“物品复活”软件概要设计说明书

版本管理

版本	日期	变更内容	编制人
V1.0	2024-12-26	初稿编写完成	代敬宇
V2.0	2025-01-05	完善了一些图表	代敬宇

1 引言

说明：

- 本软件名称为“物品复活”软件，是上海交通大学CS3331软件工程的课程项目。
- 本软件可供希望将闲置物品赠送或转卖给有需要的人（即物品复活）的用户，以及该系统的管理员使用。

2 总体设计

2.1 系统功能描述

该软件允许：

1. 添加物品的信息
2. 删除物品的信息
3. 显示物品列表
4. 查找物品的信息
5. 为了便于管理和查询，物品分成了不同的类别，不同类别的物品有不同的属性
6. 有两种类型的用户：管理员和普通用户。普通用户使用该软件必须得到管理员用户的批准，批准后只能添加物品、删除自己所发布的物品、显示物品列表、查找物品的信息；管理员用户在普通用户的基础功能上，还可以设置新的物品类型（定义物品类型的名称和各个属性）、修改物品类型、批准普通用户的注册。

2.2 运行环境

本软件开发环境是：

1. 操作系统：Windows 10
2. Python版本：3.11.9
3. Python第三方库：PyQt5

2.3 设计思想

系统构思

作为“物品复活”系统，“物品”的管理肯定是重中之重，所以实现一个`ItemManager`类，管理一个`Item`类列表，并拥有一系列核心数据操作。这些操作与负责GUI的`MainWindow`类密切联系，负责与用户的交互。管理员通过物品类型管理功能灵活地定义和修改物品的种类及其属性，而普通用户则可以添加自己的物品，管理自己的物品列表（如删除物品），并执行物品查询操作。系统通过表单验证、数据管理和实时更新UI等机制，确保了物品管理功能的完整性和便捷性。

由于物品是有用户发布的，所以也有必要实现一个`UserManager`类来管理一个`User`类列表，负责用户的登录、批准等操作。

关键数据结构

1. `Item`（物品）

`Item`类是系统中最核心的数据结构之一，表示一个物品。每个`Item`对象包含以下属性：

- `name`：物品的名称。
- `description`：物品的描述。
- `address`：物品所在地址。
- `phone`：物品相关的电话。
- `email`：物品相关的邮箱。
- `item_type`：物品类型（如食品、书籍等）。
- `item_attributes`：一个字典，存储特定物品类型的额外属性，例如食品的保质期或书籍的作者等。字典结构使得物品的属性管理更加灵活，并能够根据物品类型动态生成属性输入框，使管理员能增加或者修改物品类型。

2. `ItemManager`（物品管理器）

`ItemManager`类是用于管理系统中所有物品的核心数据结构。它内部可能维护一个物品列表（如列表或字典），用于存储所有物品，并提供对物品的增、删、查等操作。

- `items`：一个列表或字典，存储所有的Item对象。
- `get_all_items()`：返回所有物品的列表。

- `add_item(item)`：向管理器中添加物品。
- `delete_item(item_name)`：根据物品名称删除物品。
- `search_item(item_type, query_text)`：根据物品类型和查询文本搜索物品。
- `get_item_type_attributes(item_type)`：根据物品类型返回该类型的属性列表。

3. `User`（用户）

用户类（如`OrdinaryUser`和`AdministratorUser`）表示系统中的用户，包含用户的基本信息（如用户名、地址、电话、邮箱等）及权限状态（是否为管理员、是否批准等）。

-`username`：用户名。

-`is_admin`：是否为管理员。

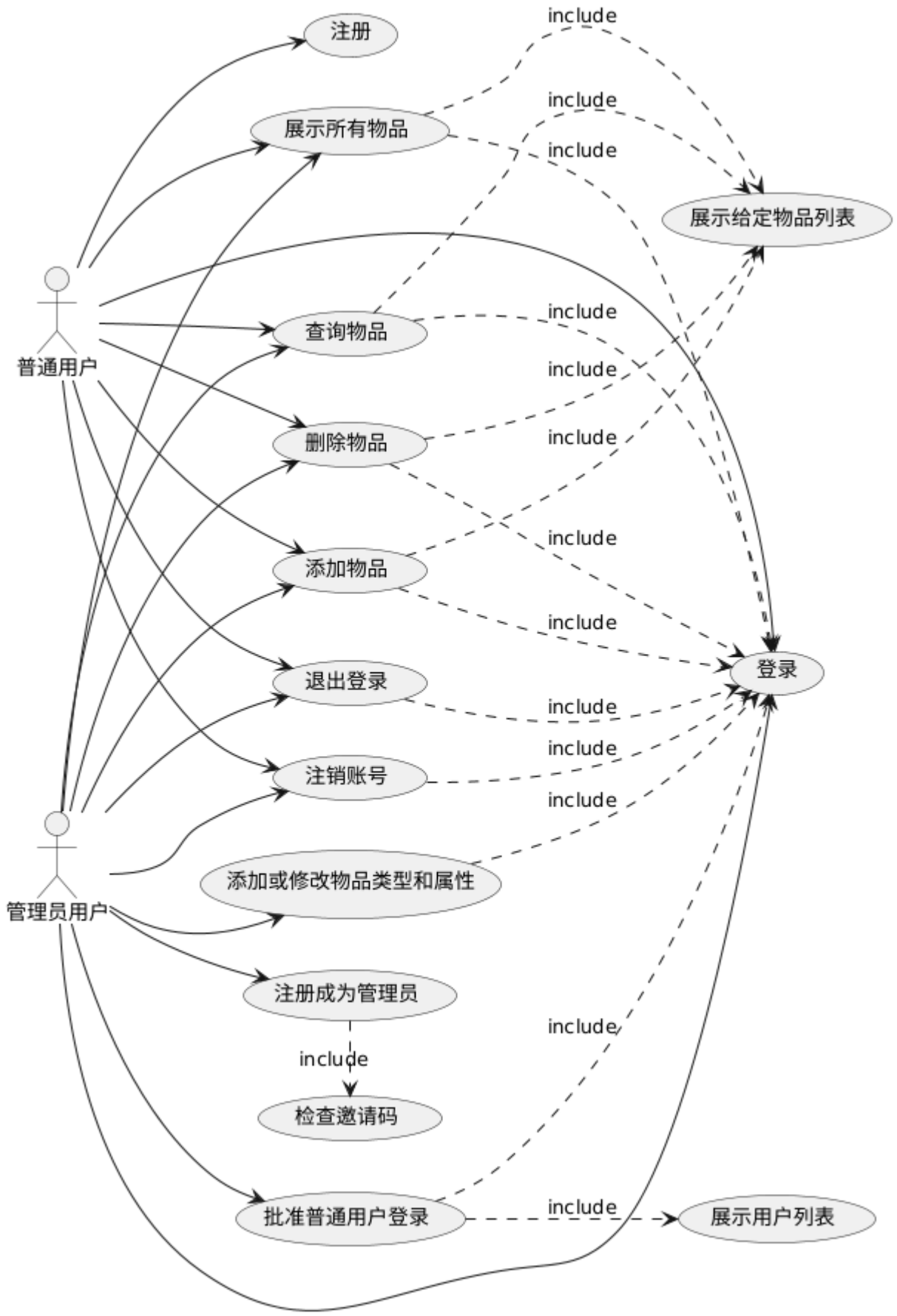
-`is_approved`：是否为已批准用户。

4. `UserManager`（用户管理器）

`UserManager`类用于管理系统中的所有用户信息，类似于`ItemManager`管理物品的方式。

- `users`：一个列表或字典，存储所有User对象。
- `get_all_users()`：返回所有用户的列表。
- `approve_user(username)`：批准用户成为正式用户。

2.4 用例图



1. 活动者分析

该用例图包含以下两个活动者：

- **普通用户**：普通用户是系统的基本用户，拥有基本的物品管理权限。
- **管理员用户**：管理员用户拥有更高的权限，除了普通用户的权限外，还包括管理用户和物品类型的权限。

2. 用例与活动者的关系

- **普通用户**：
  - 参与的用例：
    - **登录**：普通用户可以登录系统。
    - **注册**：普通用户可以注册成为新用户。
    - **添加物品**：普通用户可以添加物品。
    - **删除物品**：普通用户可以删除自己的物品。
    - **查询物品**：普通用户可以查询物品。
    - **退出登录**：普通用户可以退出系统。
    - **展示所有物品**：普通用户可以查看所有物品。
    - **注销账号**：普通用户可以注销自己的账号。
- **管理员用户**：
  - 参与的用例：
    - **登录**：管理员用户也可以登录系统。
    - **注册成为管理员**：管理员用户可以通过注册成为管理员来获得更高权限。
    - **添加物品**：管理员用户可以添加物品。
    - **删除物品**：管理员用户可以删除物品。
    - **查询物品**：管理员用户可以查询物品。
    - **展示所有物品**：管理员用户可以查看所有物品。
    - **添加或修改物品类型和属性**：管理员用户可以添加或修改物品的类型及其属性。
    - **批准普通用户登录**：管理员用户可以批准普通用户的登录请求。
    - **退出登录**：管理员用户可以退出系统。
    - **注销账号**：管理员用户可以注销自己的账号。

3. 用例之间的关系

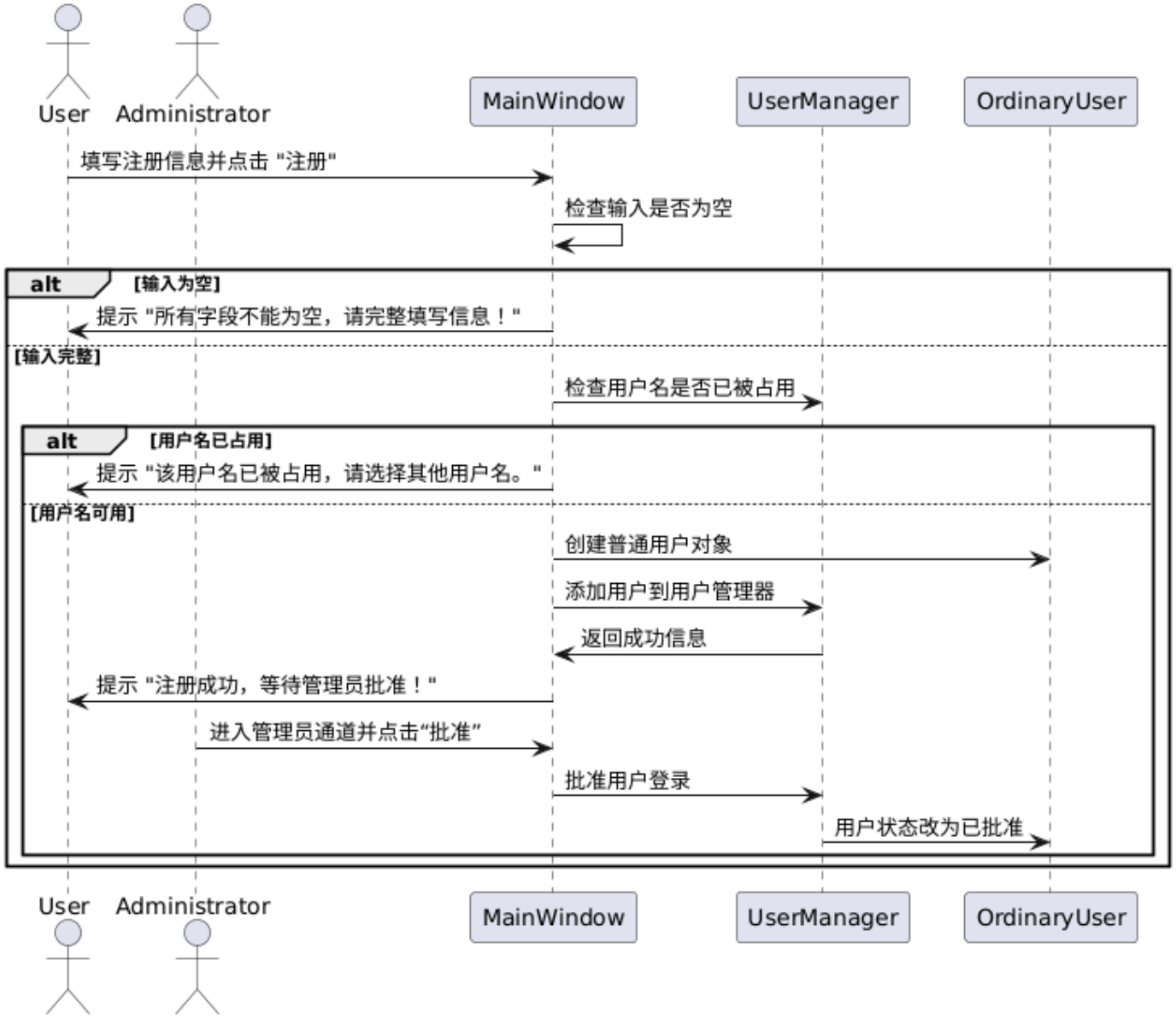
- **包含（Include）关系**：

    - **登录**是许多用例的基本前提，多个用例（如添加物品、删除物品、查询物品、展示所有物品、添加或修改物品类型和属性、批准普通用户登录、退出登录、注销账号）都需要登录才能执行。因此，这些用例与登录之间有包含关系。
    - **注册成为管理员**包含了**检查邀请码**的用例，表明只有在检查邀请码之后，用户才能注册为管理员。
    - **添加物品**、**删除物品**、**查询物品**、**展示所有物品**等操作，需要依赖**展示给定物品列表**，以便在添加、删除或查询物品时更新物品的显示。
    - **批准普通用户登录**与**展示用户列表**相关，说明管理员在批准用户登录之前，需要展示用户列表。
- **扩展（Extend）关系**：
    - 在此用例图中，未使用扩展关系。

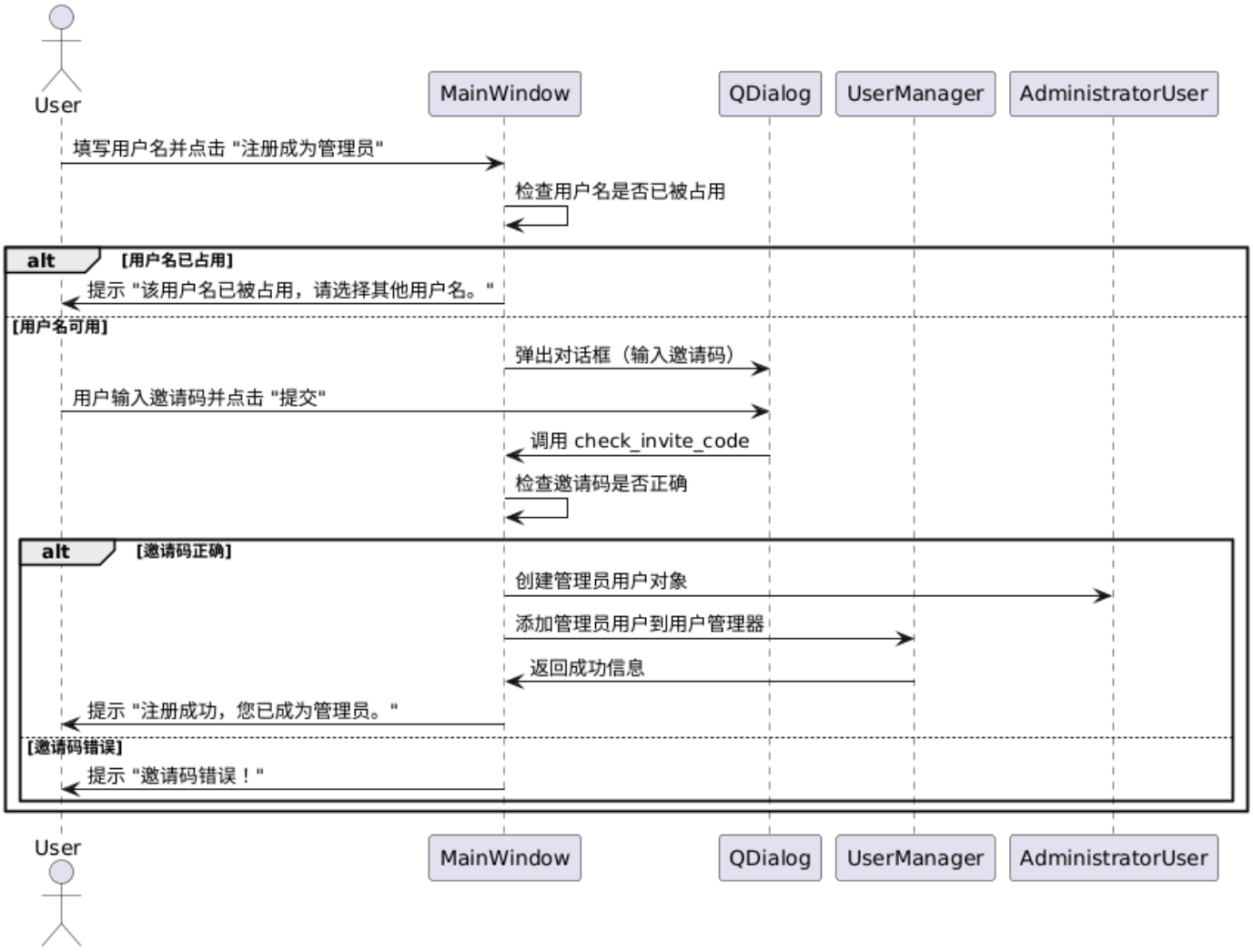
2.5 各用例的顺序图

下面几节展示了用例图中与活动者直接相关的11个用例的顺序图。

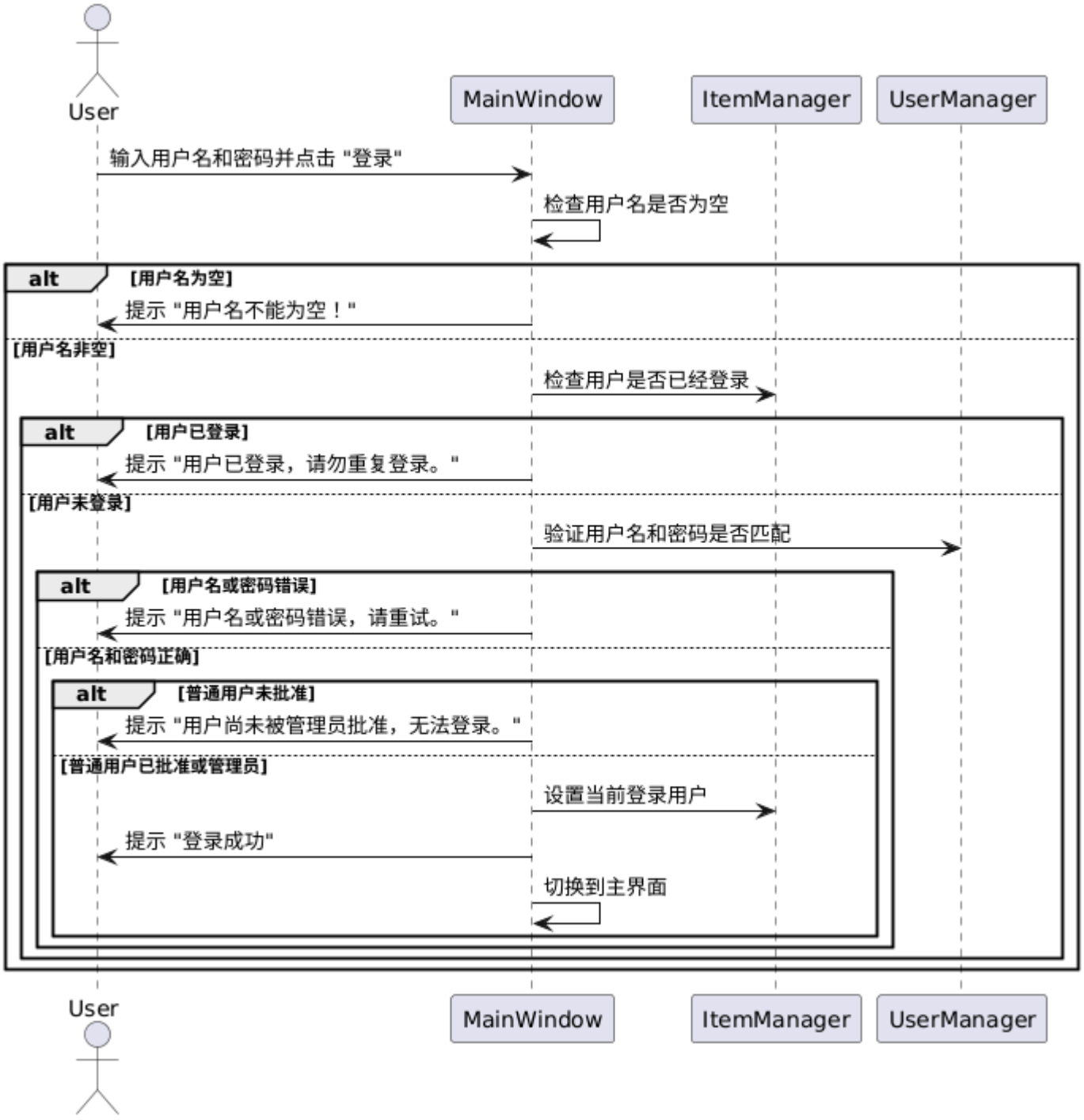
2.5.1 注册


User
Administrator
MainWindow
UserManager
OrdinaryUser
填写注册信息并点击 "注册"
检查输入是否为空
alt
[输入为空]
提示 "所有字段不能为空，请完整填写信息！"
[输入完整]
检查用户名是否已被占用
alt
[用户名已占用]
提示 "该用户名已被占用，请选择其他用户名。"
[用户名可用]
创建普通用户对象
添加用户到用户管理器
返回成功信息
提示 "注册成功，等待管理员批准！"
进入管理员通道并点击“批准”
批准用户登录
用户状态改为已批准
User
Administrator
MainWindow
UserManager
OrdinaryUser

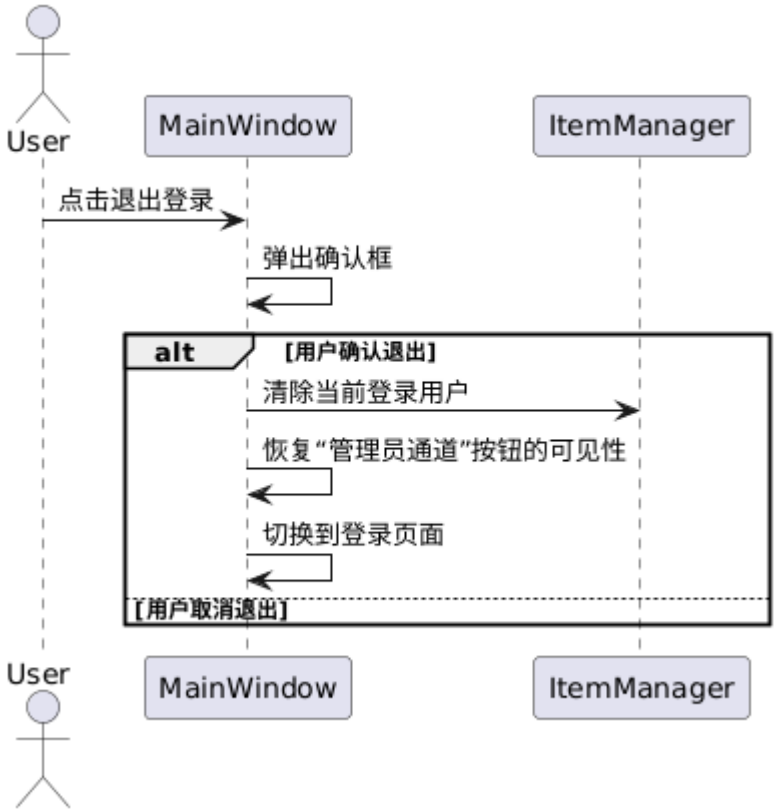
2.5.2 注册成为管理员


User
MainWindow
QDialog
UserManager
AdministratorUser
填写用户名并点击 "注册成为管理员"
检查用户名是否已被占用
alt
[用户名已占用]
提示 "该用户名已被占用，请选择其他用户名。"
[用户名可用]
弹出对话框（输入邀请码）
用户输入邀请码并点击 "提交"
调用 check_invite_code
检查邀请码是否正确
alt
[邀请码正确]
创建管理员用户对象
添加管理员用户到用户管理器
返回成功信息
提示 "注册成功，您已成为管理员。"
[邀请码错误]
提示 "邀请码错误！"
User
MainWindow
QDialog
UserManager
AdministratorUser

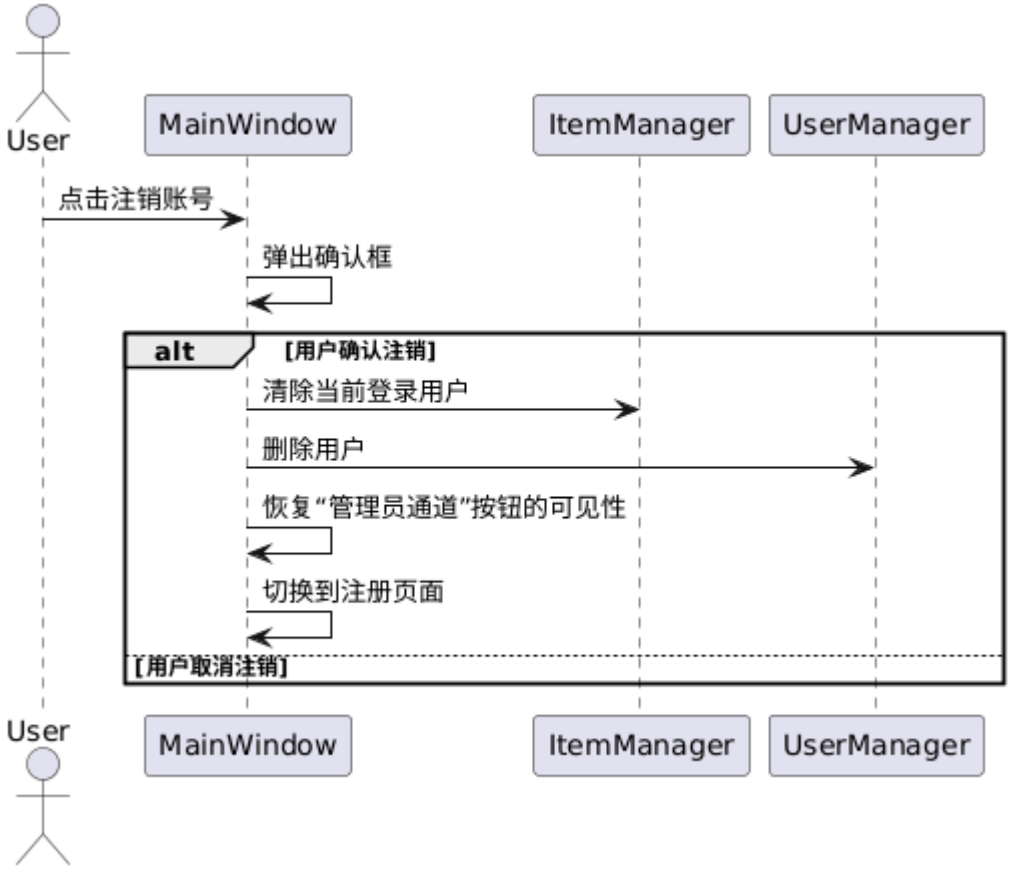
2.5.3 登录


User
MainWindow
ItemManager
UserManager
输入用户名和密码并点击 "登录"
检查用户名是否为空
alt
[用户名为空]
提示 "用户名不能为空！"
[用户名非空]
检查用户是否已经登录
alt
[用户已登录]
提示 "用户已登录，请勿重复登录。"
[用户未登录]
验证用户名和密码是否匹配
alt
[用户名或密码错误]
提示 "用户名或密码错误，请重试。"
[用户名和密码正确]
alt
[普通用户未批准]
提示 "用户尚未被管理员批准，无法登录。"
[普通用户已批准或管理员]
设置当前登录用户
提示 "登录成功"
切换到主界面
User
MainWindow
ItemManager
UserManager

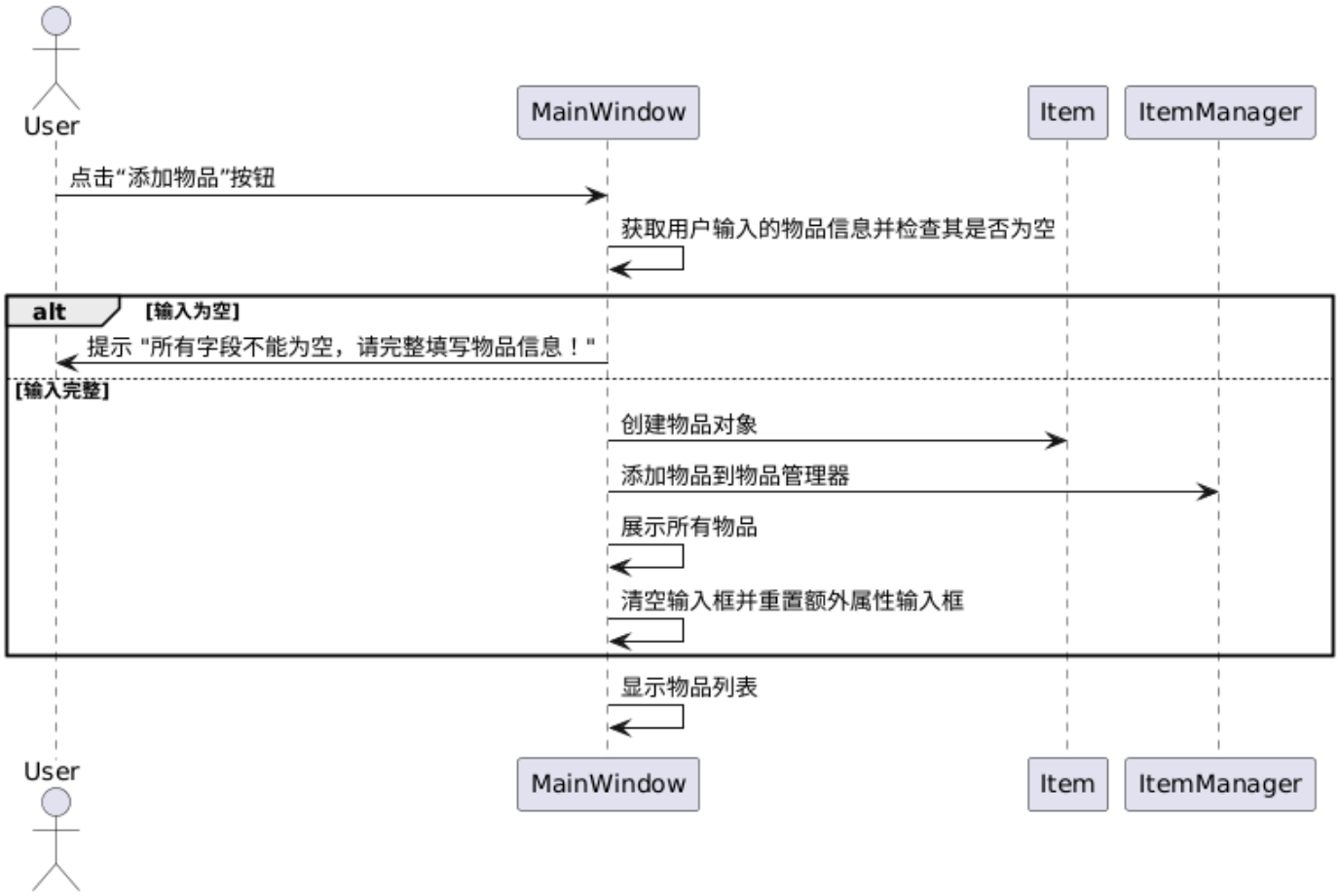
2.5.4 退出登录



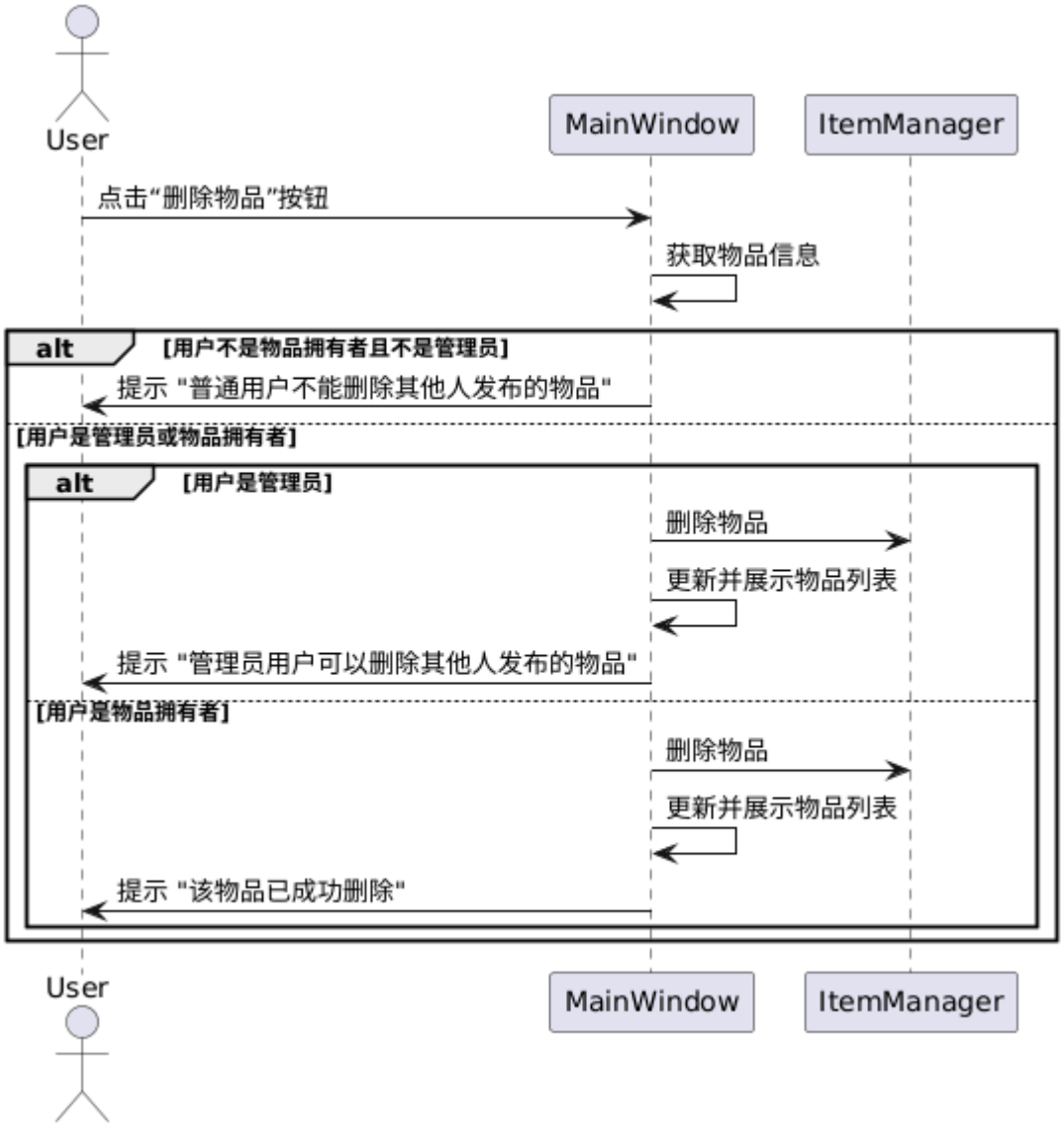
2.5.5 注销账号



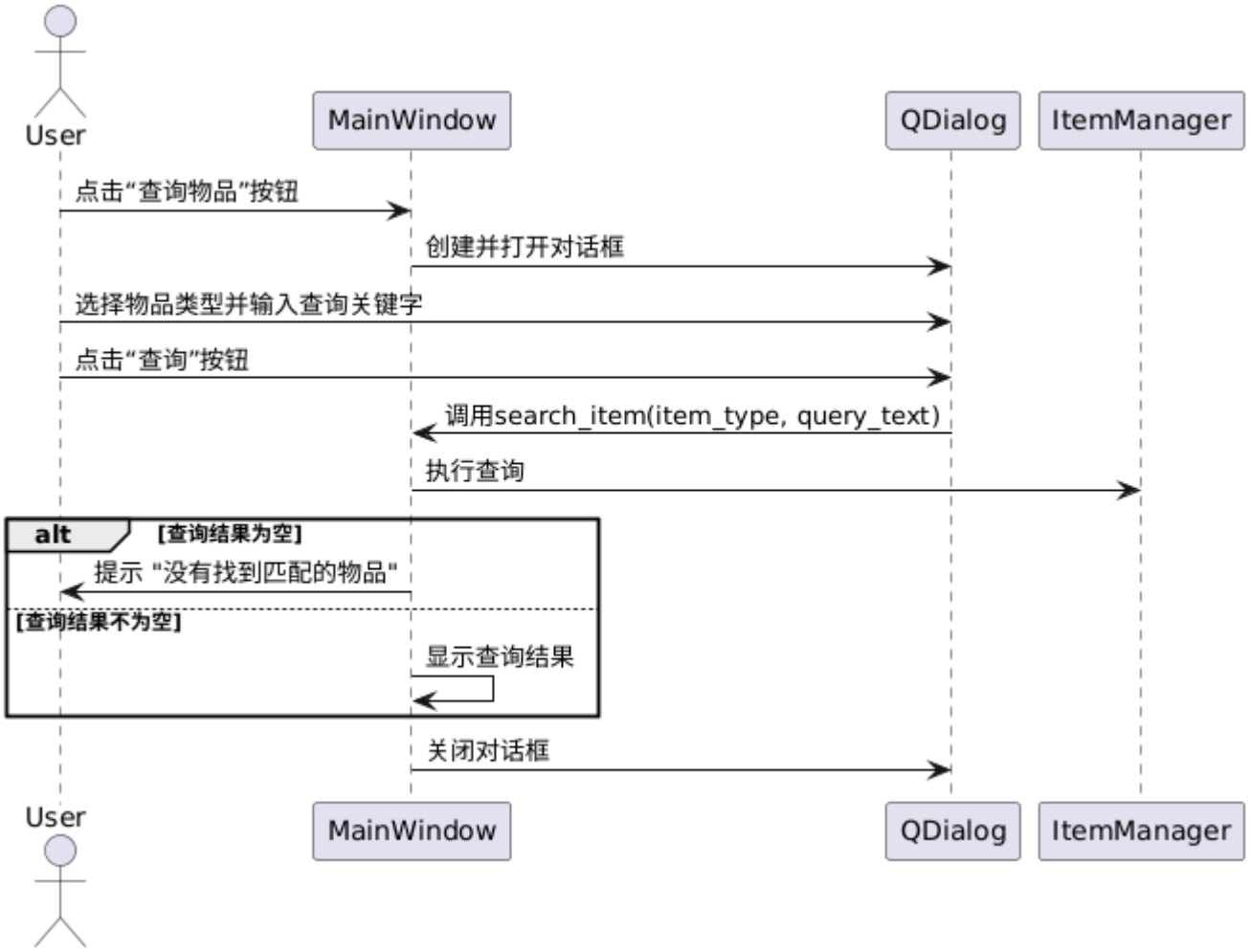
2.5.6 添加物品


User
MainWindow
Item
ItemManager
点击“添加物品”按钮
获取用户输入的物品信息并检查其是否为空
alt
[输入为空]
提示 "所有字段不能为空，请完整填写物品信息！"
[输入完整]
创建物品对象
添加物品到物品管理器
展示所有物品
清空输入框并重置额外属性输入框
显示物品列表
User
MainWindow
Item
ItemManager

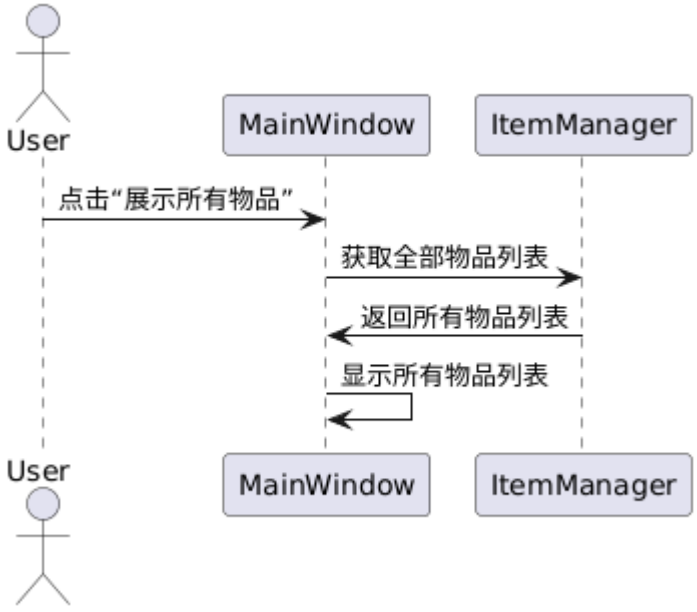
2.5.7 删除物品


User
MainWindow
ItemManager
点击“删除物品”按钮
获取物品信息
alt
[用户不是物品拥有者且不是管理员]
提示 "普通用户不能删除其他人发布的物品"
[用户是管理员或物品拥有者]
alt
[用户是管理员]
删除物品
更新并展示物品列表
提示 "管理员用户可以删除其他人发布的物品"
[用户是物品拥有者]
删除物品
更新并展示物品列表
提示 "该物品已成功删除"
User
MainWindow
ItemManager

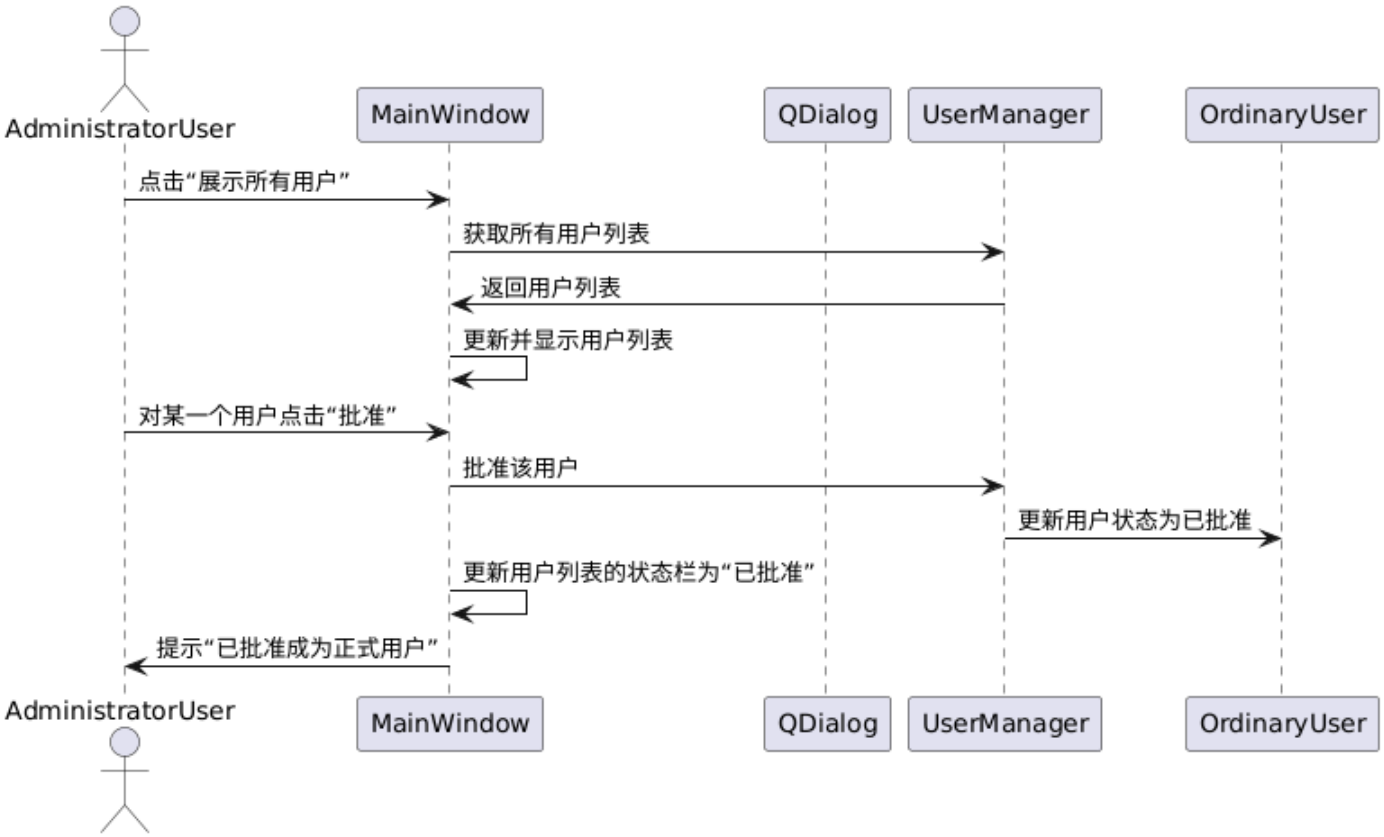
2.5.8 查询物品



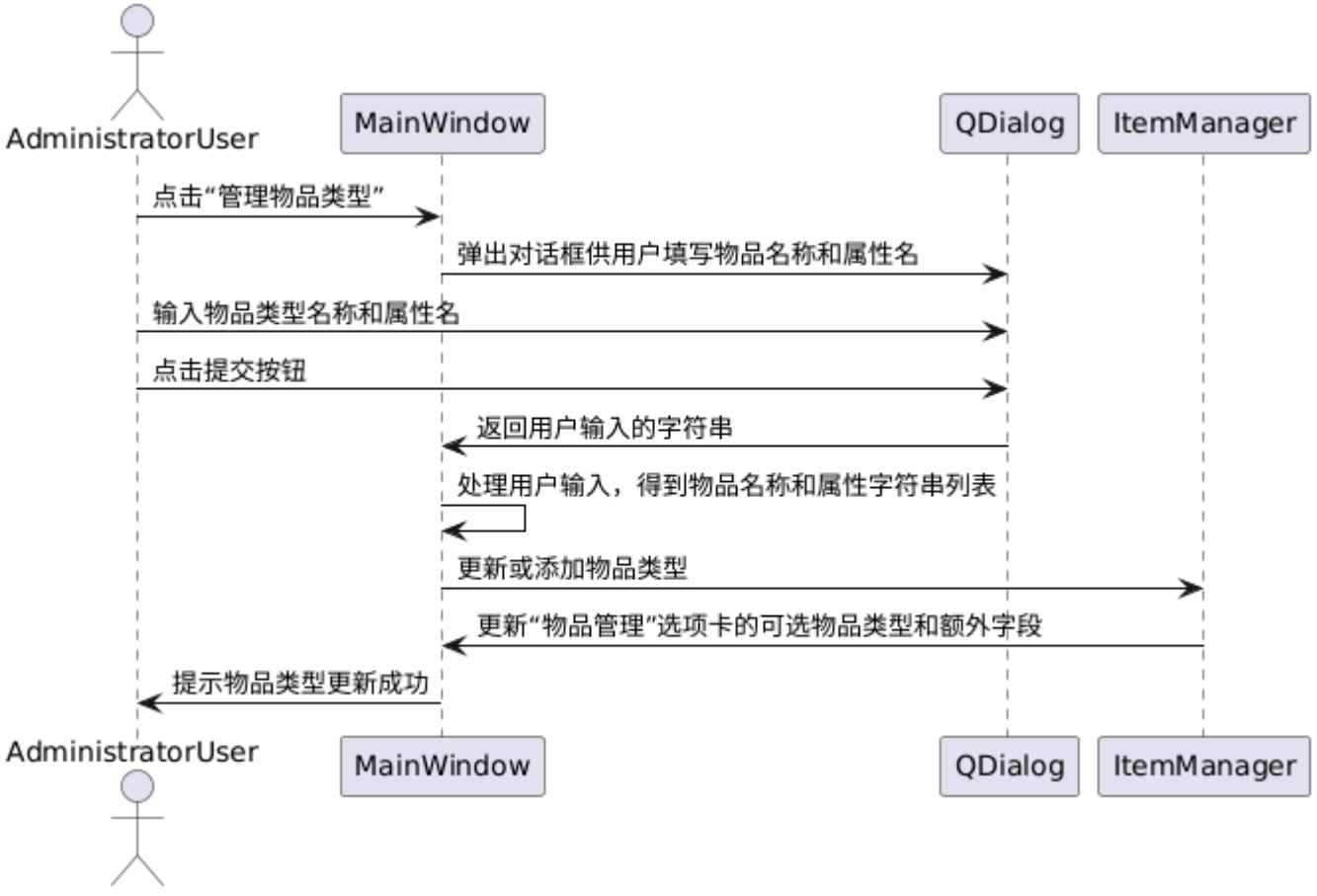
2.5.9 展示所有物品



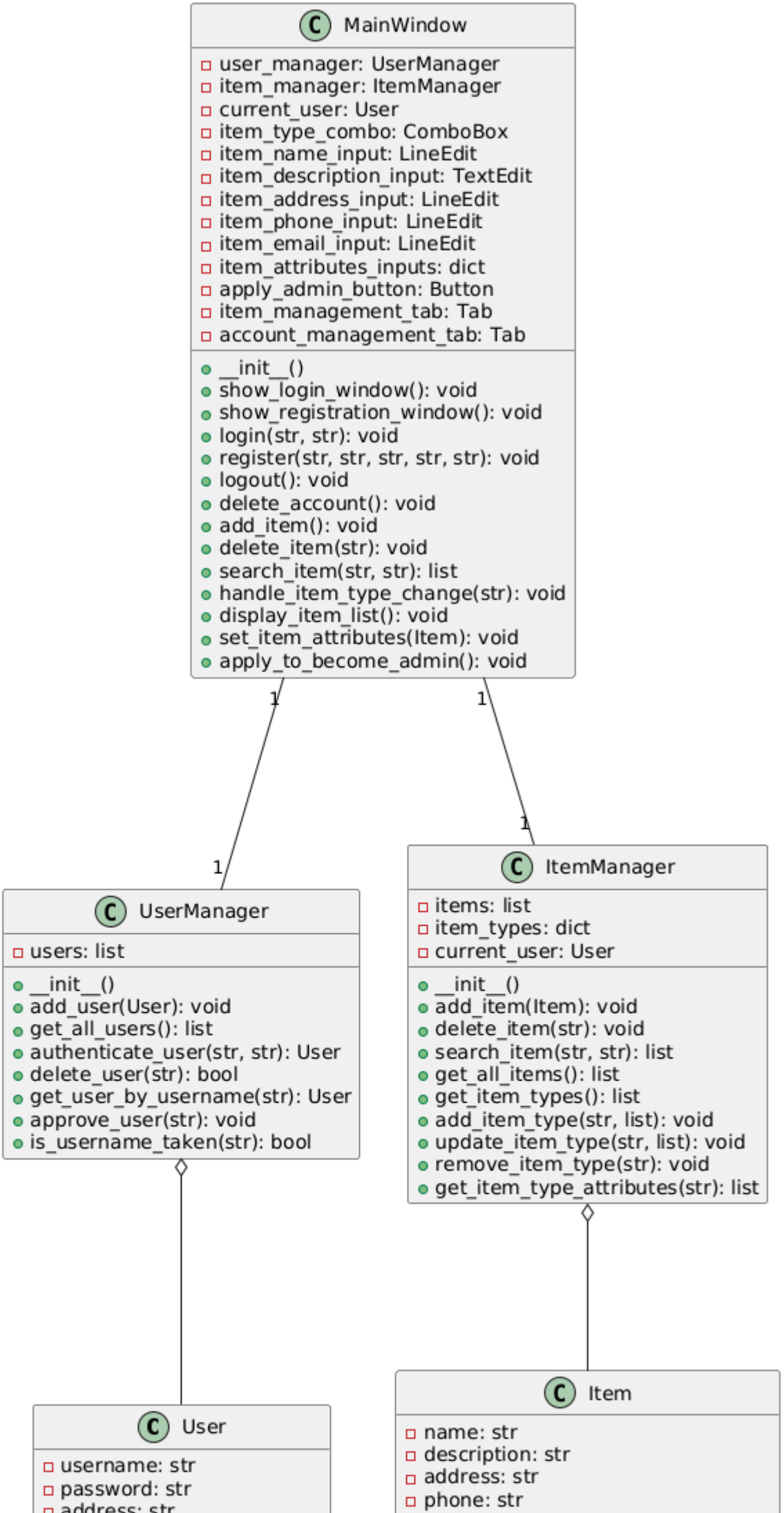
2.5.10 批准普通用户登录

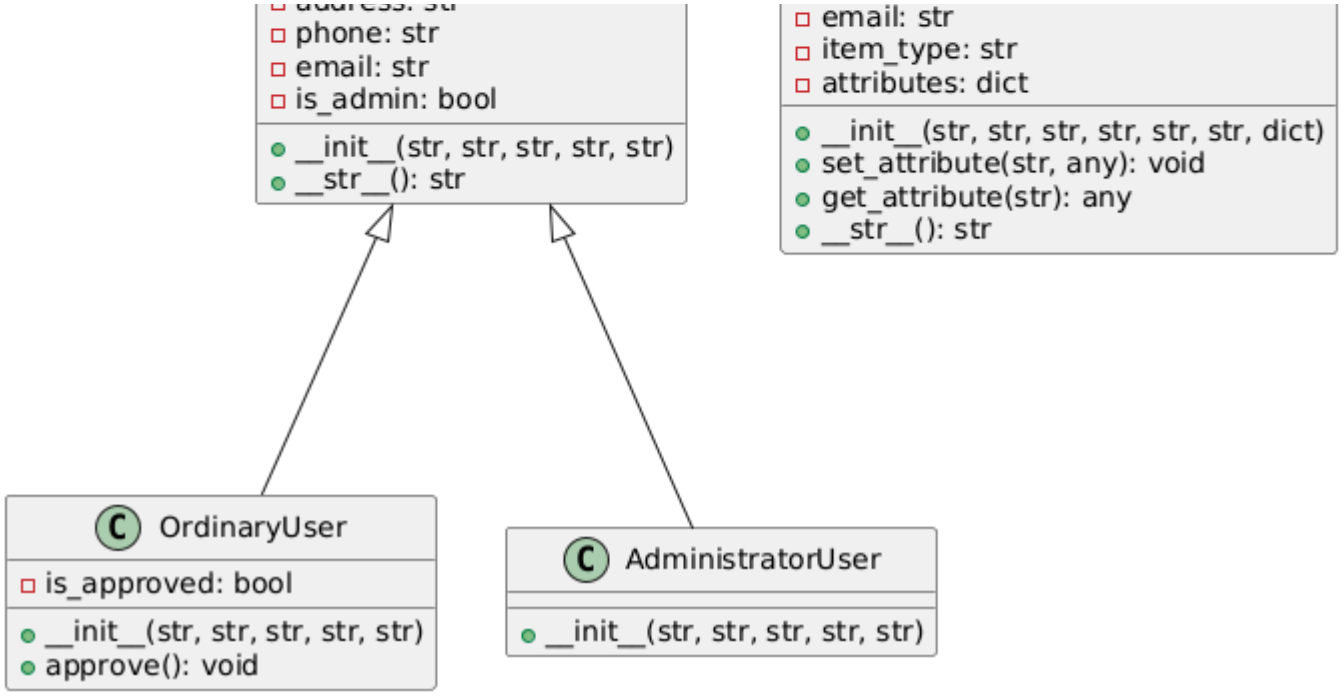


2.5.11 添加或修改物品类型和属性



2.6 类图


MainWindow
- user_manager: UserManager
- item_manager: ItemManager
- current_user: User
- item_type_combo: ComboBox
- item_name_input: LineEdit
- item_description_input: TextEdit
- item_address_input: LineEdit
- item_phone_input: LineEdit
- item_email_input: LineEdit
- item_attributes_inputs: dict
- apply_admin_button: Button
- item_management_tab: Tab
- account_management_tab: Tab
+ __init__()
+ show_login_window(): void
+ show_registration_window(): void
+ login(str, str): void
+ register(str, str, str, str, str): void
+ logout(): void
+ delete_account(): void
+ add_item(): void
+ delete_item(str): void
+ search_item(str, str): list
+ handle_item_type_change(str): void
+ display_item_list(): void
+ set_item_attributes(Item): void
+ apply_to_become_admin(): void
1
1
1
1
UserManager
- users: list
+ __init__()
+ add_user(User): void
+ get_all_users(): list
+ authenticate_user(str, str): User
+ delete_user(str): bool
+ get_user_by_username(str): User
+ approve_user(str): void
+ is_username_taken(str): bool
ItemManager
- items: list
- item_types: dict
- current_user: User
+ __init__()
+ add_item(Item): void
+ delete_item(str): void
+ search_item(str, str): list
+ get_all_items(): list
+ get_item_types(): list
+ add_item_type(str, list): void
+ update_item_type(str, list): void
+ remove_item_type(str): void
+ get_item_type_attributes(str): list
User
- username: str
- password: str
Item
- name: str
- description: str
- address: str
- phone: str



1. 类的分析

该类图包含以下几个类：

- **User**：这是一个基类，表示系统中的用户。它是**普通用户（OrdinaryUser）和管理员用户（AdministratorUser）**的父类，包含了用户的通用属性和方法。
- **OrdinaryUser**：这是**User**类的子类，表示普通用户。它继承了**User**类的所有属性和方法，并且可以增加特定的属性和方法（例如普通用户的特定权限和行为）。
- **AdministratorUser**：这是**User**类的子类，表示管理员用户。它继承了**User**类的所有属性和方法，并且可以增加特定的属性和方法（例如管理员用户的特定权限和行为）。
- **UserManager**：这是一个管理用户的类，负责处理与用户相关的操作（如添加、删除用户等）。它与**User**类之间存在“聚合（o===）”关系，表示它管理多个用户实例。
- **Item**：这是一个表示物品的类，负责物品相关的操作。它与**ItemManager**类之间也存在“聚合（o===）”关系，表示它管理多个物品实例。
- **ItemManager**：这是一个管理物品的类，负责物品的添加、删除、查询等操作。它与**Item**类之间也有“聚合”关系，表示它管理多个物品实例。
- **MainWindow**：这是主界面类，它与**UserManager**和**ItemManager**类之间有一对一的关联（"1"-up--"1"），表示主界面通过这两个管理类来处理用户和物品的管理任务。

2. 类之间的关系

- **User**与**OrdinaryUser**和**AdministratorUser**之间的关系：
  - **OrdinaryUser**和**AdministratorUser**都继承自**User**类，形成了一个**继承（<|===）关系。也就是说，OrdinaryUser和AdministratorUser**都共享**User**类的基本属性和方法，并且可以扩展其特有的属性和行为。
- **UserManager**与**User**之间的关系：
  - **UserManager**与**User**类之间的关系是**聚合（o===）**，表示**UserManager**类聚合了多个**User**实例。**UserManager**负责管理所有用户，并提供添加、删除和查找等操作。
- **ItemManager**与**Item**之间的关系：
  - **ItemManager**与**Item**类之间也是**聚合（o===）**，表示**ItemManager**类聚合了多个**Item**实例。**ItemManager**负责管理物品，并提供物品的操作（如添加、删除、查询等）。
- **MainWindow**与**UserManager**和**ItemManager**之间的关系：
  - **MainWindow**与**UserManager**和**ItemManager**之间存在一对一的关系（"1"-up--"1"）。这意味着每个**MainWindow**实例与一个**UserManager**实例和一个**ItemManager**实例关联。主界面通过这两个类来进行用户和物品的管理操作。

2.6 尚未解决的问题

1. 数据持久化

- **问题**： 当前系统未涉及数据持久化，所有的用户、物品和设置数据仅在内存中存储。即便系统的功能设计完备，一旦系统关闭，所有数据都会丢失。
- **解决方案**：为了将系统转化为一个可用的应用，必须加入数据库支持，确保数据可以持久化存储。常见的解决方案包括使用关系型数据库（如MySQL或SQLite）来存储用户信息、物品信息等，或者使用文件系统存储（例如JSON、XML或CSV格式）。

2. 用户验证与安全

- **问题**： 虽然有登录功能，但未涉及用户的密码加密、安全认证、账户安全等问题。普通用户和管理员权限的验证较为简单，且没有防止恶意用户滥用系统的安全措施。

- **解决方案**：应考虑实现用户密码加密存储（如使用哈希算法如SHA-256），以及更复杂的身份验证机制（如OAuth、双重认证等）。此外，还应加入账户锁定机制、防止SQL注入等安全防护措施，确保系统的安全性。

3. 界面与用户体验

- **问题**： 系统当前的界面设计可能较为简陋，尤其是对于复杂操作和数据展示（如物品列表、用户管理等），用户的体验可能不佳。

- **解决方案**：需要进一步优化UI设计，提高系统的易用性。例如，可以为每个功能添加更多的提示信息和反馈消息，使用户能够更加方便地理解操作。此外，还可以考虑使用现代UI框架（如Qt、React等）来提升系统界面的美观度和响应性。

4. 权限管理

- **问题**： 目前系统的管理员权限管理过于简单，管理员可以执行多种操作，但没有明确的权限控制机制（如分级管理员、特定功能权限等）。

- **解决方案**：应进一步细化权限管理，允许为不同的用户分配不同的权限。例如，可以设置不同级别的管理员权限，或者将某些操作权限限制在特定的用户组或角色上。可以采用基于角色的访问控制（RBAC）来实现这一功能。

5. 异常处理与日志记录

- **问题**： 系统的异常处理不够完善，错误和异常没有足够的捕捉和反馈机制。此外，系统的操作日志和错误日志也未记录，这可能在发生问题时导致调试困难。

- **解决方案**：需要加入完善的异常处理机制，确保系统在出现问题时能够友好地反馈给用户，并避免系统崩溃。同时，需要实现日志记录功能，记录用户的操作行为和系统运行的异常，以便开发人员进行调试和维护。

6. 系统扩展性与维护

- **问题**： 目前系统的架构看起来相对简单，可能在未来随着功能扩展而变得难以维护和扩展。例如，物品类型和属性的管理可能在后续增加新特性时显得不够灵活。

- **解决方案**：可以通过模块化设计提高系统的扩展性，确保系统能够随着需求变化进行灵活扩展。通过采用设计模式（如工厂模式、策略模式等）来管理不同功能模块，提升代码的可维护性和可扩展性。

7. 性能优化

- **问题**： 随着用户量和物品量的增加，系统可能面临性能瓶颈，特别是在展示所有物品或查询物品时，若数据量过大，响应时间可能变长。
- **解决方案**：需要考虑数据库优化（如索引、查询缓存等）和前端性能优化（如分页加载、大数据量处理等）。还可以通过引入异步处理和并行计算来提高系统的响应速度。

8. 测试与质量保障

- **问题**： 当前的系统并未体现出测试框架，系统的质量保障机制较弱，可能导致在开发过程中出现错误未被及时发现。
- **解决方案**：应增加单元测试、集成测试和系统测试等自动化测试手段，确保系统在不同条件下的可靠性。同时，应加入持续集成（CI）和持续交付（CD）机制，保证代码的高质量和稳定性。

9. 跨平台支持

- **问题**： 当前系统是否可以在不同的操作系统上运行未提及。若是使用特定平台的技术（如Windows的某些库），可能导致系统无法在其他平台（如Linux或MacOS）上运行。
- **解决方案**：如果要实现跨平台支持，应选择适合跨平台开发的框架（如Qt或Electron）进行开发，或通过容器化技术（如Docker）确保系统在不同平台上的一致性。

10. API与第三方集成

- **问题**： 目前系统没有提到与外部服务或API的集成，例如与支付系统、电子邮件发送服务或其他系统的集成。
- **解决方案**：可以在系统中加入与第三方服务的集成功能（如通过API与外部支付平台或通知系统进行通信），这将使系统功能更加丰富，提升用户体验。