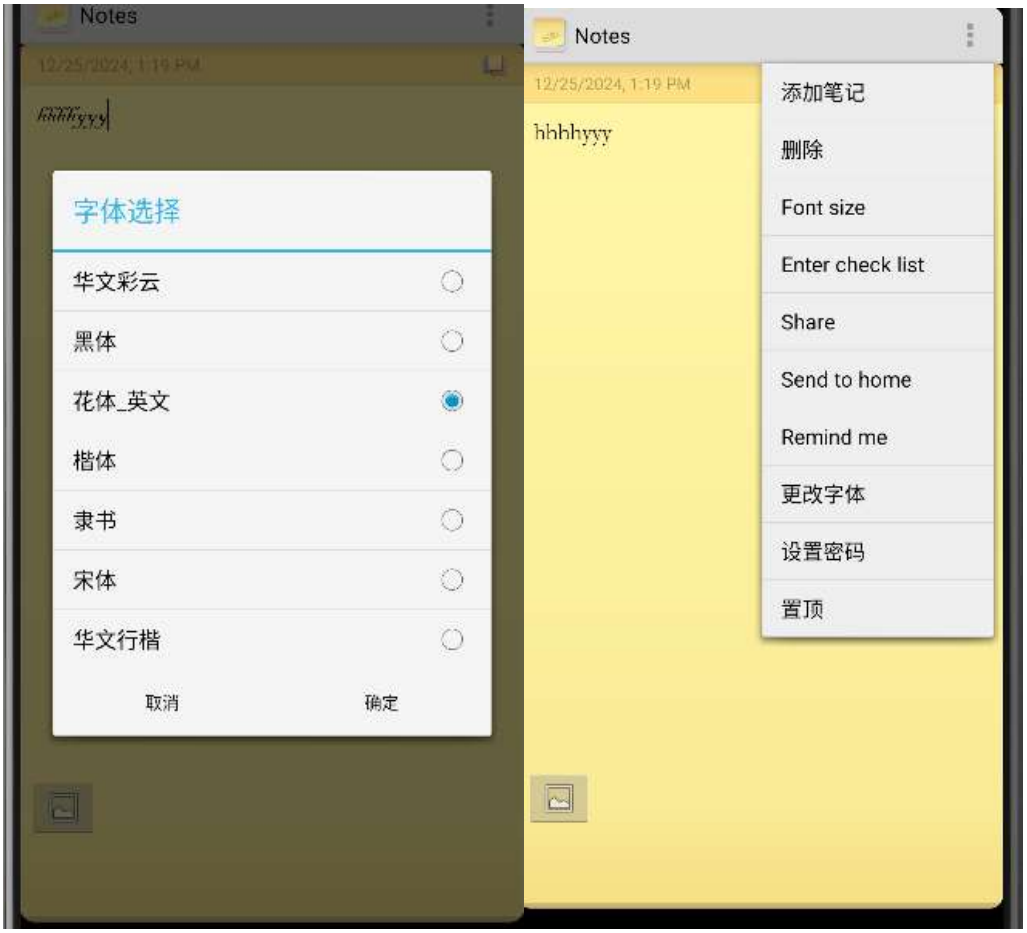


图 152 便签修改字体功能界面图

3.4.9 九宫格手势密码锁功能

小米便签应用中集成的手势密码锁功能，旨在为用户数据的隐私性和安全性提供强有力的保障。当用户首次启动应用并进入主界面时，系统会自动对便签列表进行遍历，并检查每个便签项的加密状态。对于未加密的便签，用户可以无障碍地查看其标题和内容；而对于那些已经加密的便签，系统则仅显示“已加密”的提示，并辅以锁形图标，以此直观地向用户表明该便签处于保护状态。界面设

计如图 153 所示。

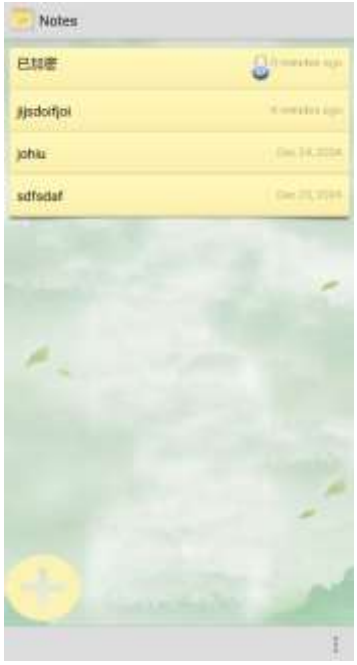


图 153 便签加密后界面图

在用户尝试访问便签列表中的某个便签项时，系统会判断该便签是否已被锁定。若便签未加密，用户将直接进入 NoteEditActivity 进行编辑操作；若便签已加密，则用户将被重定向至 UnlockActivity 以进行解锁。在 UnlockActivity 中，用户需输入手势密码，若输入正确，系统将显示绿色提示并允许用户进入 NoteEditActivity；若输入错误，则系统将显示红色提示并返回主界面。界面设计如图 154 所示。

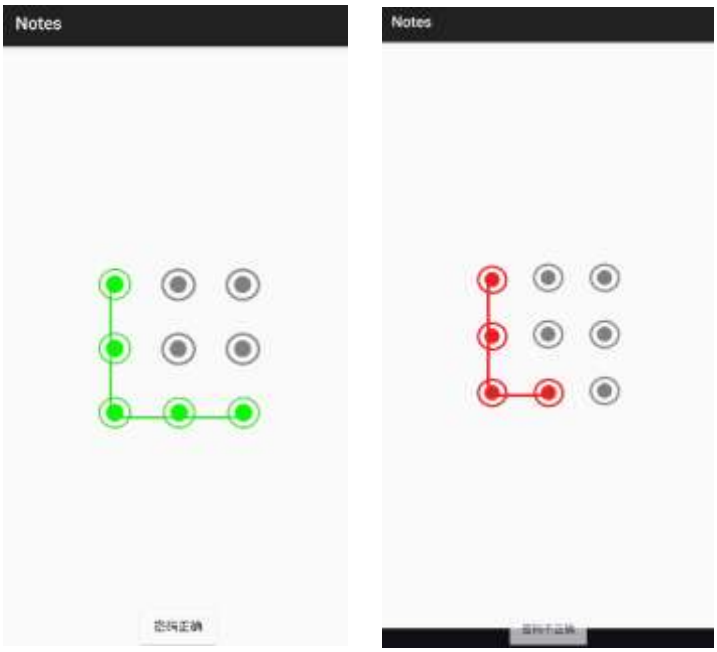


图 154 便签解锁界面图

在 NoteEditActivity 中，用户可以通过点击右上角的菜单按钮来管理便签的加密状态。对于未加密的便签，菜单中将提供“设置密码”选项，用户可通过此选项跳转至 SetLockActivity 以设置新的手势密码。在 SetLockActivity 中，用户完成密码输入并确认后，系统将新密码存储于数据库中，从而完成加密设置。对于已加密的便签，菜单中则提供“删除密码”选项，用户点击后，系统将清除数据库中该便签的密码信息，便签随即解锁，并在 NoteListActivity 中以正常状态显示。界面设计如图 155 图 156 所示。

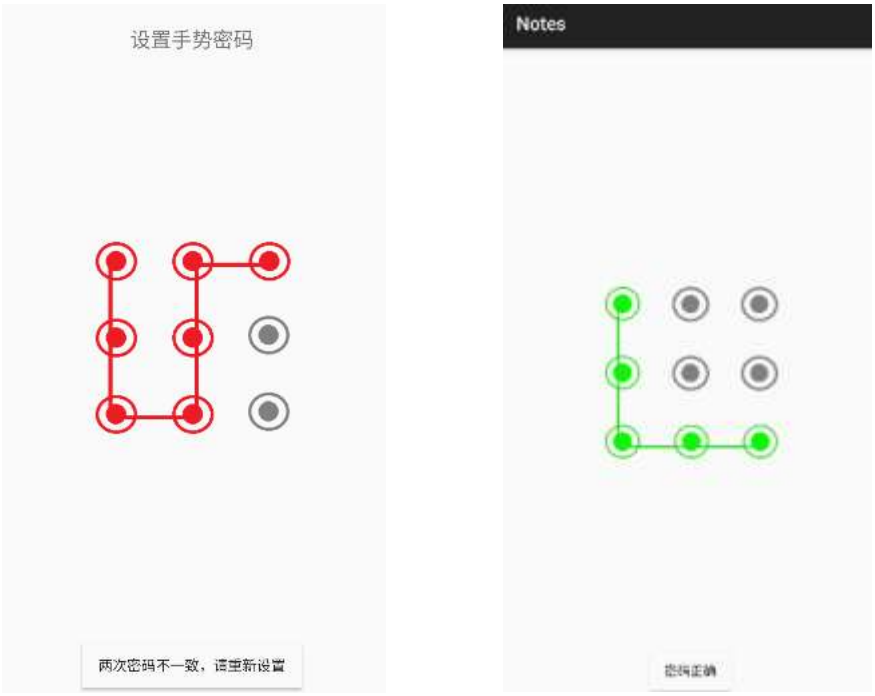


图 155 便签设置密码界面图

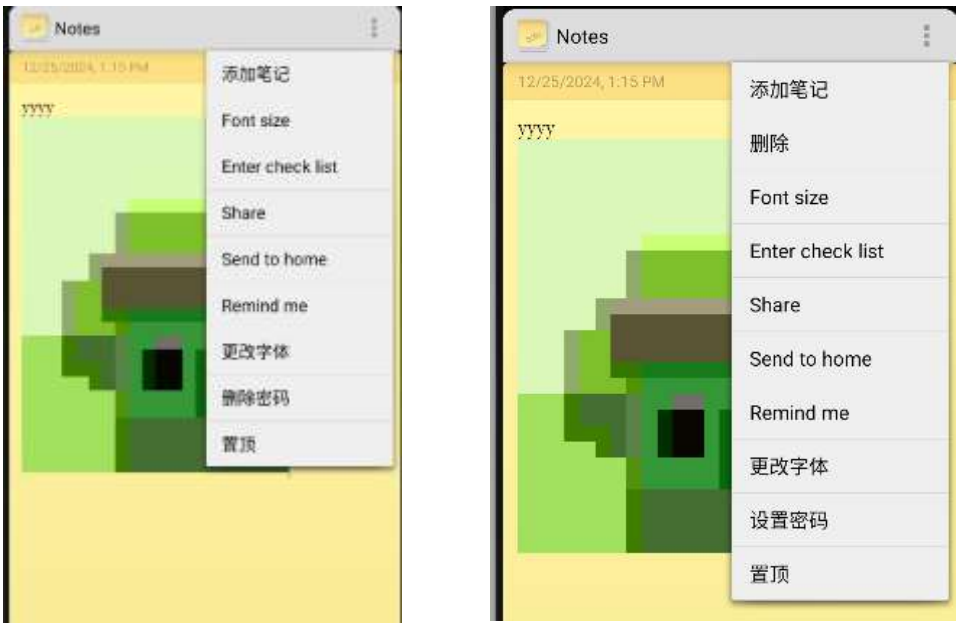


图 156 便签按钮隐藏界面图

3.4.11 插入图片功能

在小米便签应用中，用户现被赋予了插入图片的功能，以增强便签内容的丰富性和信息表达的力度。为启用此项功能，用户必须先行授权小米便签访问其媒体库的权限。权限一经授权，用户即可在便签编辑界面左下角识别并激活“插入图片”按钮。随即，系统将呈现一个界面，展示用户媒体库中的图片供其选择。用户仅需从提供的图片列表中挑选合适的图片，即可便捷地将其嵌入便签之中。界面设计如图 157 所示。

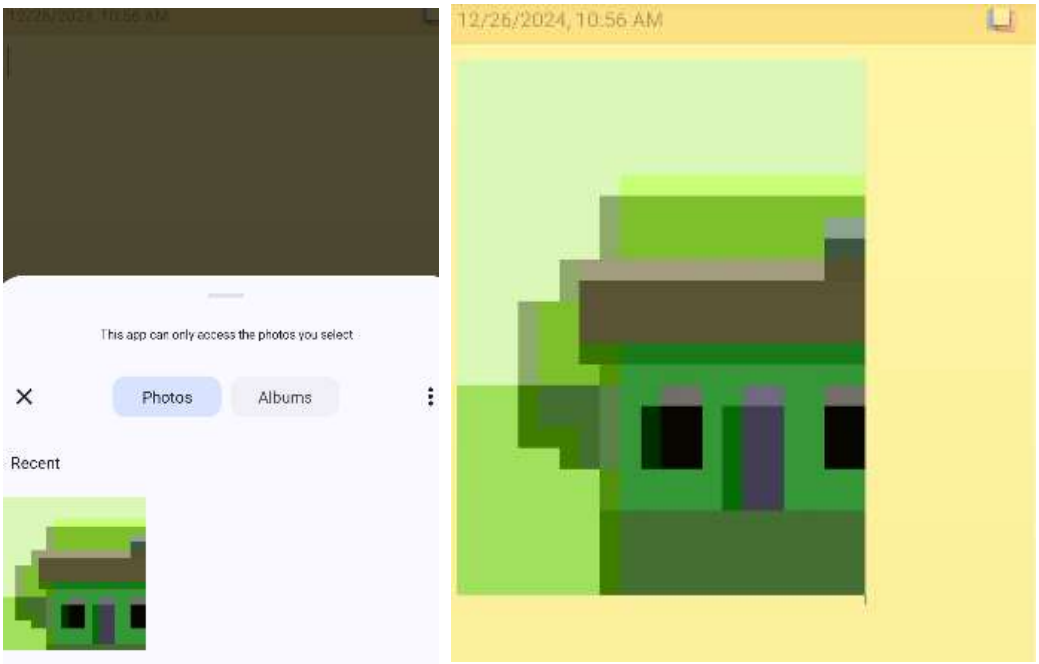


图 157 便签修改字体功能界面图

3.4.12 字符统计功能

用户在编辑便签内容时，系统会自动统计字符数并显示在标题栏的字符数显示区域。这样用户就可以实时查看当前便签的字符数。字符数统计功能不仅可以统计所有字符，还可以过滤掉空格和换行符，确保统计结果的准确性和实用性。界面设计如图 158 所示。

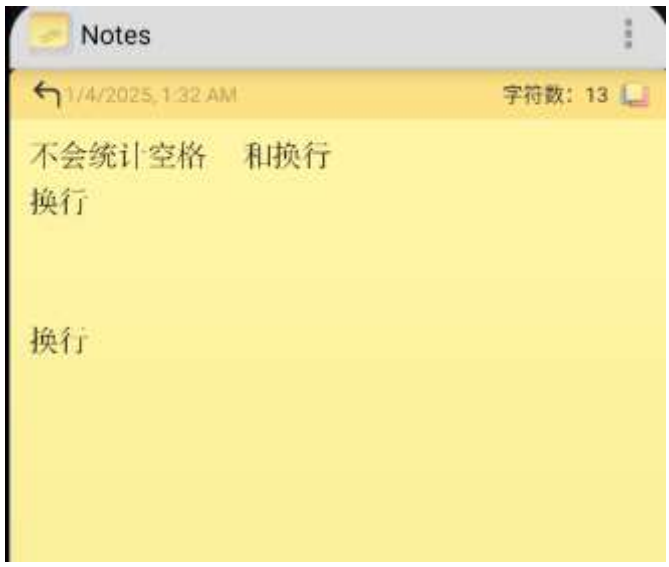


图 158 字符统计功能界面图

3.4.13 文本撤回功能

对于文本撤回功能，用户在编辑便签时，可以撤回最近一次的文本修改。具体的操作界面如图 159 所示。在便签编辑界面（note_edit.xml），我们在标题栏添加了一个用于撤回操作的按钮。用户在编辑便签内容时，可以点击撤回按钮，系统会自动撤回最近一次的文本修改。并且我们提供了详细的系统提示来规范非法操作。文本撤回功能通过保存文本的历史记录，实现了便捷的撤回操作，提升了用户的编辑体验。

图 159 文本撤回功能界面图

3.4.14 朗读文本功能

朗读文本功能的用户界面设计旨在为用户提供直观且便捷的操作体验。在便签编辑界面中，用户可以通过点击菜单中的“朗读”按钮启动朗读功能。启动后，屏幕底部会显示两个功能按钮：一个用于暂停或继续朗读，另一个用于退出朗读模式。暂停按钮的文本会根据当前朗读状态动态切换为“暂停”或“继续”，以便用户清楚地知道当前的操作选项。退出按钮则允许用户直接结束朗读模式，返回到正常的编辑界面。

在朗读过程中，系统会实时显示朗读进度，并通过语音反馈告知用户当前的操作状态。例如，当用户点击暂停按钮时，系统会暂停朗读并提示“朗读已暂停”；当用户点击继续按钮时，系统会从暂停的位置继续朗读并提示“朗读继续”。如果用户选择退出朗读模式，系统会停止朗读并隐藏底部的功能按钮，恢复到默认的编辑界面。

此外，系统还提供了详细的错误提示和引导。例如，如果用户在未选中文本的情况下点击朗读按钮，系统会提示“请先输入文本”；如果 TTS 引擎初始化失败，系统会提示“语音引擎初始化失败，请重试”。通过这些设计，用户可以轻松掌握朗读功能的使用方法，并在需要时获得及时的帮助和反馈。

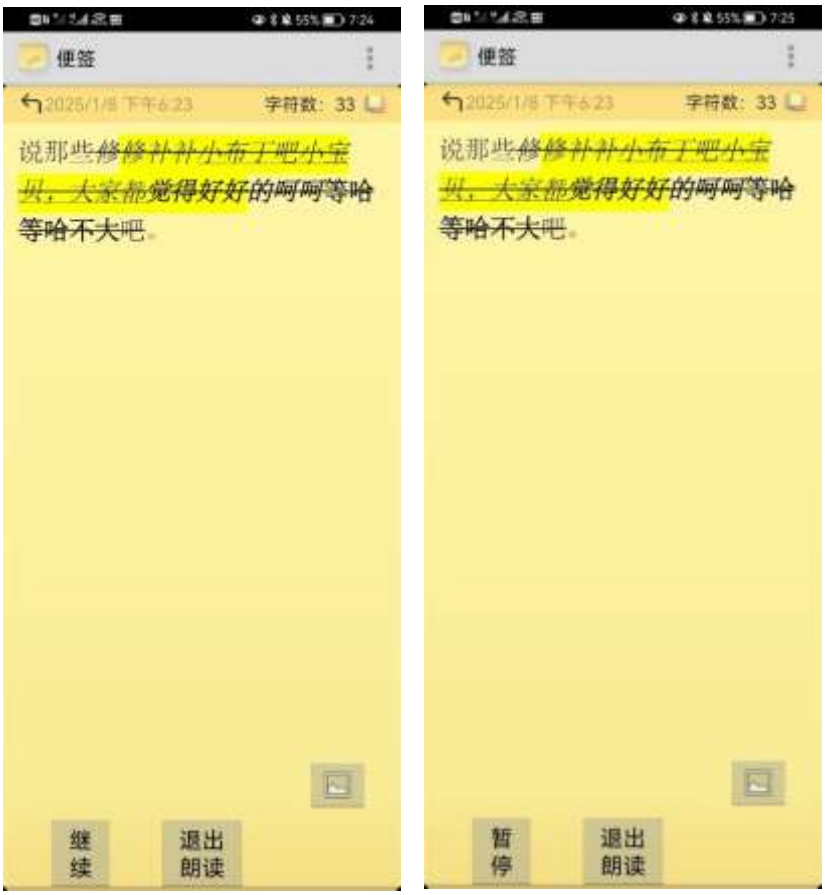


图 160 朗读文本功能界面图

3.4.15 大模型交互功能

在小米便签中集成讯飞星火大模型功能的用户界面设计旨在提供直观且高效的操作体验。当用户点击“大模型对话”按钮时，界面从全屏编辑模式切换为分屏模式，左侧为聊天界面，包含一个 RecyclerView 用于显示聊天记录、一个 EditText 用于输入问题以及一个“发送”按钮，右侧为缩放后的编辑界面，用户可以在聊天界面中与大模型进行实时对话，并将对话中有用的内容复制到右侧的编辑界面中。聊天界面通过 isChatVisible 变量管理显示状态，点击“退出对话”按钮后恢复全屏编辑界面。当用户点击“语音听写”按钮时，界面跳转至讯飞星火提供的语音听写功能主界面，该界面提供了语音识别、语音合成、人像识别、语音评测等多种语音相关功能的入口，用户选择具体功能后进入相应的功能界面，系统会在进入功能界面前检查用户是否同意隐私政策并请求必要的权限（如录音权限、存储权限、摄像头权限等），确保用户在使用语音功能时的隐私和安全。整体界面设计简洁明了，功能入口清晰，操作流畅，旨在提升用户在使用便签应用时的智能化体验。

图 161 大模型交互功能界面图

3.4.16 富文本编辑功能

富文本编辑功能的用户界面设计旨在为用户提供直观且便捷的文本格式化操作体验。当用户在便签中长按选中文字时，屏幕底部会浮现出一个格式工具栏，工具栏中包含加粗、倾斜、删除线、高亮和下划线等多种格式按钮，用户可以通过点击这些按钮对选中的文字应用相应的格式，并实时看到文本样式的更新。工具栏的设计简洁明了，按钮图标清晰易识别，确保用户能够快速找到所需的格式功能。在便签退出时，系统会自动将当前文本的富文本样式信息转换为 JSON 格式并存储到数据库中，下次打开便签时，系统会从数据库中读取这些样式信息并自动恢复文本的格式，确保用户编辑内容的连贯性和一致性。整体界面设计注重实用性和美观性，格式工具栏的显示与隐藏流畅自然，不会干扰用户的编辑操作，同时通过实时渲染和格式存储机制，为用户提供了高效且灵活的富文本编辑体验。

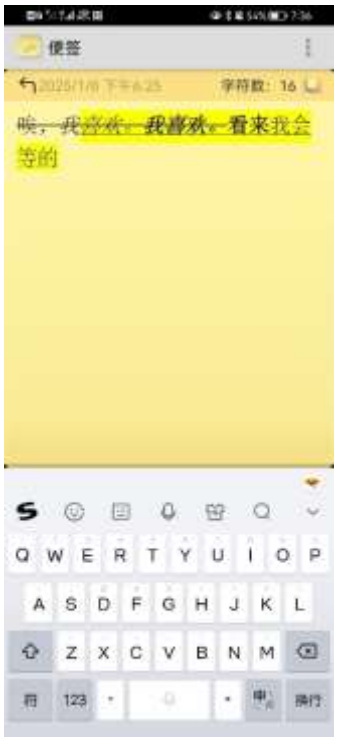


图 162 富文本编辑功能界面图

3.4.17 百度翻译功能

百度翻译功能的用户界面设计旨在为用户提供简洁直观的翻译操作体验。在便签编辑界面中，用户可以通过点击翻译按钮触发翻译功能，系统会弹出一个翻译选项对话框，对话框中列出了多种语言选项（如“中文 → 英文”、“英文 → 中文”、“中文 → 日语”等），用户可以根据需求选择源语言和目标语言。选择完成后，系统会自动调用百度翻译 API，将编辑框中的文本内容翻译成目标语言，并将翻译结果实时显示在编辑框中，同时通过提示信息告知用户翻译成功或失败。翻译过程中，界面会保持流畅的交互体验，确保用户能够快速完成跨语言的文本转换。整体设计注重功能的高效性和易用性，翻译选项对话框布局清晰，语言选择一目了然，翻译结果的显示与编辑框无缝衔接，为用户提供了便捷的多语言处理能力，同时通过签名生成机制和错误提示功能，确保了翻译过程的安全性和可靠性。

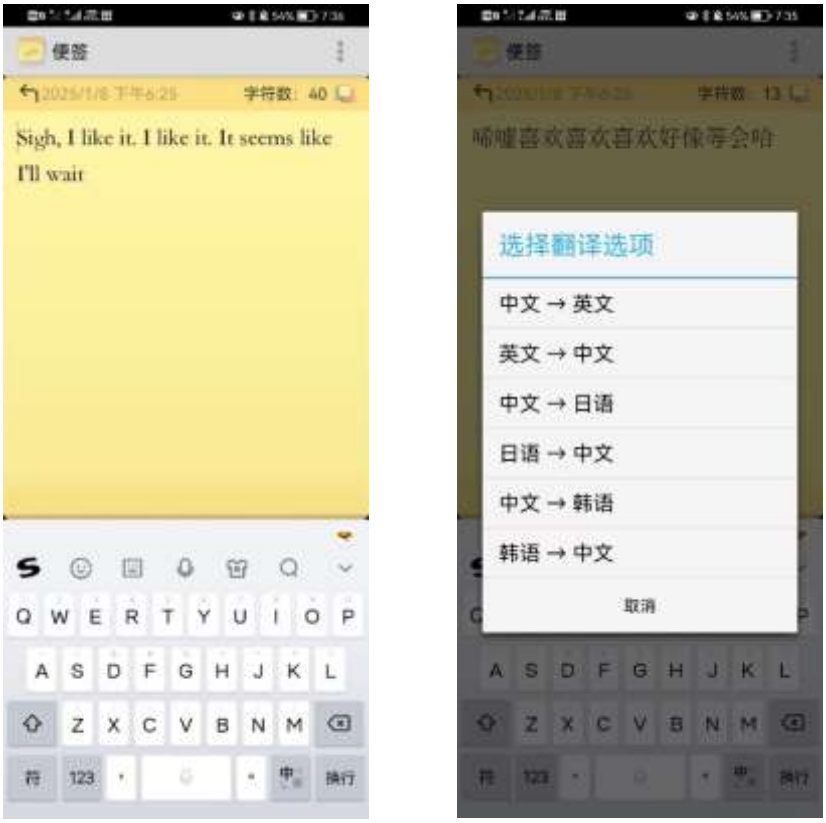


图 163 百度翻译功能界面图

3.4.18 好友管理功能

管理好友功能的用户界面设计致力于为用户提供一个清晰直观的社交管理体验。当用户点击应用主界面的"添加好友"按钮时，系统会弹出一个简洁的对话框，供用户输入想要添加的好友账号。对话框设计简单明了，包含一个文本输入框和两个操作按钮（"添加"和"取消"），方便用户快速完成好友添加操作。

在好友列表的显示方面，系统采用垂直线性布局，每个好友项目以卡片形式呈现，清晰展示好友的账号信息。用户可以通过长按好友项触发删除操作，系统会弹出确认对话框以防止误操作。好友列表支持实时更新，当添加或删除好友后，列表会立即刷新以反映最新状态。

为了增强用户体验，系统在每次操作后都会提供即时反馈。例如，添加好友成功时会显示"好友添加成功"的提示，账号不存在时会提示"账户不存在"。整体界面设计注重简洁性和易用性，确保用户能够轻松管理其社交关系。通过数据库的实时同步机制，用户的好友列表在不同设备间保持一致，提供了稳定可靠的好友管理体验。具体界面设计如动图 164 所示。

图 164 管理好友功能界面图

3.4.19 转发便签功能

转发便签功能的用户界面设计旨在为用户提供一个流畅且友好的信息分享体验。当用户在便签列表界面选中一条便签并点击转发按钮时，系统会展开一个好友选择对话框，其中清晰列出了当前用户的所有好友。对话框采用列表的形式，让用户可以选择好友作为转发对象。

对话框顶部设有醒目的标题"选择好友"，底部配备了"取消"操作按钮，布局层次分明。用户选择完成后点击要转发的好友，系统会立即开始转发操作，并通过 Toast 提示信息实时反馈转发进度。为了确保用户体验的流畅性，转发过程采用异步处理机制，避免界面卡顿。转发成功后，系统会显示"便签已转发给 xxx"

的提示信息，让用户明确知道操作结果。具体界面设计如动图 165 所示。

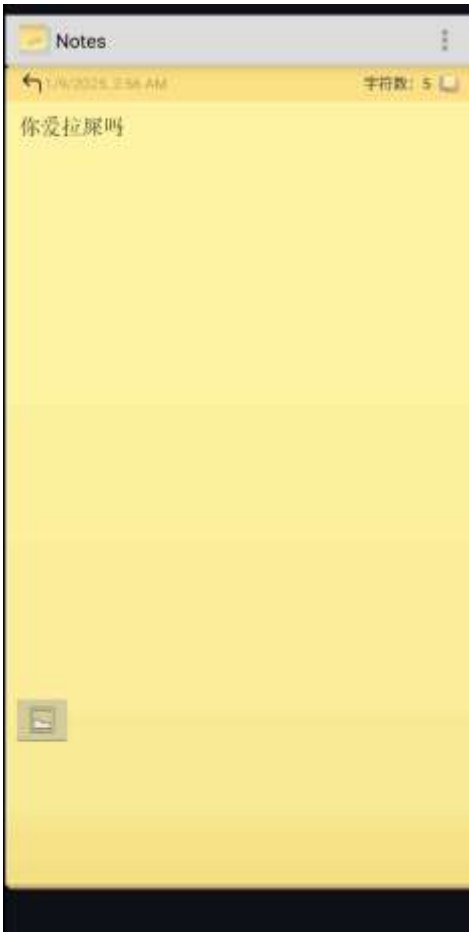


图 165 转发便签界面图

4. 实践收获和体会

收获 1：深入理解软件工程思想和原则

通过阅读、分析和维护开源软件，我们深刻领会了软件工程的理念、思想和方法。这个过程让我们切身理解了软件工程的思想和原则，深入掌握了高质量开发方法及技术，积累了软件开发技能和工程素质。

收获 2：提升代码阅读和维护能力

实践过程中，我们学会了如何阅读和理解大量高质量的代码，理解软件设计及编码风格，高水平编程技巧和方法。同时，我们也学会了如何运用所学的方法和技能来维护开源系统，解决实践中遇到的问题，并使用分布式协同手段进行软件开发。

收获 3：合作的重要性

通过参与开源项目，我们深刻体会到合作的力量。开源项目的成功很大程度

上依赖于团队成员的共同努力，包括代码审查、测试和反馈等环节。团队成员的共同努力可以显著降低软件的维护成本，成员可以帮助编写代码、测试和文档，分担维护的重任。

收获 4：深入理解开源文化和协作模式

通过参与开源项目，我们深入了解了开源文化的核心价值，包括开放性、协作性、共享性和透明度。我们学会了如何在一个多元化的社区中有效沟通和协作，这对于提升我们的团队合作能力和跨文化沟通技巧非常有帮助。

收获 5：提升项目管理和领导能力

在维护开源软件的过程中，我们有机会参与项目管理和领导工作，如规划项目路线图、协调团队成员、解决冲突等。这些经验让我们在项目管理和领导方面获得了宝贵的实践机会，增强了我们的组织和协调能力。

收获 6：掌握先进的开发工具和实践

在开源项目中，我们接触到了许多先进的开发工具和实践，如 Git、GitHub、Jenkins、Docker 等。这些工具和实践不仅提高了我们的技术能力，也让我们能够更高效地进行软件开发和维护。

收获 7：如何保证开源项目的代码质量

保证开源项目中的代码质量依赖于几个核心的策略：代码审查、持续集成（CI）、测试覆盖率、文档完整性、以及社区活跃度。这些策略共同形成了一个高效的生态系统，确保了代码的高质量和项目的持续发展。

问题 1：代码编写规范性问题

我们在实际编写代码过程中，发现自身编写的代码中存在的较大的质量问题。这包括代码编写规范行、编程风格等问题。这些问题的发现和解决过程，让我们更加重视代码编写质量的重要性。

问题 2：开源项目的维护成本

开源软件的维护成本取决于软件的复杂性、社区活跃度、自身维护策略及持续的更新与支持需求。对于庞大、复杂且广泛使用的开源软件，维护成本可能包括专业的团队、基础设施费用以及合规性、安全性相关的开销。

问题 3：代码冲突的处理

在开源项目中，当多个开发者同时修改同一代码文件时，可能会出现代码冲

突。解决这些冲突需要使用版本控制系统查看冲突的文件和代码行，理解每个开发者的意图，修改冲突的代码，进行必要的测试，并提交合并后的代码。

问题 4：开源项目的文档和知识传递问题

在实践中，我发现文档的完整性和易理解性对于开源项目的成功至关重要。如何有效地创建、维护和更新项目文档，以及如何确保知识在社区中的传递，是开源项目需要解决的问题。

5. 大模型辅助实践的认识

5.1 使用的大模型 CASS 工具

在本次软件开发实践活动中，本小组借助了多种大模型工具来辅助开展工作，具体包括 Kimi、GitHub copilot、智谱清言、deepseek 等。这些工具各具特色，为我们提供了丰富的功能支持，助力软件开发实践的顺利推进。

5.2 大模型辅助的实践工作

本小组主要利用大模型工具在以下几个方面的软件开发实践工作中发挥了重要作用：

1. 代码理解和调试：在面对复杂的代码逻辑时，大模型工具能够快速帮助我们理解代码的功能和结构。例如，当遇到一段晦涩难懂的算法实现代码时，通过向大模型工具提问，它能够以通俗易懂的语言解释代码的运行流程和关键点，让我们迅速把握代码的核心。同时，在调试代码过程中，当出现 bug 时，大模型工具可以基于代码片段分析可能的错误原因，并给出相应的修复建议。比如，当数据库操作部分出现了问题，大模型会引导我们检查相关代码，最终成功修复了 bug。具体与大模型交互的过程如图 166 所示。

图 166 分析错误交互过程图

2. 文档生成和优化：软件开发中，编写高质量的文档是必不可少的环节。大模型工具能够根据代码自动生成函数结构的相关信息，包括函数的功能描述、参数说明、返回值等关键信息。我们以一个自定义的数据处理函数为例，大模型工具能清晰地列出了函数的输入输出以及大致的处理逻辑，我们在此基础上进一步参考和完善，大大提高了文档编写的效率和质量。具体交互过程如图 167 所示。

图 167 获取函数信息交互过程图

5.3 大模型辅助实践的收获和问题

使用大模型工具辅助开展软件开发实践工作，效果显著，但也存在一些问题：

- **收获：**
  - **开发效率大幅提升：**大模型工具在代码理解、调试、文档生成以及技术问题解答等方面为我们节省了大量的时间和精力。以往需要花费数小时甚至数天去排查的 bug，在大模型工具的辅助下，往往能在较短时间内得到解决；文档的编写速度也因工具的初步生成而加快，整体开发进度明显加快。
  - **知识获取更加便捷：**借助大模型工具，我们能够轻松获取到丰富的技

术知识和解决方案。无论是代码撰写方面的问题，还是软件的搭建部署问题，都能通过简单的提问得到回应，拓宽了我们的知识面，提升了团队的技术水平。

- **问题：**
  - **生成内容的准确性有待提高：**虽然大模型工具在大多数情况下能够给出有用的建议和解答，但有时生成的内容存在不准确的情况。比如在某些特定场景下，给出的代码修复方案可能会引入新的问题，或者提供的技术解释不够精确，需要我们结合实际情况进行甄别和调整。
  - **对上下文理解不够深入：**大模型工具在处理复杂的软件开发场景时，有时对代码的上下文环境理解不够深入。例如，在一个大型项目中，代码模块之间的关联较为复杂，大模型工具可能无法准确把握某个代码片段在整个项目架构中的作用和依赖关系，导致给出的建议不够贴合实际项目需求。
  - **依赖性风险：**在使用大模型工具的过程中，团队成员可能会逐渐形成对工具的过度依赖。一旦遇到工具无法解决的问题，或者工具出现故障无法使用时，成员可能会感到束手无策，影响开发进度。因此，我们需要在使用工具的同时，不断提升自身的专业技能和独立解决问题的能力。