G08小组

“城院跳蚤”微信小程序

详细设计报告



摘要

本详细设计说明书旨在为大学校园二手交易微信小程序的开发提供详细的设计指导，明确软件系统的设计细节，包括程序结构、功能、性能、接口等方面，确保开发团队成员对系统设计有清晰一致的理解，便于后续的开发、测试、维护等工作，主要读者包括项目开发团队成员

目录

[1引言 5](#_Toc182767476)

[1.1编写目的 5](#_Toc182767477)

[1.2背景 5](#_Toc182767478)

[1.3定义 5](#_Toc182767479)

[1.4参考资料 6](#_Toc182767480)

[2程序系统的结构 6](#_Toc182767481)

[2.1功能模块划分 6](#_Toc182767482)

[2.2模块间层级关系 6](#_Toc182767483)

[2.2.1用户模块 6](#_Toc182767484)

[2.2.2商品模块 6](#_Toc182767485)

[2.2.3交易模块 7](#_Toc182767486)

[2.2.4消息模块 7](#_Toc182767487)

[2.2.5评价模块 7](#_Toc182767488)

[2.2.6管理员模块 7](#_Toc182767489)

[2.3数据流动逻辑 7](#_Toc182767490)

[2.3.1 用户相关数据流动 7](#_Toc182767491)

[2.3.2 商品相关数据流动 8](#_Toc182767492)

[2.3.3 消息相关数据流动 8](#_Toc182767493)

[3用户个人信息管理说明 9](#_Toc182767494)

[1. 概述 9](#_Toc182767495)

[2. 功能 9](#_Toc182767496)

[3. 性能 9](#_Toc182767497)

[4. 输入项 9](#_Toc182767498)

[5. 输出项 10](#_Toc182767499)

[6. 算法 10](#_Toc182767500)

[7. 流程逻辑 10](#_Toc182767501)

[8. 接口 11](#_Toc182767502)

[9. 存储分配 11](#_Toc182767503)

[10. 注释设计 11](#_Toc182767504)

[4 管理员信息管理说明 11](#_Toc182767505)

[1. 概述 11](#_Toc182767506)

[2. 功能 12](#_Toc182767507)

[3. 性能 12](#_Toc182767508)

[4. 输入项 12](#_Toc182767509)

[5. 输出项 12](#_Toc182767510)

[6. 算法 12](#_Toc182767511)

[7. 流程逻辑 13](#_Toc182767512)

[5 程序设计说明示例：用户注册模块 13](#_Toc182767513)

[1 程序描述 13](#_Toc182767514)

[2 功能 13](#_Toc182767515)

[3 性能 13](#_Toc182767516)

[4 输入项 13](#_Toc182767517)

[5 输出项 13](#_Toc182767518)

[6 算法 14](#_Toc182767519)

[7 流程逻辑 14](#_Toc182767520)

[8 接口 14](#_Toc182767521)

[9 存储分配 14](#_Toc182767522)

[10 注释设计 14](#_Toc182767523)

[6 帖子搜索分类设计说明 14](#_Toc182767524)

[1概述 14](#_Toc182767525)

[2关键词搜索 14](#_Toc182767526)

[实现方式： 14](#_Toc182767527)

[技术细节： 15](#_Toc182767528)

[伪代码示例： 15](#_Toc182767529)

[3 分类标签筛选 15](#_Toc182767530)

[实现方式： 15](#_Toc182767531)

[技术细节： 15](#_Toc182767532)

[伪代码示例： 15](#_Toc182767533)

[4时间筛选 16](#_Toc182767534)

[实现方式 16](#_Toc182767535)

[技术细节： 16](#_Toc182767536)

[伪代码示例： 16](#_Toc182767537)

[5 综合搜索 17](#_Toc182767538)

[实现方式 17](#_Toc182767539)

[技术细节： 17](#_Toc182767540)

[伪代码示例： 17](#_Toc182767541)

[6总结 18](#_Toc182767542)

[**7** 帖子编辑删除设计说明 18](#_Toc182767543)

[1 概述 18](#_Toc182767544)

[2 编辑权限 18](#_Toc182767545)

[权限控制 18](#_Toc182767546)

[实现方式 18](#_Toc182767547)

[编辑内容 18](#_Toc182767548)

[3 编辑逻辑 18](#_Toc182767549)

[前端界面 18](#_Toc182767550)

[后端处理： 18](#_Toc182767551)

[4 删除逻辑 19](#_Toc182767552)

[权限控制： 19](#_Toc182767553)

[实现方式： 19](#_Toc182767554)

[删除操作： 19](#_Toc182767555)

[软删除（可选） 19](#_Toc182767556)

[5 后台实现 19](#_Toc182767557)

[数据库设计： 19](#_Toc182767558)

[API设计： 19](#_Toc182767559)

[日志记录： 19](#_Toc182767560)

[异常处理： 20](#_Toc182767561)

[6 总结 20](#_Toc182767562)

# 

# 

# 1引言

## 1.1编写目的

本详细设计说明书旨在为大学校园二手交易微信小程序的开发提供详细的设计指导，明确软件系统的设计细节，包括程序结构、功能、性能、接口等方面，确保开发团队成员对系统设计有清晰一致的理解，便于后续的开发、测试、维护等工作，主要读者包括项目开发团队成员，如软件工程师、测试人员、产品经理等，以便他们能够依据此文档进行后续的详细设计、编码实现、测试以及维护工作。

## 1.2背景

说明：

1.名字“城院跳蚤”--大学二手交易微信小程序；

2.本项目由林钰航、刘智豪、高琪阳、朱化宇共同提出并开发，此小程序通过微信打开并直接使用。

## 1.3定义

二手物品：曾经被使用过，目前处于闲置状态且仍具有一定使用价值，可供在平台上进行交易转让的物品。

用户：在小程序中注册并参与二手物品交易活动的个人，包括出售物品的卖家和购买物品的买家。

订单：买家与卖家之间达成交易意向后生成的记录，包含交易物品信息、价格、买卖双方信息、交易状态等详细内容。

微信小程序：小程序的一种，是一种不需要下载安装即可使用的应用，它实现了应用“触手可及”的梦想，用户扫一扫或搜一下即可打开应用。

## 1.4参考资料

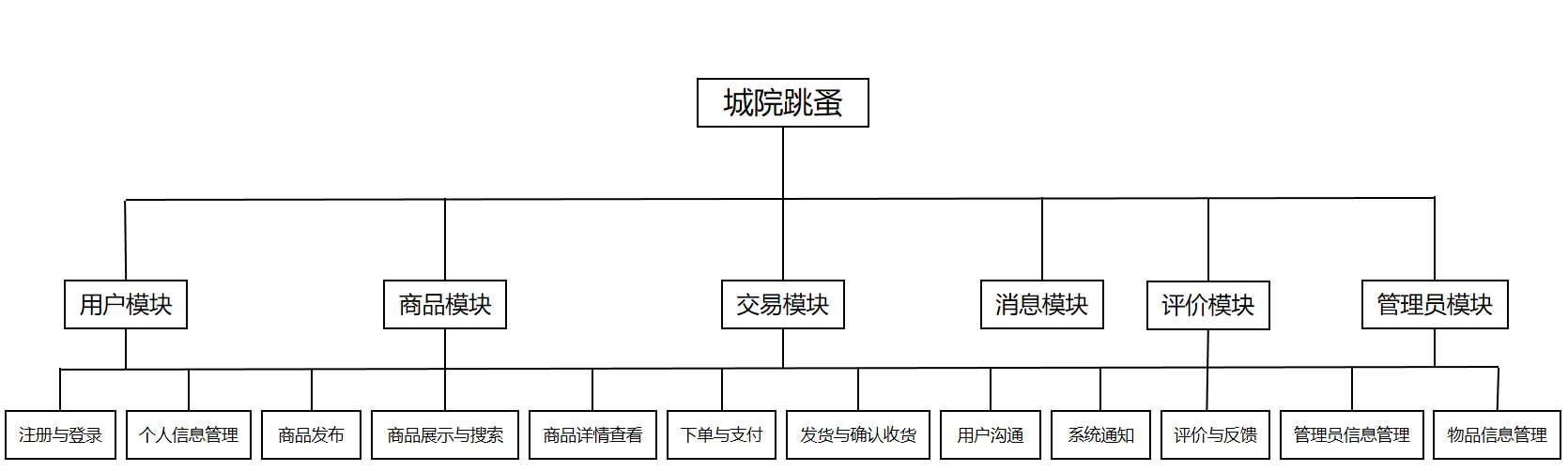
GB8567 - 88《计算机软件产品开发文件编制指南》

# 2程序系统的结构

用一系列图表列出本程序系统内的每个程序（包括每个模块和子程序）的名称、标识符和它们之间的层次结构关

本大学校园二手交易微信小程序系统主要由以下功能模块组成，各模块间层级关系明确，数据流动逻辑清晰，共同协作实现系统的整体功能。

## 2.1功能模块划分



## 2.2模块间层级关系

### 2.2.1用户模块

整个系统的基础，为其他模块提供用户身份信息，其他模块在操作过程中会涉及到用户信息的调用与更新，例如交易模块需要获取买家和卖家的用户信息来记录交易，评价模块需要根据用户的交易记录来进行评价操作

### 2.2.2商品模块

与用户模块紧密关联，用户可以发布商品（通过商品发布功能），同时商品展示与搜索功能为用户提供查找所需商品的途径，商品详情查看功能也依赖于商品模块中的数据存储。商品模块与交易模块、评价模块相互作用，商品的交易和评价都基于商品模块中的商品信息，交易完成后会对商品状态产生影响（如已售出等），评价则会为商品提供更多的参考信息。

### 2.2.3交易模块

依赖于用户模块和商品模块，在交易过程中需要获取用户信息进行身份验证和交易记录关联，同时涉及商品信息的处理（如商品库存更新等）。交易模块与消息模块有信息交互，交易状态的变化会触发系统通知给用户，买家和卖家也可通过消息模块进行交易相关的沟通。

### 2.2.4消息模块

与用户模块、交易模块相互配合，用户模块确定消息的发送和接收对象，交易模块中的交易事件（如下单、发货等）触发消息模块的通知功能，同时用户之间的沟通也在消息模块中进行，消息记录与用户信息相关联。

### 2.2.5评价模块

用户模块、商品模块、交易模块都存在关联，用户在交易完成后进行评价（依赖交易模块的交易记录），评价内容与商品模块中的商品相关联，同时评价信息也可供其他用户查看（涉及用户模块的用户身份识别），并且会影响商品的推荐（个性化推荐模块）。

### 2.2.6管理员模块

处于系统的管理层面，对商品模块、用户模块等进行管理和维护，例如管理员通过物品信息管理来管理商品相关数据，通过管理员信息管理来管理自身信息，管理员模块的操作会影响整个系统的数据准确性和正常运行。

## 2.3数据流动逻辑

### 2.3.1 用户相关数据流动

用户注册时，用户输入的注册信息（如用户名、密码、联系方式等）从用户界面流向用户模块中的注册功能，经过验证后存储到用户信息库。用户登录时，输入的账号密码信息流向用户模块进行身份验证，验证通过后用户可在系统中进行各项操作，其操作过程中产生的个人信息修改需求（如修改头像、联系方式等）在个人信息管理模块中处理，更新后的信息回写至用户信息库；在交易过程中，用户模块将买家和卖家的用户信息传递给交易模块，交易模块根据交易情况（如交易成功、发货、确认收货等）更新用户的交易记录，这些记录可能会被评价模块调用，用户对交易的评价信息也会与用户信息关联存储，同时评价信息可能影响个性化推荐模块对该用户的商品推荐，个性化推荐模块根据用户信息库中的个性化需求特征和历史选择记录库中的数据进行推荐计算，推荐结果展示给用户，用户的浏览和选择行为又会被记录到历史选择记录库中，形成数据闭环。

### 2.3.2 商品相关数据流动

卖家发布商品时，商品信息（名称、描述、照片、价格、发布时间等）从用户界面流向商品模块的发布功能，然后存储到商品表中。商品展示与搜索功能根据用户输入的搜索条件或个性化需求，从商品表、物品 tag 信息库（Tag 筛选物品模块）或结合用户信息库（个性化推荐模块）中获取数据，筛选出符合条件的商品信息展示给用户。用户查看商品详情时，商品模块从商品表中读取相应商品的详细信息（包括图片、名称、价格、描述、卖家联系方式、交易评价等）并展示在详情页面；交易模块在处理下单、发货、确认收货等流程时，会读取和更新商品表中的商品状态信息（如库存数量、销售状态等）。评价模块在用户进行交易评价后，将评价内容和评分存储到评价表中，同时评价信息会反馈给商品模块，影响商品的综合评价展示，也可能被个性化推荐模块用于调整推荐策略。管理员在物品信息管理和物品 tag 管理模块中的操作，直接对商品表和物品 tag 信息库中的数据进行添加、删除或修改，从而影响商品的展示和搜索结果。

### 2.3.3 消息相关数据流动

交易模块中的交易事件（如买家下单、卖家发货等）触发消息模块生成系统通知，通知内容包含交易相关信息（如订单编号、交易状态等），消息模块根据用户模块中的用户联系方式将通知发送给相应用户。用户之间的聊天信息从聊天界面流向消息模块，消息模块将聊天记录存储到消息表中，用户可随时查看聊天历史记录，聊天记录与用户信息关联，确保消息的准确传递和分类展示。

# 3用户个人信息管理说明

## 1. 概述

用户个人信息管理模块负责用户注册、登录、信息更新等核心功能。其设计目标是确保用户数据的安全性和可靠性，同时提供便捷的操作体验。

## 2. 功能

用户信息的录入：用户通过注册界面录入姓名、邮箱、密码等基本信息。

用户信息的更新：用户可通过个人中心修改个人信息，如昵称、头像、联系方式等。

用户信息的安全管理：对敏感信息进行加密存储，确保隐私数据的安全性。

## 3. 性能

**精度**：对输入数据（如邮箱、手机号）进行格式校验，确保数据的准确性。

**灵活性**：支持用户随时更新信息并实时生效。

**时间特性**：数据更新和验证的响应时间小于1秒。

## 4. 输入项

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 格式 | 有效范围 | 来源 | 频率 |
| 姓名 | Name | String | 任意字符 | 不超过30个字符 | 用户输入 | 每次注册 |
| 邮箱/手机号 | Email/Phone | String | 邮箱或11位数字 | 合法邮箱或手机号格式 | 用户输入 | 每次注册或更新 |
| 密码 | Password | String | 字母+数字 | 至少8位，包含1个字母和1个数字 | 用户输入 | 每次注册 |

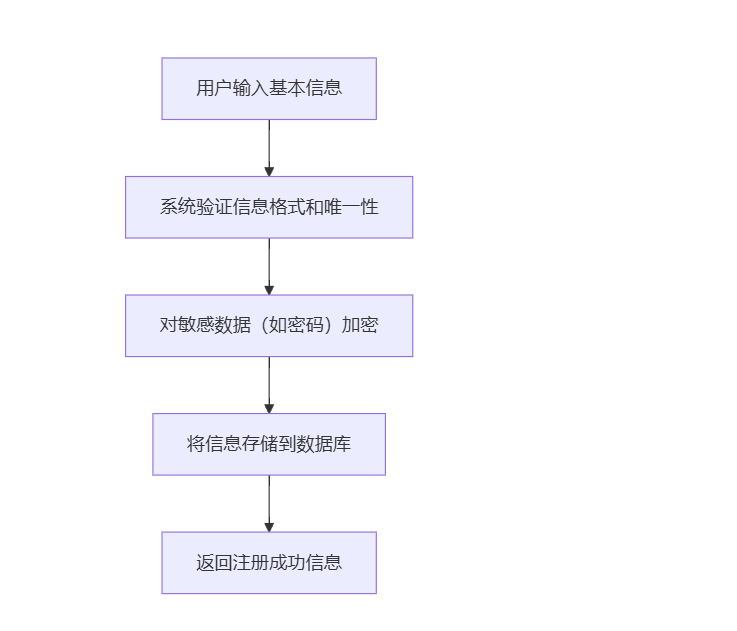
## 5. 输出项

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 格式 | 有效范围 | 频率 |
| 用户唯一标识 | UserID | String | UUID格式 | 系统自动生成 | 每次注册 |
| 更新状态 | UpdateStatus | Boolean | 成功/失败 | 更新成功或失败原因 | 每次更新 |

## 6. 算法

## 7. 流程逻辑

用户注册与更新的主要流程如下：



## 8. 接口

**数据库接口**：用户信息表，包含字段（UserID, Name, Email/Phone, Password, Avatar, UpdatedAt）。

**API接口**：

POST /register 用于注册。

PUT /update-info 用于信息更新。

## 9. 存储分配

每个用户分配一个唯一标识符（UUID），与其所有数据关联存储在关系型数据库中。

## 10. 注释设计

在模块首部加注释，说明模块功能和使用方法。

在密码加密逻辑和身份验证处添加详细注释，说明逻辑和条件。

# 4 管理员信息管理说明

## 1. 概述

管理员信息管理模块负责平台的权限控制，管理员可通过该模块管理用户信息、审核内容以及执行数据维护操作。

## 2. 功能

权限控制：确保管理员仅访问授权模块。

用户审核：筛选违规用户或内容。

数据管理：增删改查平台数据。

## 3. 性能

**精度**：管理员操作需准确定位目标数据。

**灵活性**：支持多种数据管理操作，如批量处理。

**时间特性**：审核操作的延迟小于1秒。

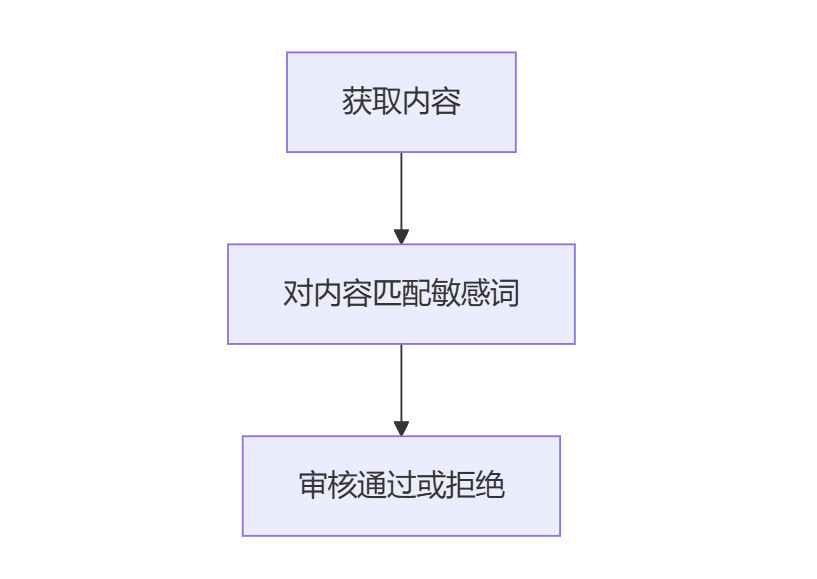
## 4. 输入项

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 来源 |
| 管理员账号 | AdminID | String | 系统分配 |
| 审核内容 | AuditData | JSON | 用户提交内容 |
| 操作命令 | Command | String | 管理员输入 |

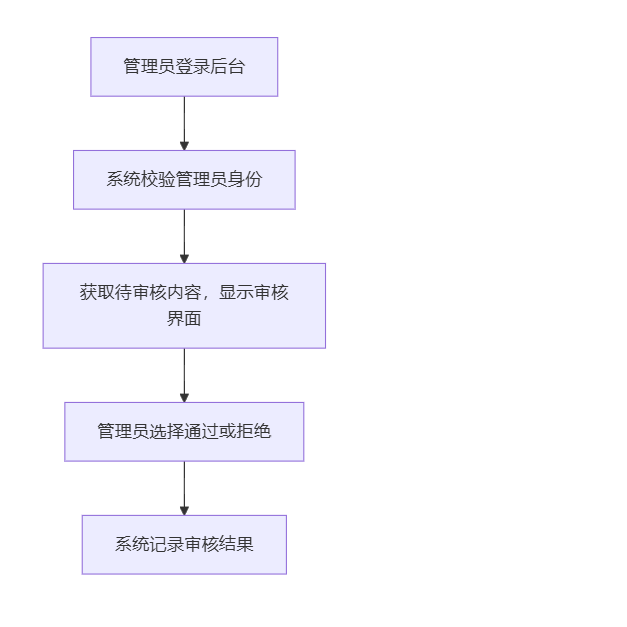
## 5. 输出项

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 说明 |
| 审核状态 | AuditStatus | Boolean | 是否通过审核 |
| 操作反馈 | Feedback | JSON | 包含处理结果 |

## 6. 算法



## 7. 流程逻辑



# 5 程序设计说明示例：用户注册模块

## 1 程序描述

用户注册模块为临时调用的子程序，旨在完成用户账号创建，支持并发处理并返回唯一标识符。

## 2 功能

输入：用户姓名、邮箱、密码等。

处理：验证输入，保存用户信息。

输出：注册成功或失败的消息。

## 3 性能

响应时间：<1秒。

可扩展性：支持1000+并发。

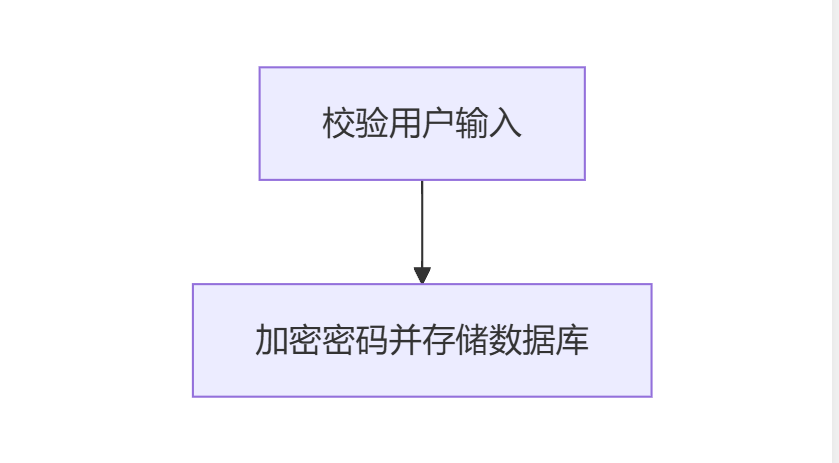
## 4 输入项

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 格式 |
| 姓名 | String | UTF-8 |

### 5 输出项

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 格式 |
| 注册反馈信息 | String | JSON |

## 6 算法



## 7 流程逻辑

详见流程图。

## 8 接口

数据库接口：用户表（UserID, Name, Password）。

API接口：POST /register。

## 9 存储分配

为每个用户分配唯一ID存储。

## 10 注释设计

注释在字段验证和加密逻辑处标明逻辑细节。

# 6 帖子搜索分类设计说明

## 1概述

在二手交易小程序中，帖子搜索分类功能对于用户查找所需商品至关重要。通过关键词、分类标签、时间等多种筛选条件，用户可以快速定位到感兴趣的帖子，提高交易效率。本部分将详细描述这些筛选条件的实现方式。

## 2关键词搜索

实现方式：采用全文搜索引擎（如Elasticsearch）对帖子标题、内容、标签等字段进行索引。当用户输入关键词时，搜索引擎在索引中查找匹配的帖子，并按照相关性排序返回结果。

### 技术细节：

**分词处理**：对关键词和帖子内容进行分词，以便准确匹配。

**权重分配**：为不同字段（如标题、内容）设置不同的权重，以影响搜索结果的相关性排序。

**同义词处理**：支持同义词扩展，提高搜索的灵活性。

### 伪代码示例：

**function searchByKeyword(keyword):**

**posts = fetchAllPosts() // 获取所有帖子**

**result = []**

**for post in posts:**

**if keyword in post.title or keyword in post.content:**

**result.append(post)**

**return result**

## 3 分类标签筛选

实现方式：建立分类标签体系，每个帖子在发布时选择相应的标签。用户在搜索时，可以选择一个或多个标签进行筛选。

### 技术细节：

**标签体系设计**：根据商品类型、品牌、状态等维度设计标签体系，确保标签的覆盖度和准确性。

**标签关联**：帖子发布时，通过后台或前端界面让用户选择标签，并建立帖子与标签的关联关系。

**筛选逻辑**：根据用户选择的标签，过滤出包含这些标签的帖子。

### 伪代码示例：

**function searchByCategory(category):**

**posts = fetchAllPosts() // 获取所有帖子**

**result = []**

**for post in posts:**

**if post.category == category:**

**result.append(post)**

**return result**

## 4时间筛选

实现方式：提供时间筛选条件，如“最近一周”、“最近一个月”等，用户可以选择特定时间段进行筛选。

### 技术细节：

**时间字段**：在帖子数据表中添加时间字段（如发布时间、更新时间），用于记录帖子的时间信息。

**时间范围计算**：根据用户选择的时间段，计算对应的时间范围，并在查询时加入时间筛选条件。

**排序逻辑**：在时间筛选的基础上，可以进一步按时间顺序（如从新到旧或从旧到新）对结果进行排序。

### 伪代码示例：

**function searchByTime(timeRange):**

**posts = fetchAllPosts() // 获取所有帖子**

**currentTime = getCurrentTime() // 获取当前时间**

**result = []**

**for post in posts:**

**PostTime = post.postTime // 获取帖子发布时间**

**if postTime is within timeRange(currentTime, timeRange):**

**result.append(post)**

**return result**

**function timeRange(currentTime, range):**

**// 根据时间范围判断帖子是否在规定时间内**

## 5 综合搜索

实现方式：支持关键词、分类标签、时间等多种筛选条件的组合使用，用户可以根据需要选择多个条件进行搜索。

### 技术细节：

查询构建：根据用户选择的筛选条件，动态构建SQL查询或Elasticsearch查询语句。

**结果合并**：对于多个筛选条件的查询结果，进行合并和去重处理，确保结果的准确性和完整性。

### 伪代码示例：

**function searchPosts(keyword, category, timeRange):**

**posts = fetchAllPosts() // 获取所有帖子**

**result = []**

**for post in posts:**

**if (keyword in post.title or keyword in post.content) and**

**post.category == category and**

**post.postTime is within timeRange(getCurrentTime(), timeRange):**

**result.append(post)**

**return result**

## 6总结

帖子搜索分类功能通过关键词搜索、分类标签筛选和时间筛选等多种方式，为用户提供灵活高效的搜索体验。在实现过程中，需要关注索引性能、查询效率、标签体系的合理性和用户界面的友好性等方面。

# 7 帖子编辑删除设计说明

## 1 概述

在二手交易小程序中，用户对已发布帖子的编辑和删除功能是其管理个人发布内容的重要手段。本部分将详细描述用户对已发布帖子的编辑权限、删除逻辑及后台实现。

## 2 编辑权限

权限控制：只有帖子的原始发布者才能对其帖子进行编辑。

实现方式：

**身份验证**：帖子只能由发帖本人进行编辑。

**权限检查**：在后台处理编辑请求时，检查用户是否具有编辑权限。

编辑内容：允许用户修改帖子标题、内容、价格、图片等字段。

## 3 编辑逻辑

前端界面：提供编辑界面，让用户选择要修改的字段并输入新内容。

### 后端处理：

**接收请求**：接收前端发送的编辑请求，包括帖子ID和修改后的内容。

**数据校验**：对修改后的内容进行数据校验，确保符合平台规定（如标题长度、图片格式等）。

**数据库更新**：将修改后的内容保存到数据库中，更新对应帖子的记录。

**版本控制**（可选）：记录帖子的编辑历史，以便在必要时进行版本回滚。

## 4 删除逻辑

权限控制：只有帖子的原始发布者才能删除其帖子。

### 实现方式：

**身份验证**：用户在删除帖子时，需要进行身份验证。

**权限检查**：在后台处理删除请求时，检查用户是否具有删除权限。

### 删除操作：

**前端界面**：提供删除按钮或链接，让用户确认是否删除帖子。

**后端处理**：

**接收请求**：接收前端发送的删除请求，包括帖子ID。

**数据库删除**：从数据库中删除对应帖子的记录。

**关联处理**：如帖子有评论、收藏等关联数据，需一并处理（如删除评论、取消收藏等）。

软删除（可选）：为了保留数据痕迹和防止误删，可以采用软删除方式，即不直接删除帖子记录，而是将其标记为已删除状态。

## 5 后台实现

数据库设计：在帖子数据表中添加必要的字段（如发布者ID、发布时间、编辑时间、删除状态等）以支持编辑和删除功能。

API设计：为前端提供编辑和删除的API接口，包括请求参数、返回结果和错误处理机制。

日志记录：记录编辑和删除操作的日志信息，以便进行审计和故障排查。

异常处理：对可能出现的异常情况（如数据库连接失败、数据校验失败等）进行捕获和处理，确保系统的稳定性和可靠性。

## 6 总结

帖子编辑删除功能通过严格的权限控制、合理的编辑逻辑和删除逻辑以及完善的后台实现，为用户提供了安全可靠的帖子管理手段。在实现过程中，需要关注权限控制的准确性、数据校验的严格性、数据库操作的效率和异常处理的完善性等方面。

# 8评论功能设计说明

## 1. 概述

评论功能模块是社交平台中用户互动的核心部分，允许用户对帖子发表评论、回复其他用户的评论以及删除自己的评论。设计目标是提供一个简单、直观的界面，同时确保评论内容的实时更新和数据一致性。

## 2. 功能

发布评论：用户可以对特定帖子发表评论，评论内容可以包括文本、图片或视频。

回复评论：用户可以对其他用户的评论进行回复，形成评论链。

删除评论：用户可以删除自己发布的评论，管理员可以删除违规评论。

## 3. 性能

精度：确保评论内容的准确性和完整性。

灵活性：支持用户随时发布、回复和删除评论。

时间特性：评论的发布、回复和删除操作的响应时间小于1秒。

## 4. 输入项

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 格式 | 有效范围 | 来源 | 频率 |
| 评论内容 | CommentContent | String | 文本/图片/视频 | 文本不超过500个字符 | 用户输入 | 每次发布或回复 |
| 帖子标识 | PostID | String | UUID格式 | 系统自动生成 | 系统提供 | 每次发布 |
| 父评论标识 | ParentCommentID | String | UUID格式 | 系统自动生成 | 系统提供 | 每次回复 |
| 用户标识 | UserID | String | UUID格式 | 系统自动生成 | 系统提供 | 每次操作 |

## 5. 输出项

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 格式 | 有效范围 | 频率 |
| 评论唯一标识 | CommentID | String | UUID格式 | 系统自动生成 | 每次发布 |
| 操作状态 | OperationStatus | Boolean | 成功/失败 | 操作成功或失败原因 | 每次操作 |

## 6. 算法

评论发布算法：验证用户输入的评论内容，将其与帖子标识和用户标识关联，存储到数据库，并生成唯一的评论标识。

回复评论算法：验证用户输入的回复内容，将其与父评论标识和用户标识关联，存储到数据库，并生成唯一的评论标识。

删除评论算法：验证用户是否有权限删除指定的评论，如果验证通过，则从数据库中删除该评论。

## 7. 流程逻辑

发布评论流程：

1. 用户选择帖子并输入评论内容。

2. 系统验证评论内容的格式和长度。

3. 系统将评论内容与帖子标识和用户标识关联，存储到数据库。

4. 系统生成唯一的评论标识并返回给用户。

回复评论流程：

1. 用户选择要回复的评论并输入回复内容。

2. 系统验证回复内容的格式和长度。

3. 系统将回复内容与父评论标识和用户标识关联，存储到数据库。

4. 系统生成唯一的评论标识并返回给用户。

删除评论流程：

1. 用户选择要删除的评论。

2. 系统验证用户是否有权限删除该评论。

3. 如果验证通过，系统从数据库中删除该评论。

4. 系统返回操作状态给用户。

## 8. 接口

数据库接口：评论信息表，包含字段（CommentID, PostID, UserID, ParentCommentID, CommentContent, CreatedAt）。

API接口：

- POST /comment 用于发布评论。

- POST /reply 用于回复评论。

- DELETE /comment 用于删除评论。

## 9. 存储分配

每个评论分配一个唯一标识符（UUID），与其所有数据关联存储在关系型数据库中。

## 10. 注释设计

在模块首部加注释，说明模块功能和使用方法。

在评论发布、回复和删除逻辑处添加详细注释，说明逻辑和条件。

## 伪代码：

// 评论发布

function postComment(UserID, PostID, CommentContent):

if validateCommentContent(CommentContent):

CommentID = generateUniqueID()

timestamp = getCurrentTimestamp()

storeComment(CommentID, UserID, PostID, CommentContent, timestamp)

return {"CommentID": CommentID, "OperationStatus": true}

else:

return {"OperationStatus": false, "Reason": "Invalid comment content"}

// 评论回复

function replyComment(UserID, ParentCommentID, CommentContent):

if validateCommentContent(CommentContent):

CommentID = generateUniqueID()

timestamp = getCurrentTimestamp()

storeReply(CommentID, UserID, ParentCommentID, CommentContent, timestamp)

return {"CommentID": CommentID, "OperationStatus": true}

else:

return {"OperationStatus": false, "Reason": "Invalid comment content"}

// 评论删除

function deleteComment(UserID, CommentID):

if userHasPermissionToDelete(UserID, CommentID):

deleteCommentFromDatabase(CommentID)

return {"OperationStatus": true}

else:

return {"OperationStatus": false, "Reason": "No permission to delete"}

// 辅助函数

function validateCommentContent(CommentContent):

// 验证评论内容的格式和长度

return length(CommentContent) <= 500 and isValidFormat(CommentContent)

function generateUniqueID():

// 生成唯一的评论标识符

return UUID()

function getCurrentTimestamp():

// 获取当前时间戳

return currentTimestamp()

function storeComment(CommentID, UserID, PostID, CommentContent, timestamp):

// 将评论信息存储到数据库

database.insert("Comments", {CommentID, UserID, PostID, null, CommentContent, timestamp})

function storeReply(CommentID, UserID, ParentCommentID, CommentContent, timestamp):

// 将回复信息存储到数据库

database.insert("Comments", {CommentID, UserID, null, ParentCommentID, CommentContent, timestamp})

function deleteCommentFromDatabase(CommentID):

// 从数据库中删除评论

database.delete("Comments", {CommentID: CommentID})

function userHasPermissionToDelete(UserID, CommentID):

// 验证用户是否有权限删除评论

return database.checkPermission(UserID, CommentID)

# 9历史浏览设计说明

## 1. 概述

历史浏览模块负责记录和管理用户的浏览活动，包括保存用户的浏览记录、提供查询功能以及允许用户清理历史记录。设计目标是确保用户能够方便地查看和管理自己的浏览历史，同时保护用户隐私。

## 2. 功能

保存浏览记录：自动记录用户浏览的页面或内容。

查询浏览记录：允许用户根据时间、内容类型等条件查询浏览历史。

清理浏览记录：用户可以手动清理部分或全部浏览历史记录。

## 3. 性能

精度：确保记录的准确性和完整性。

灵活性：支持用户根据多种条件查询和清理历史记录。

时间特性：浏览记录的保存、查询和清理操作的响应时间小于1秒。

## 4. 输入项

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 格式 | 有效范围 | 来源 | 频率 |
| 浏览内容标识 | ContentID | String | UUID格式 | 系统自动生成 | 系统提供 | 每次浏览 |
| 用户标识 | UserID | String | UUID格式 | 系统自动生成 | 系统提供 | 每次操作 |
| 浏览时间 | ViewTime | DateTime | 标准日期时间格式 | 系统自动生成 | 系统提供 | 每次浏览 |
| 查询条件 | QueryCriteria | JSON | 自定义格式 | 用户定义 | 用户输入 | 每次查询 |

5. 输出项

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 标识符 | 数据类型 | 格式 | 有效范围 | 频率 |
| 浏览记录 | ViewHistory | JSON | 自定义格式 | 包含浏览内容标识、浏览时间等 | 每次查询 |
| 操作状态 | OperationStatus | Boolean | 成功/失败 | 操作成功或失败原因 | 每次操作 |

## 6. 算法

保存浏览记录算法：当用户浏览内容时，系统自动记录内容标识、用户标识和浏览时间，并存储到数据库。

查询浏览记录算法：根据用户输入的查询条件，系统检索数据库中的浏览记录，并返回匹配的结果。

清理浏览记录算法：验证用户是否有权限清理指定的浏览记录，如果验证通过，则从数据库中删除该记录。

## 7. 流程逻辑

保存浏览记录流程：

1. 用户浏览内容。

2. 系统自动生成浏览时间戳。

3. 系统将浏览内容标识、用户标识和浏览时间存储到数据库。

查询浏览记录流程：

1. 用户输入查询条件。

2. 系统根据查询条件检索数据库。

3. 系统返回匹配的浏览记录。

清理浏览记录流程：

1. 用户选择要清理的浏览记录。

2. 系统验证用户是否有权限清理该记录。

3. 如果验证通过，系统从数据库中删除该记录。

4. 系统返回操作状态给用户。

## 8. 接口

数据库接口：历史浏览记录表，包含字段（UserID, ContentID, ViewTime）。

API接口：

- POST /view-history 用于保存浏览记录。

- GET /view-history 用于查询浏览记录。

- DELETE /view-history 用于清理浏览记录。

## 9. 存储分配

每个用户的浏览记录与其用户标识关联存储在关系型数据库中。

## 10. 注释设计

在模块首部加注释，说明模块功能和使用方法。

在保存、查询和清理浏览记录逻辑处添加详细注释，说明逻辑和条件。

伪代码：

// 保存浏览记录

function saveViewHistory(UserID, ContentID):

ViewTime = getCurrentTimestamp()

storeViewHistory(UserID, ContentID, ViewTime)

return {"OperationStatus": true}

// 查询浏览记录

function queryViewHistory(UserID, QueryCriteria):

ViewHistory = retrieveViewHistory(UserID, QueryCriteria)

return {"ViewHistory": ViewHistory, "OperationStatus": true}

// 清理浏览记录

function clearViewHistory(UserID, ContentID):

if userHasPermissionToClear(UserID, ContentID):

deleteViewHistory(UserID, ContentID)

return {"OperationStatus": true}

else:

return {"OperationStatus": false, "Reason": "No permission to clear"}

// 辅助函数

function getCurrentTimestamp():

// 获取当前时间戳

return currentTimestamp()

function storeViewHistory(UserID, ContentID, ViewTime):

// 将浏览记录存储到数据库

database.insert("ViewHistory", {UserID, ContentID, ViewTime})

function retrieveViewHistory(UserID, QueryCriteria):

// 根据查询条件检索数据库中的浏览记录

return database.select("ViewHistory", {UserID: UserID}, QueryCriteria)

function deleteViewHistory(UserID, ContentID):

// 从数据库中删除浏览记录

database.delete("ViewHistory", {UserID: UserID, ContentID: ContentID})

function userHasPermissionToClear(UserID, ContentID):

// 验证用户是否有权限清理浏览记录

return database.checkPermission(UserID, ContentID)