

软件需求规格说明书

**项目名称：** 基于以太坊智能合约+NestJS 微服务的

游戏社区与任务市场系统

学生姓名 闵俊涛

学 号 202202083

学 院 区块链学院

年 级 2022级

专 业 区块链技术

指导教师 邹林薏

完成日期 2024年9月28日

目录

**[一、引言 1](#_Toc977)**

[1. 开发背景及目的 1](#_Toc20923)

[2. 术语和缩写词 1](#_Toc13592)

[3. 参考资料 1](#_Toc17058)

[4. 版本信息 2](#_Toc22103)

**[二、系统描述 2](#_Toc31919)**

[1. 用户简介 2](#_Toc25073)

[2. 业务流程 2](#_Toc29500)

**[三、系统功能需求 3](#_Toc24147)**

[1. 功能总体需求 3](#_Toc22935)

**[四、非功能性需求 7](#_Toc9698)**

[1. 性能需求 7](#_Toc27571)

[2. 易用性 7](#_Toc11095)

[3. 安全性与可靠性 8](#_Toc17809)

**基于以太坊智能合约+NestJS 微服务的**

**游戏社区与任务市场系统需求规格说明书**

**一、引言**

**1. 开发背景及目的**

随着区块链技术的蓬勃发展和游戏社区的不断壮大，传统游戏社区存在的中心化管理、交易不透明等问题日益凸显。本项目旨在构建基于以太坊智能合约和 NestJS 微服务架构的游戏社区与任务市场系统，利用以太坊智能合约实现去中心化的任务发布、接受与结算，确保交易透明性和安全性，采用 NestJS 微服务架构提高系统可扩展性和可维护性，为玩家提供丰富多样的游戏任务和社区活动，满足不同玩家需求。

**2. 术语和缩写词**

（1）以太坊智能合约：基于以太坊区块链平台的智能合约技术，用于实现去中心化的交易和业务逻辑。

（2）NestJS：一种用于构建高效、可扩展的 Node.js 服务器端应用程序的框架。

（3）gRPC：一种高性能、开源的远程过程调用（RPC）框架。

（4）Kafka：一种高吞吐量的分布式发布订阅消息系统。

（5）TDD：测试驱动开发，一种软件开发方法，先编写测试用例，再编写实现代码。

（6）CI/CD：持续集成/持续部署，一种软件开发实践，通过自动化工具实现代码集成、测试和部署。

（7）社区积分：系统内的虚拟货币，可通过完成任务、参与社区活动等方式获取，用于交易或兑换物品。

（8）任务发布者：在系统中发布任务的用户。

（9）任务领取者：在系统中领取并完成任务的用户。

**3. 参考资料**

[1]以太坊基金会.以太坊官方文档[OL].https://ethereum.org/en/developers/docs/,2024

[2]NestJS 团队.NestJS 官方文档[OL].https://docs.nestjs.com/,2024

[3]gRPC 团队.gRPC 官方文档[OL].https://grpc.io/docs/,2024

[4]Apache Kafka 团队.Kafka 官方文档[OL].https://kafka.apache.org/documentation/,2024

**4. 版本信息**

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本 | 发布时间 | 更新人 | 更新内容 |
| V1.0 | 2024年9月28日 | 闵俊涛 | 首次完成文档。 |
|  |  |  |  |
| ... | ... | ... | ... |

**二、系统描述**

**1. 用户简介**

1）游客：在实际应用中，尚未注册过本游戏社区平台账号的用户。游客可以浏览部分平台信息，但无法使用其他功能，游客可以通过注册操作以进一步使用本游戏社区与任务市场系统。

2）玩家：在实际应用中，已经注册而拥有平台账号的用户。玩家通过登录功能对本平台进行使用，可以发布和接受任务、参与社区活动、进行积分交易等，不同玩家可以根据自身需求参与各种活动。

3）任务发布者：在实际应用中，拥有发布任务权限的用户。任务发布者通过登录功能对本平台进行使用，可以发布任务并管理任务的执行和奖励发放。

4）系统管理员：在实际应用中，拥有管理本平台的权限，负责本平台的正常运行。系统管理员负责维护用户信息、任务信息以及社区管理等。

**2. 业务流程**

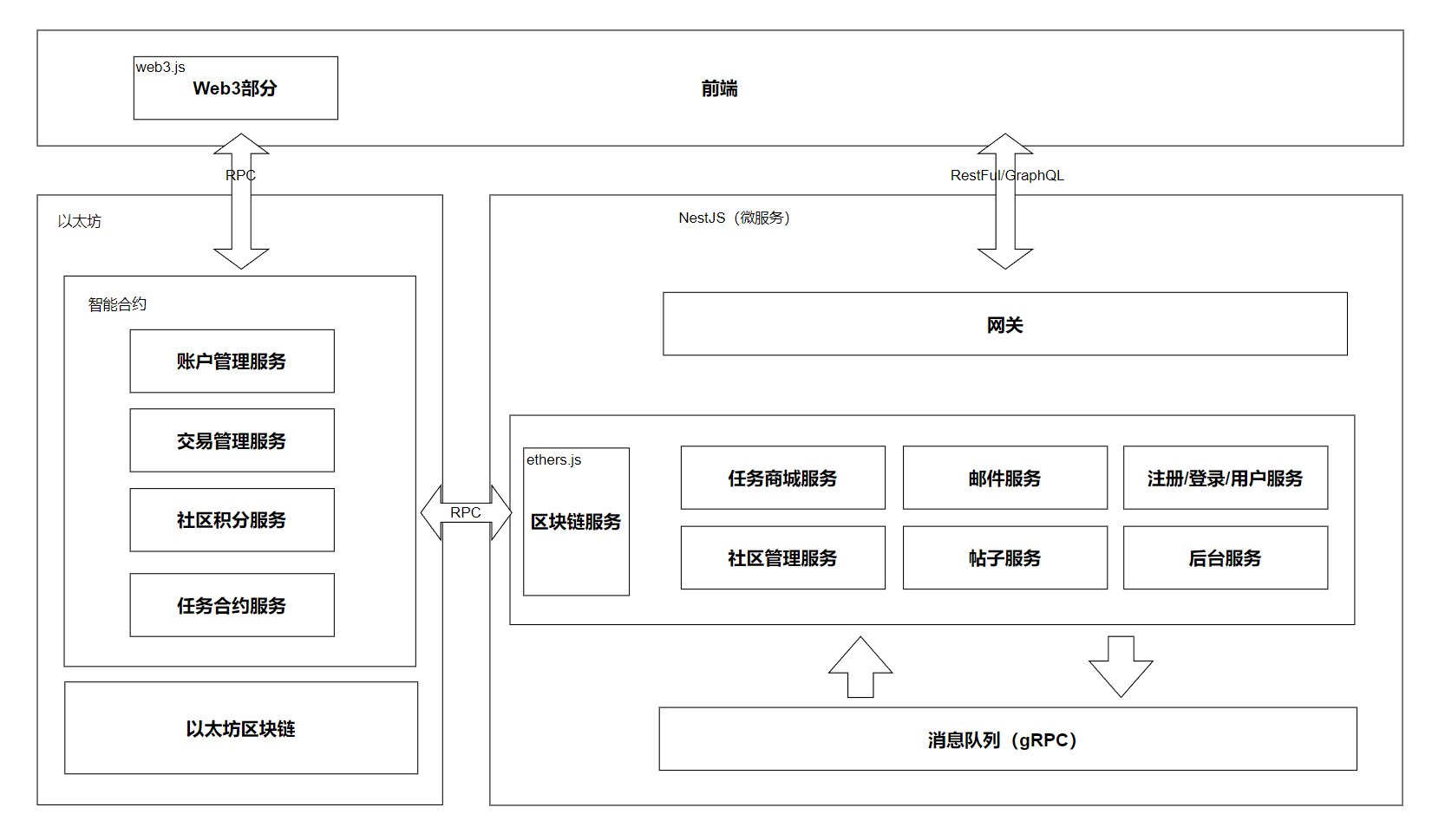
用户注册和登录后，任务发布者可发布任务，玩家在任务市场领取任务，完成后提交申请，任务发布者确认，系统发放奖励。用户还可在游戏社区发布帖子、参与民主活动，通过完成任务和活动获取社区积分，在积分市场进行交易，也可管理个人账户、查看任务交易记录、进行登录认证、管理积分和充值。

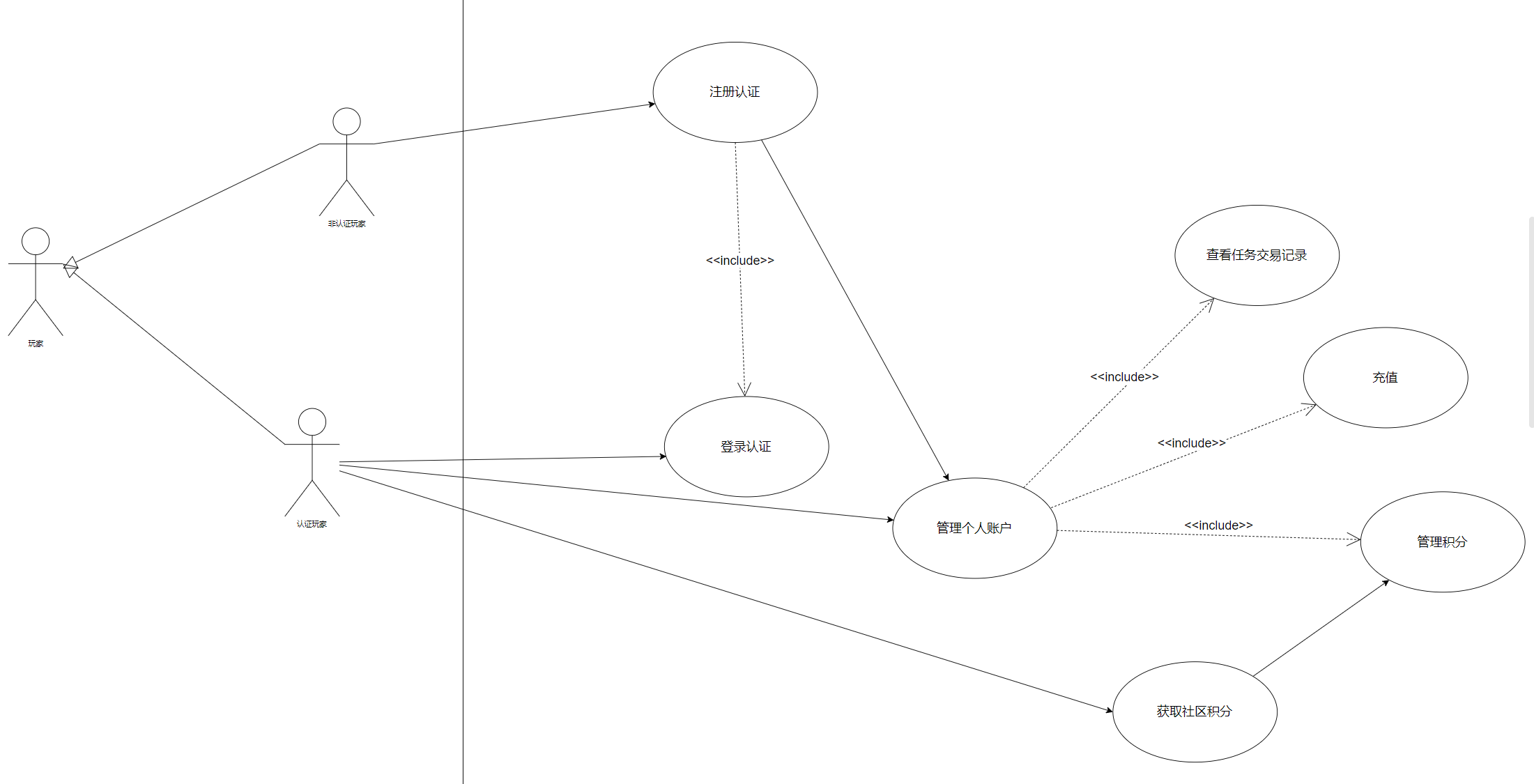
**三、系统功能需求**

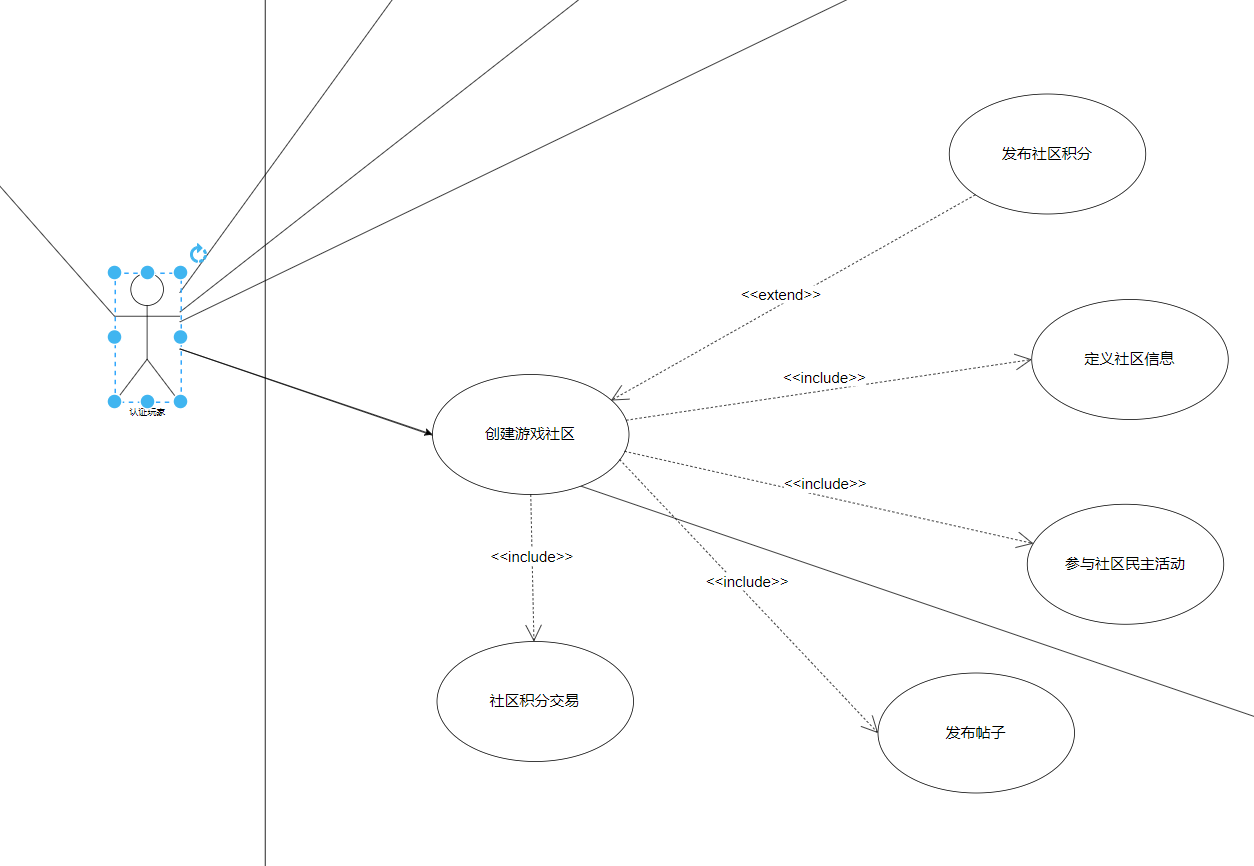
**1. 功能总体需求**

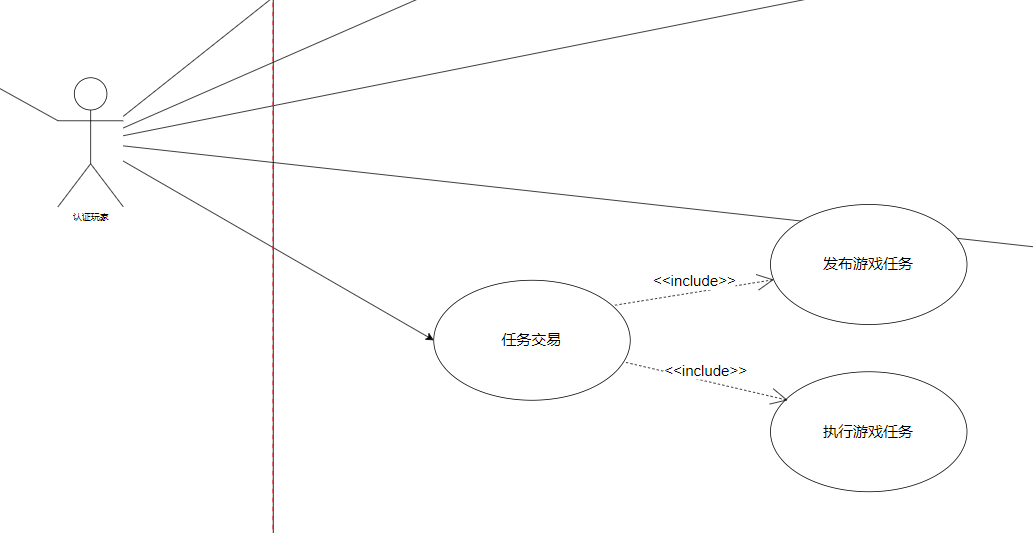
基于区块链构建去中心化游戏社区平台，确保任务发布、接受与结算的透明性和安全性。搭建任务市场系统，提供全流程管理功能，支持多种任务类型。实现 NestJS 微服务架构，独立开发各模块，确保微服务高效通信。提升系统安全性与可扩展性，优化用户体验。

下面是系统应用架构设计图:

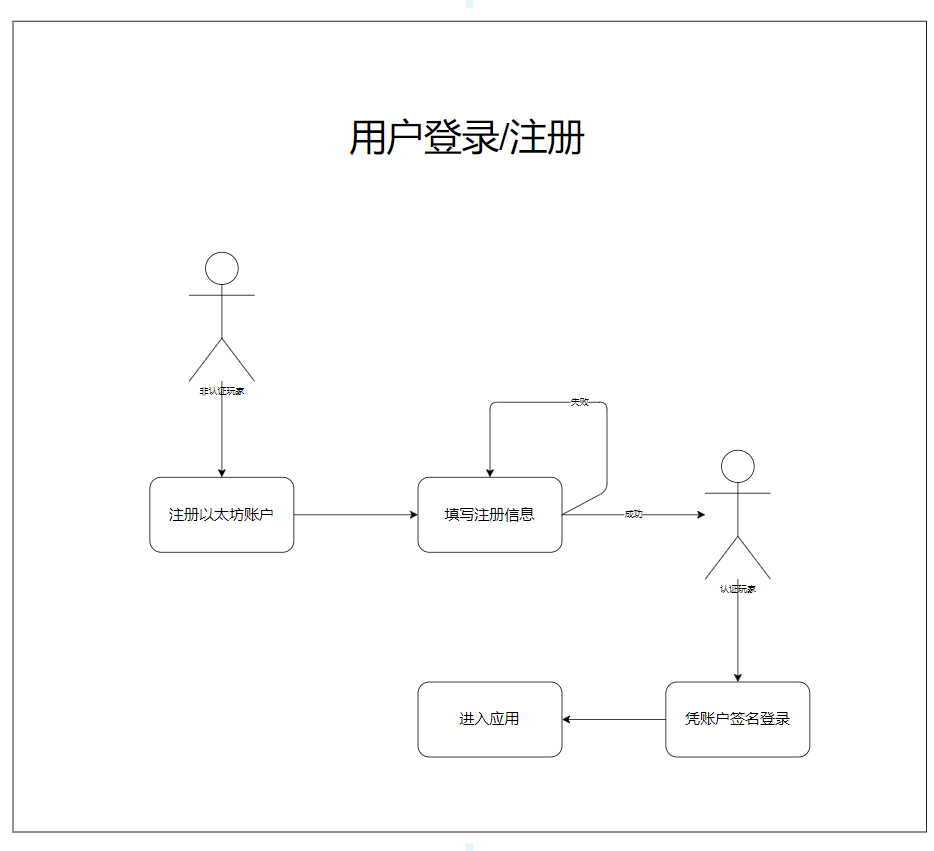
1. **用例建模**





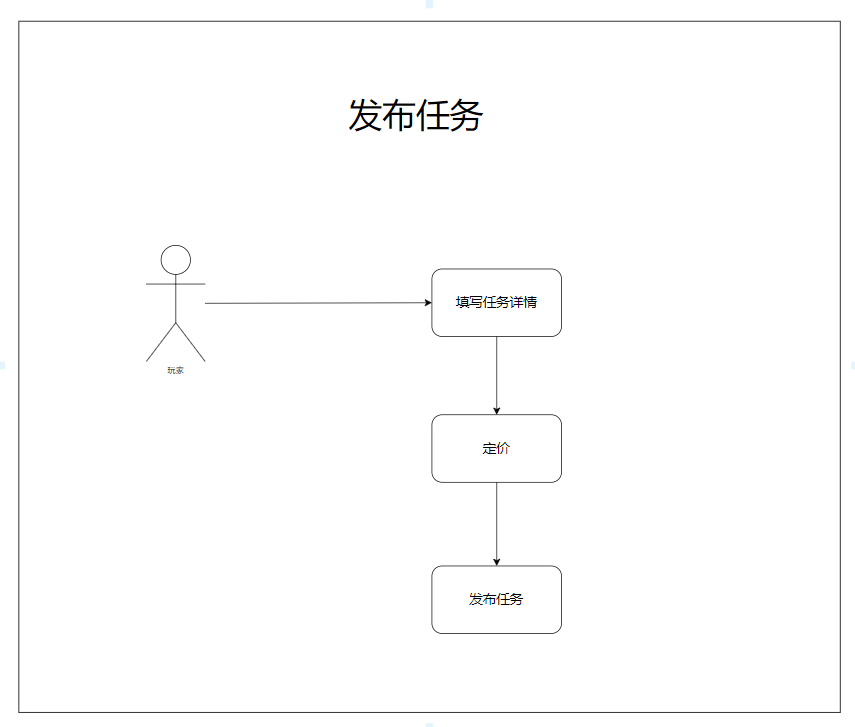


**非认证玩家注册认证成为认证玩家：**

非认证玩家希望获得更多平台功能和权限时，可进入注册认证页面填写信息提交申请，系统审核通过后转变为认证玩家，若已认证注册会提示已注册。

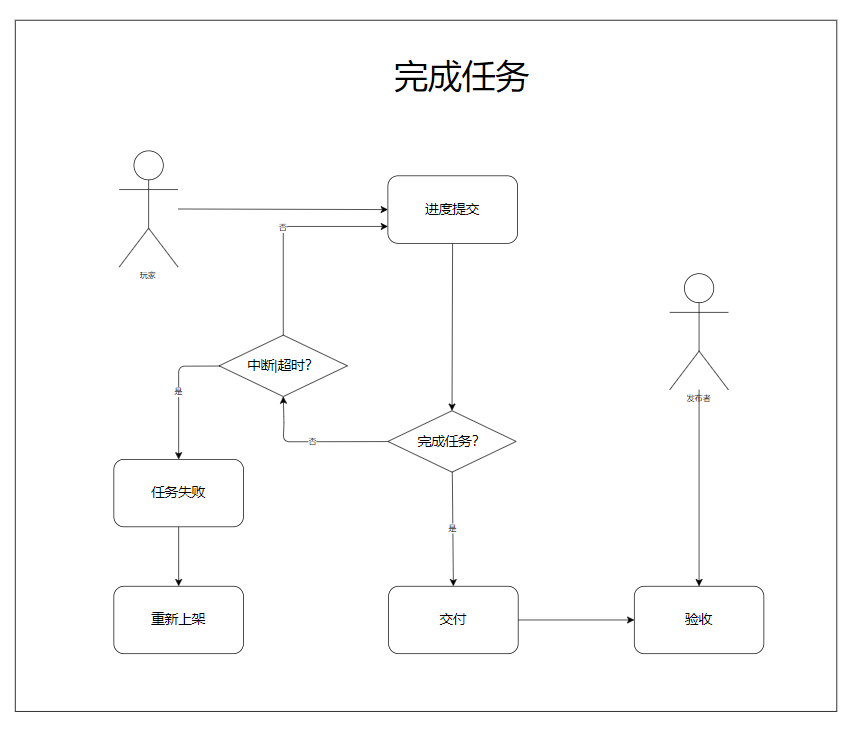
**认证玩家发布游戏任务：**

认证玩家有特定任务需求时，进入发布任务页面填写描述和奖励等信息提交申请，系统审核通过后任务在平台展示，必须注册认证并加入对应社区才能发布。



**玩家执行游戏任务：**

玩家看到感兴趣任务时，可查看任务列表并选择任务，按照要求执行后提交完成申请，完成后可获得奖励，任务发布者可确认完成情况，必须注册认证并加入对应社区才能执行。

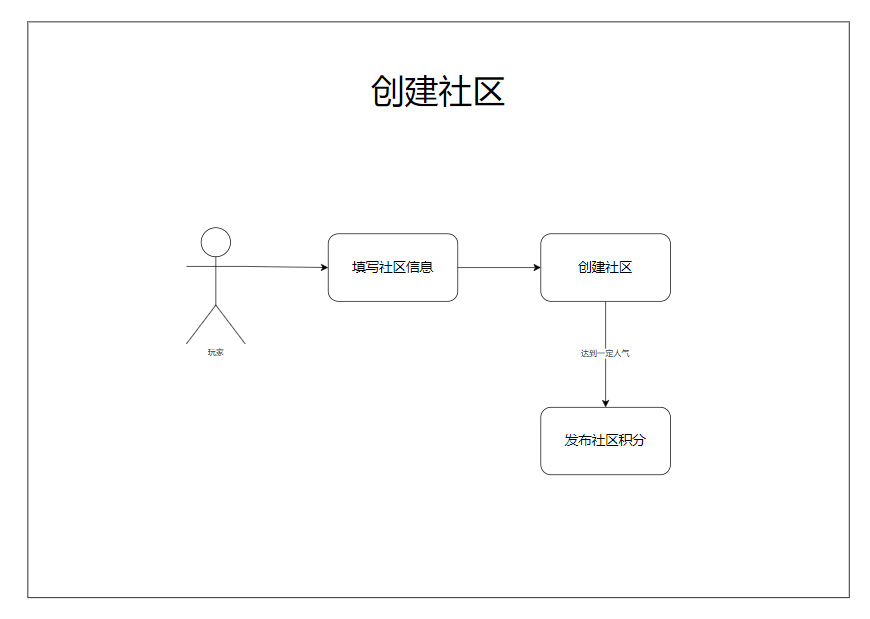


**认证玩家创建游戏社区：**

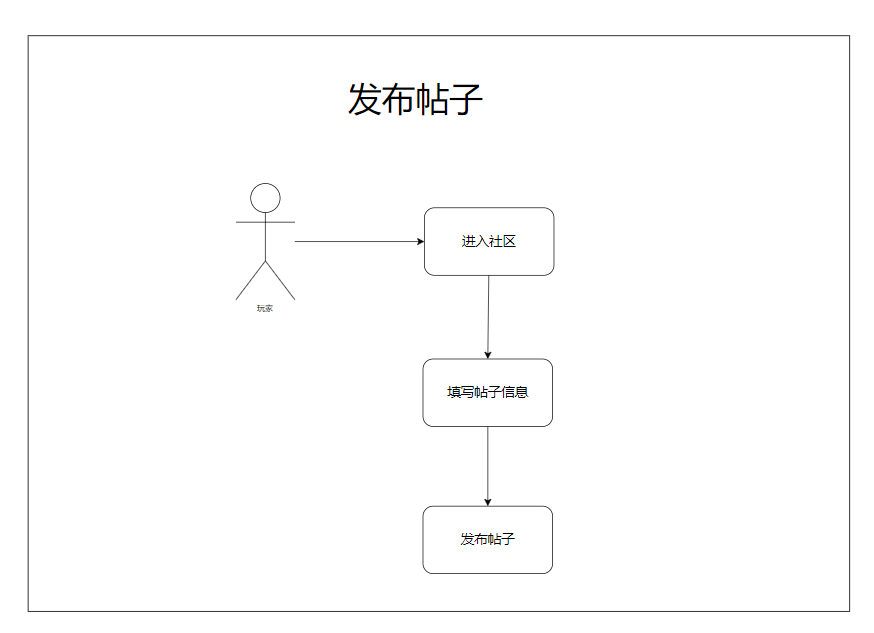
认证玩家希望创建特定主题社区时，进入创建页面填写名称和主题等信息提交申请，创建成功后其他玩家可加入，必须注册认证并加入对应社区才能创建。

**玩家发布社区积分：**

认证玩家希望奖励或激励其他玩家时，进入发布积分页面填写数量和原因等信息提交申请，发布成功后其他玩家可获取，需认证且创建社区积攒人气。

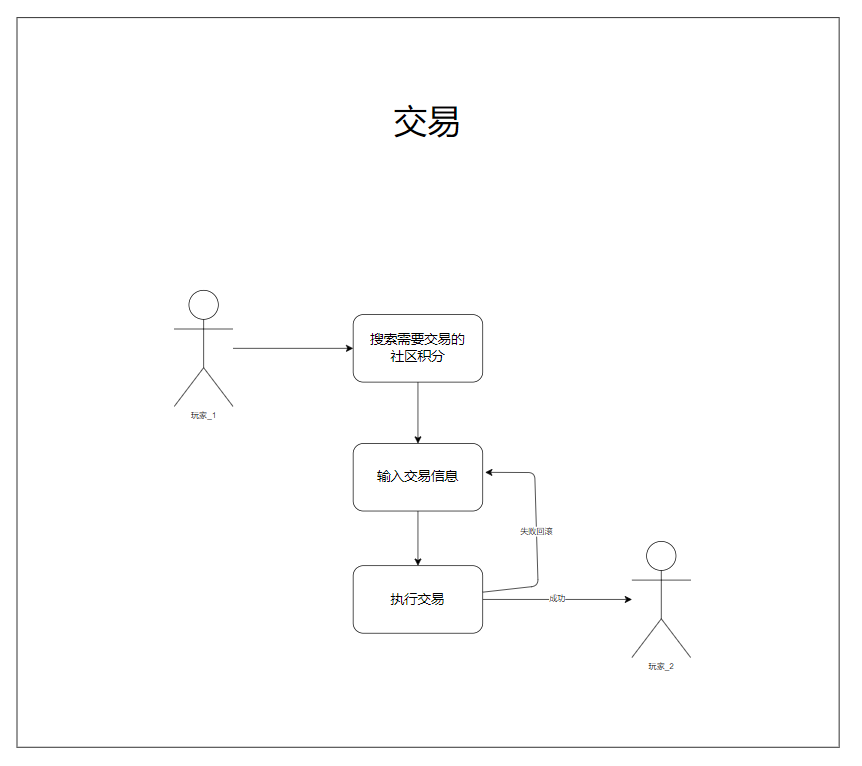


**玩家发布帖子：**

玩家有分享欲望时，进入发布帖子页面填写标题和内容等信息提交申请，发布成功后其他玩家可查看回复，必须加入社区才能发布。

**玩家进行社区积分交易：**

玩家有积分交易需求时，进入交易页面选择对象和方式，确认信息后提交申请，系统审核通过后交易完成，若对方未加入相应社区需先加入。



**四、非功能性需求**

**1. 性能需求**

- 系统响应时间：在高并发的情况下，系统的响应时间应严格控制在 3 秒以内，确保用户能够快速获得反馈，提升用户体验。对于关键操作，如任务发布、领取、完成确认等，响应时间应更短，以保证操作的流畅性。

- 并发处理能力：系统应具备强大的并发处理能力，能够稳定支持至少 1000 个并发用户。随着用户数量的增长，系统应能够轻松扩展，以适应更多的并发请求。可以通过优化服务器架构、采用负载均衡等技术手段来提高并发处理能力。

- 可扩展性：系统应设计为具有高度的可扩展性，能够轻松应对不断增长的用户数量和任务数量。可以采用微服务架构、分布式存储等技术，使得系统能够在不影响性能的情况下进行扩展。同时，系统的扩展应尽可能简单方便，降低维护成本。

- 任务操作处理时间：任务操作的处理时间应尽可能短，包括任务发布、领取、完成确认等。可以通过优化数据库设计、采用异步处理等技术手段来提高任务操作的处理速度。对于复杂的任务，系统应提供进度提示，让用户了解任务的处理状态。

**2. 易用性**

- 新用户引导：新用户注册后，系统应提供详细的使用教学，包括任务发布、领取、社区互动等基本操作。教学内容应使用通俗易懂的语言描述操作逻辑，避免使用专业术语，让用户能够快速上手。可以通过视频教程、图文说明等多种方式进行教学。

- 操作简化：系统的操作应尽可能简化，减少用户的操作步骤。对于常用操作，可以设置快捷方式，方便用户快速访问。同时，对于可能导致严重后果的操作按钮，应使用醒目的颜色标注，并在用户操作前进行提醒，以避免误操作。

- 用户界面设计：用户界面应设计得简洁、直观，易于操作。界面布局应合理，信息展示清晰，让用户能够快速找到所需的功能。同时，界面应具有良好的响应式设计，能够适应不同设备的屏幕尺寸，方便用户在不同设备上使用系统。

- 反馈机制：系统应提供及时的反馈机制，让用户了解操作的结果。对于成功的操作，应给予明确的提示；对于失败的操作，应给出详细的错误信息，帮助用户快速解决问题。

**3. 安全性与可靠性**

- 用户隐私保护：保障用户隐私信息的安全是系统的重要任务。系统应采用加密存储技术，对用户的账号密码、个人信息等隐私数据进行加密存储，防止数据泄露。同时，系统应严格控制用户数据的访问权限，只有经过授权的人员才能访问用户数据。

- 数据备份与恢复：系统应定期进行数据备份，以防止数据丢失。备份数据应存储在安全的地方，确保在系统出现故障时能够快速恢复数据。同时，系统应建立完善的数据恢复机制，能够在最短的时间内恢复系统的正常运行。

- 程序稳定性：降低程序崩溃的概率是提高系统可靠性的关键。系统应经过严格的测试，确保在各种情况下都能够稳定运行。可以采用自动化测试、压力测试等技术手段，发现并解决潜在的问题。同时，系统应建立完善的错误处理机制，当出现程序错误时，能够及时进行处理，保证系统的稳定性。

- 支付安全：为了保证用户支付与资金的安全，系统需要提高数据传输的安全性。可以采用加密传输技术，确保支付数据在传输过程中不被窃取。同时，系统应使用安全的支付方式进行支付，如第三方支付平台等，保证支付过程的安全可靠。系统还应保存好支付记录，以便在出现问题时进行查询和处理。

- 区块链安全：确保区块链上的交易安全、透明，防止篡改和欺诈行为。系统应采用先进的区块链技术，如加密算法、共识机制等，保证区块链上的交易数据不可篡改。同时，系统应建立完善的区块链监控机制，及时发现并处理异常交易行为。