

BrokerChain (academic) Testnet  
An Introduction & Manual  
(Chinese Version)

黃華威研究組 (HuangLab)

BlockIn @ Hong Kong

Copyright preserved<sup>®</sup>

2025 年 7 月 17 日



# 目 录

1	<i>BrokerChain</i> 区块链概述	1
1.1	BrokerChain (academic) 是什么?	1
1.2	BrokerChain (academic) Testnet	5
1.3	BrokerChain 有什么特别的地方?	7
1.4	BrokerChain (academic) Testnet 测 试网可能会遇到的风险 . . . . .	8
2	<i>BrokerChain</i> 区块链技术原理	9
2.1	BrokerChain 分片协议的原理 . . .	9
2.2	BrokerChain 网络可扩展性 . . . .	18
3	通证经济模型与激励机制	20
3.1	通证机制设计 . . . . .	20
3.2	长期出块奖励机制 . . . . .	21
3.3	共识节点激励函数设计 . . . . .	23
3.4	可能会遭受的攻击 . . . . .	27
4	使用手册	29
4.1	Researcher: 基于 BrokerChain 进 行链上应用生态的创新 . . . . .	29
4.2	Miner: 运行共识节点 . . . . .	30
4.3	Developer: 部署智能合约 . . . . .	41

4.4	User: 使用 BrokerChain Wallet 体验 dApps . . . . .	47
4.5	使用“水龙头”领取少量 BKC . . . . .	51
4.6	BrokerChain 的区块链浏览器 . . . . .	52
5	未来升级计划概述 . . . . .	53
6	代表性 <i>dApp Demo</i> 示例 . . . . .	54
6.1	BrokerFi: 构建在 BrokerChain 上的流动性质押 DeFi 协议 . . . . .	55
6.2	NFT-Fi: 构建在 BrokerChain 上的 NFT 流通性增强方案 . . . . .	63
6.3	DePIN-RWA: 算力设备代币化 . . . . .	137

# 第一章 BrokerChain 区块链概述

## 1.1 BrokerChain (academic) 是什么？

BrokerChain 最初是黄华威教授研究组 (HuangLab) 于 2022 年提出的分片区块链协议。2025 年 6 月，由 BlockInLab 团队<sup>1</sup>作为主体，与黄教授研究组合作，上线了 BrokerChain (academic) Testnet，即分片区块链 BrokerChain 的测试网。

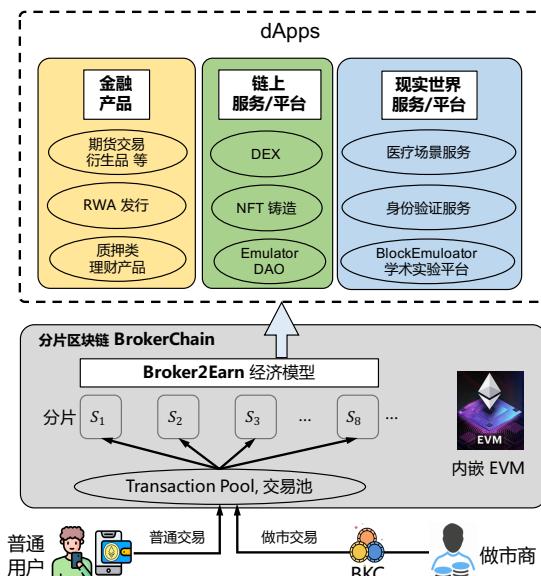


图 1.1: BrokerChain 区块链以及链上构建的 dApps 示例。

<sup>1</sup>X 账号: @blockinlab

如图1.1所示，位于底部的 BrokerChain 区块链是一个基础设施，可以用来构建上层各种去中心化应用（decentralized applications, dApps）。

### 1.1.1 BrokerChain 分片协议

BrokerChain 分片协议首次展示在论文“BrokerChain: A Cross-Shard Blockchain Protocol for Account/Balance- based State Sharding”[3]，并发表在国际会议（计算机网络领域顶会）INFOCOM 2022。该协议擅长处理分片区块链中广泛存在的“跨分片交易（Cross-shard Transactions, CTXs）”，旨在解决传统分片区块链系统中的两大核心问题：i) 跨分片交易比例过高，和 ii) 分片间交易负载不均衡。简单来讲，该协议通过动态调整账户状态来实现分片间的负载均衡并减少跨分片交易比例。该协议提出了创新的状态分片（state sharding）机制、账户网络分割（account-graph segmentation）技术和基于“做市商账户（broker accounts）”的协同设计，可显著提升分片区块链的可扩展性和交易处理效率。

BrokerChain 分片协议的论文发表后获得了学术界的关注，论文发表三年内获得了 180 多次引用。2025 年 3 月，研究团队发布了 BrokerChain 协议的扩展版论文，进一步深化了 BrokerChain 的研究内容，展示了优化之后的分片区块链设计。扩展版论文题目是“BrokerChain: A Blockchain Sharding Protocol by Exploiting Broker Accounts”[5]，于 2025 年 3 月发表在计算机网络领域的顶刊 IEEE Trans. on Networking (ToN)。

### 1.1.2 针对原始协议的优化、扩展的技术方案

尽管 BrokerChain 原始协议在处理跨分片交易时展示了巨大的优势，但是它仍然存在若干技术问题尚未解决。为此，研究团队持续对原始协议进行技术扩展与优化。相关的后续代表性研究总结如下。

## 1. BrokerChain 协议的激励机制 Broker2Earn [1]

- 为解决做市商账户 (broker accounts) 来源的问题，研究组设计了为 BrokerChain 分片区块链招募 broker 账户的激励机制。该机制提出的“随机舍入”算法可同时兼顾最大化 Broker 账户的质押收益与分片区块链的系统内部的通证流动性。
- Broker 账户 **质押闲置通证** 可赚取稳定得收益，同时可帮助底层分片区块链减少跨分片交易比例。
- 该激励机制已整合至 BrokerChain 区块链的链上 DeFi 应用 BrokerFi (详见后文第 6.1 章节所述)。

## 2. 分片间账户迁移方案 (Fine-tuned Account Migration [2])

- 分片区块链在分片重组的时候，会牵涉到账户的跨分片迁移，这就需要提出一个方案来满足这个偏工程实现的需求。
- 在账户的跨分片迁移的过程中，为了减轻账户迁移对其关联交易的影响，作者提出了一种使用细粒度锁的账户迁移协议 (Fine-tuned Lock)。
- 为了实现账户迁移，作者设计了新的账户状态和区块的数据结构。并且对传统的 relay transaction 机制 [6] 进行了修改，用于处理账户迁移过程中可能出现的迁移失败情况。

## 3. 分片间共识公平性改进 (Justitia 方案 [8])

- 相较于交易池中的片内交易 ITXs (Intra-shard Transactions)，跨分片交易 (CTXs) 需要在多个分片参与共识，因此 CTXs 的手续费会被拆分成多份，这会导致 CTXs 延迟上链。为了保障跨

分片交易 CTXs 相对于片内交易 ITXs 的公平性，作者提出了动态手续费分配规则，并参考合作博弈的 Shapley Value 策略，提升了 CTXs 在交易池中排队参与共识的公平性。而且将方案扩展至多输入多输出（MIMO, multiple-input multiple-output）类型的 CTXs。

- 作者通过严格的理论分析证明了 Justitia 机制在分片区块链中能够保证安全性、原子性和公平性，尤其是显著减少了 CTX 的排队延迟。
- 作者在开源分片区块链实验平台 BlockEmulator [4] 上实现了 Justitia 机制，并基于以太坊的历史交易数据进行了广泛的性能评估。实验结果表明，Justitia 不仅显著降低了 CTXs 的排队延迟，还避免了系统高额补贴导致的经济通胀问题，证明了其面向分片区块链系统的实用性。

#### 4. 开源区块链仿真实验平台 BlockEmulator [4]

- 黄华威研究组还开源了轻量级的区块链实验仿真平台 BlockEmulator<sup>1</sup>，它支持研究者快速复现 BrokerChain 等分片协议，还允许研究者在该仿真平台上进行创新、二次开发新版本的分片协议与机制，并支持便捷的性能测试（包括交易吞吐量、交易上链时延、跨片交易比例、交易池的拥塞程度等指标）。
- BlockEmulator 已被应用于数十篇区块链论文，当做这些论文的实验工具。

BrokerChain 协议及其衍生研究（如 Broker2Earn、Fine-tuned Account Migration、Justitia）连续入选了计算机网络领域的顶会 INFOCOM

---

<sup>1</sup><https://www.blockemulator.com/>

## 1.2 BrokerChain (academic) Testnet

基于上述学术研究成果，位于香港的 BlockIn 团队联合黄华威教授研究组一起将“纸上谈兵”的理论转化为了真实运行的区块链系统。为了致敬原始协议，上线的区块链系统仍然取名 BrokerChain。而且，黄华威研究组继续担任 BrokerChain 区块链上线后的技术支持团队。

BlockInLab 团队深知搭建和运营公链生态的不易，故本次先以“测试网”来试水，命名为 **BrokerChain (academic) Testnet**，未来会上线 BrokerChain 主网。

那么，一个自然的问题：BrokerChain (academic) Testnet 有什么用？

回答这个问题，首先需要回答另一个相关的问题：BrokerChain 区块链有哪些参与者角色？

### 1.2.1 几类典型的参与者角色

如图 1.2 所示，BrokerChain 的参与者包含四类典型角色：Researcher（研究者），Miner（共识节点，俗称“矿工”），与 Developer（链上应用 dApps 的开发者），User（一般用户，dApps 的使用者）。

他们与 BrokerChain 的交互分别介绍如下。

### 1.2.2 BrokerChain (academic) 可以用来做什么？

In BrokerChain's ecosystem, anyone can become a researcher, miner, staker, market maker, or dApp developer.

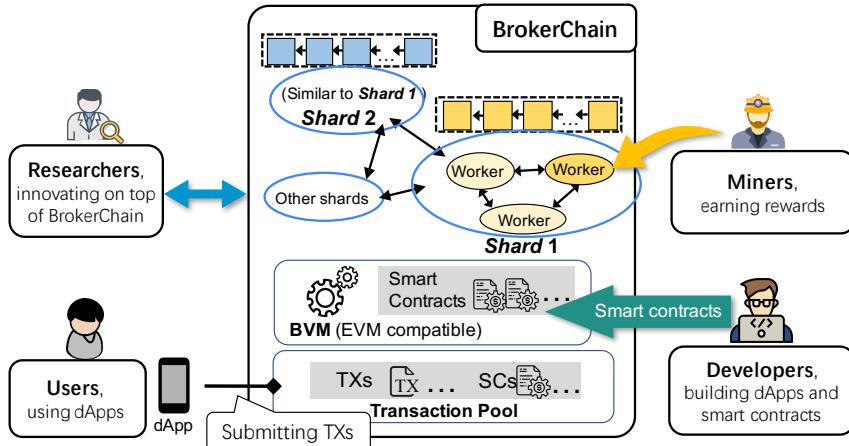


图 1.2: BrokerChain 区块链的几种参与者角色。值得注意的是，图中的 BVM (BrokerChain Virtual Machine) 是 BrokerChain 区块链的智能合约执行虚拟机。测试网 BrokerChain (academic) Testnet 暂且内嵌了以太坊的 EVM 当做智能合约的执行环境，因此兼容部署在以太坊上的智能合约。未来黄华威研究组计划推出自研的 BVM，但仍然兼容 Solidity 语言编写的智能合约。

BrokerChain (academic) Testnet 的定位是优先支持区块链的研究者在 BrokerChain 区块链上进行创新。当然，除了 Researcher 角色，它也可以支持图1.2 所示的其他几类角色，分别阐述如下。

- 1. Researchers:** 研究者可以参考第6章节展示的三个 dApps demo 进行应用层的创新，或者使用在 Github 项目的开源代码开展链底层的技术创新。Please find more explanation in Chapter 4.1.
- 2. Miners** can join the Testnet to earn mining rewards. Please refer to Chapter 4.2 to learn how to join the network as a consensus node.
- 3. Developers** can build dApps via deploying smart contracts to the embedded EVM. Please refer to Chapter 4.3 to learn how to de-

ploy/invoke smart contracts.

4. **Users:** 即一般用户，他们使用 dApps 可以发起转账交易、可以投资 BrokerChain 链上金融产品、还可以参与链上应用发起的活动等等。例如，用户在使用 BrokerChain Wallet 时，使用目的可以细分为提交转账交易、收款、质押 tokens 到 DeFi 协议、成为 Broker 角色 (i.e., market maker) 赚取流动性 staking 收益、领取“水龙头”tokens、参与空投 (airdrops) 等等。

### 1.3 BrokerChain 有什么特别的地方？

这是一个被经常问起的问题，为此研究团队总结了如下几个特性，来阐述 BrokerChain 分片区块链的特性与竞争力。

1. **技术原生：**链内部自带原生 DeFi 金融协议框架（详见章节6.1.2）、参数灵活调整、提供优先级金融服务、可在协议层定制化开发（风险可控）、支持可插拔监管（未来主网版本将会支持）。
2. **从开发者角度：**兼容以太坊，gas 费定制化设计；未来版本会嵌入智能合约漏洞实时检测功能，帮助开发者在上线智能合约前给出一个代码漏洞的评估方案；链的研发团队提供了典型的链上应用 demo 与示例代码（详见 Chapter 6），可显著降低开发者的开发难度。
3. **从参与共识的用户角度：**不需要拼算力即可加入网络参与共识，获取奖励；支持轻量级的用户设备参与共识；可将赚取的 token 投入链自带的 LP（Liquidity Providing, 流动性提供）池子赚取额外的“流动性质押”收益（收益稳定、风险极低，详见章节6.1介绍）。

4. **从 dApp 用户角度**: 交易 User Experience 体验好, 因为 BrokerChain 区块链可以保证用户提交的交易上链确认时延低。
5. **从研发团队角度**: 后续会持续对链进行升级迭代, 运用专业科研团队的力量与智力可快速解决新问题、提高链各方面的性能, 例如下一代 BrokerChain 将支持功能强大的自研虚拟机 BrokerVM (BVM), 支持链上 AI 与实时 social media 应用等等现有的链支持不了的特性。

## 1.4 BrokerChain (academic) Testnet 测试网

### 可能会遇到的风险

BrokerChain (academic) Testnet 可能会由于技术故障、或者代码触发了严重 bug 而重新启动。重新启动后, BrokerChain (academic) Testnet 的状态数据可能会被重置 (Reset 操作)。Reset 操作可能会导致部分账户的信息丢失。请各位测试网的用户注意相关风险。

## 第二章 BrokerChain 区块链技术原理

### 2.1 BrokerChain 分片协议的原理

BrokerChain 的分片协议 [5] 和相关共识机制 [1,2,8] 是其高效运行的核心，通过分片划分和多阶段共识实现高吞吐、低延迟的交易处理，同时保障系统的安全性和负载均衡。

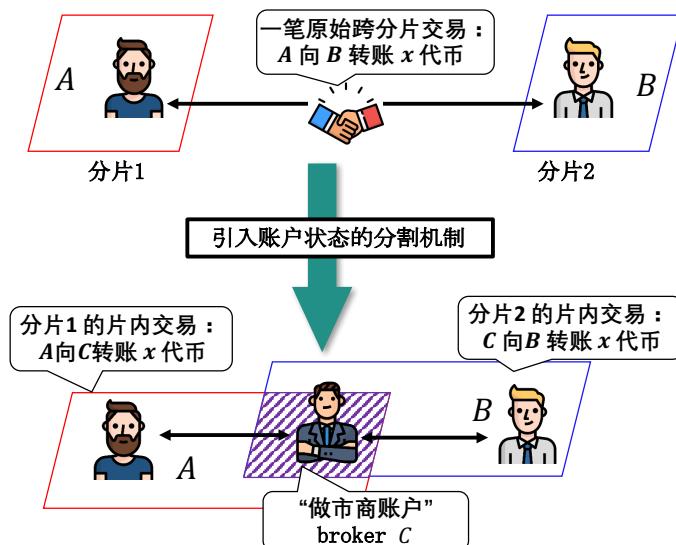


图 2.1: 基于“做市商账户 (broker)”机制的跨分片交易处理原理。

### 2.1.1 Broker 机制的原理

Broker 账户也就是“做市商账户”，运用状态分割技术，让每个分片中存在 Broker，当用户提交了一个跨分片交易，如图 2.1 所示，某分片的账户 A 向另一个分片的账户 B 发起一笔交易，那么这一笔跨分片交易就可以拆解为两笔片内交易，分别为账户 A 和同分片的 Broker 账户 C 进行一笔交易以及账户 B 和同分片的 Broker 账户 C 进行一笔交易，当交易完成后，实现跨分片交易处理，以减少跨分片交易的数量。

### 2.1.2 系统架构设计

如图 2.2 所示，BrokerChain 系统的分片架构由两类功能各异的分片组成：“共识分片”（miner shard, M-Shard）和“划分分片”（partition shard, P-Shard）。M-Shard 负责交易的收集、验证和区块生成，采用改进的 PBFT 共识算法来确保分片内的交易一致性。这种设计使得多个 M-Shard 可以并行处理交易，显著提高了系统的整体吞吐量。每个 M-Shard 都维护着部分账户状态，只需存储与其分配账户相关的数据，这大大降低了单个节点的存储负担。

P-Shard 作为系统的核心协调者，承担着全局状态管理的重任。它持续监控来自各个 M-Shard 的交易数据，构建并维护一个动态的状态图。这个状态图以账户为节点，以交易频率为边权重，准确刻画了系统中账户间的交互模式。在每个运行周期（Epoch）结束时，P-Shard 会运用先进的图划分算法（如 Metis）对状态图进行分析，计算出最优的账户分片分配方案。这一过程充分考虑了负载均衡和跨分片交易最小化两个关键指标。

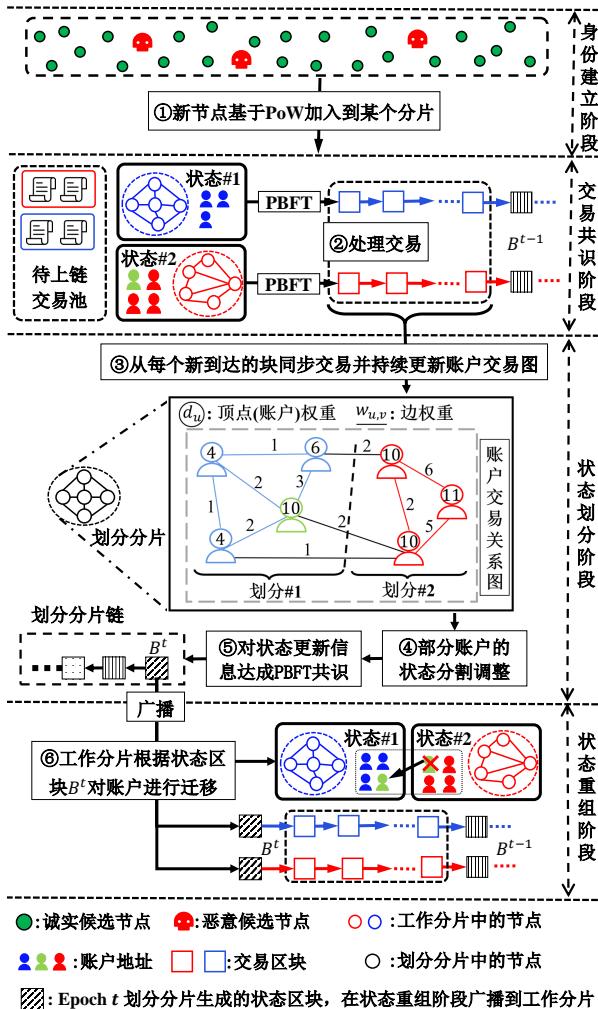


图 2.2: BrokerChain 协议 [3,5] 概览。其中, 第 6 步中对账户进行迁移的方案请参考研究论文 [2]。

### 2.1.3 片内共识机制

BrokerChain 系统的共识设计采用周期性的运行模式, 每个 Epoch 都经历四个严谨的阶段。在交易区块共识阶段, 各个 M-Shard 独立工作, 通

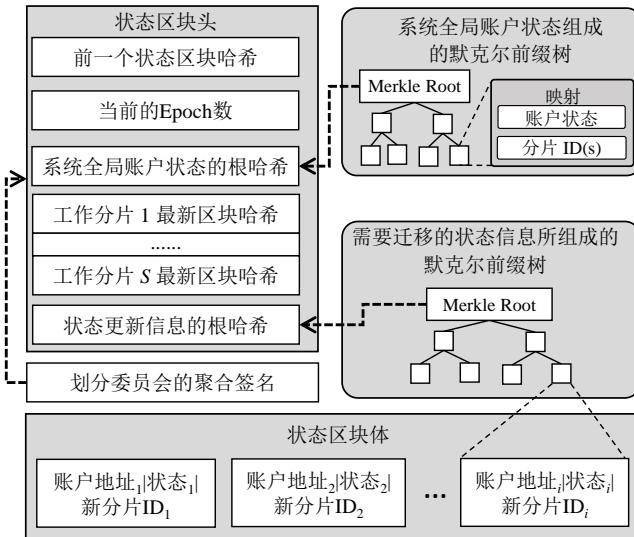


图 2.3: 状态区块 (state block) 概览。

过 PBFT 共识机制对收集到的交易达成一致，生成经过验证的交易区块。这些区块随后被传送至 P-Shard，为状态图的更新提供数据支持。状态图划分阶段是系统的智能核心，P-Shard 通过分析交易模式，动态调整账户分布，确保系统始终保持在最优的运行状态。

状态区块共识阶段采用同样的 PBFT 机制，确保所有节点对新的账户分配方案达成一致。这一阶段生成的“状态区块”（state block，如图2.3所示）记录了完整的账户分片映射关系，是系统状态演化的关键见证。最后的状态重配置阶段，各个 M-Shard 根据最新的状态区块调整本地存储的账户数据，为下一个 Epoch 的运行做好准备。这种周期性的状态更新机制，使系统能够动态适应不断变化的交易模式。

安全性方面，BrokerChain 采用了多层次防护措施。P-Shard 的节点选择采用随机分配策略，防止恶意节点集中攻击。PBFT 共识算法确保每个分

片都能容忍不超过 1/3 的拜占庭节点。定期的分片重组机制 (Cuckoo Rule) 进一步增强了系统抵抗女巫攻击的能力。这些安全设计使得 BrokerChain 在提升性能的同时，仍然保持了区块链应有的安全特性。

#### 2.1.4 BrokerChain 账户分割机制

在区块链分片系统中，账户管理方式直接影响着系统的性能和可扩展性。BrokerChain 提出的账户分割 (Account Segmentation) 机制，从根本上改变了传统分片区块链的账户管理模式，为解决负载不均衡和跨分片交易问题提供了创新性解决方案。

早期的分片区块链系统，如 Monoxide，采用基于账户地址前缀的静态分片分配策略。这种方式虽然实现简单，但存在明显缺陷：每个账户被固定分配到一个特定分片，无法根据实际交易模式动态调整。这种刚性分配导致两个主要问题：首先，高频交易账户会使其所在分片成为处理瓶颈，形成所谓的“热分片”；其次，涉及多个分片的账户间交易必须通过复杂的跨分片协议来完成，显著增加了系统延迟。

BrokerChain 的账户分割机制允许单个账户的状态被智能地分割存储在多个分片中，同时保持账户地址的唯一性。这一创新类似于银行账户中的“子账户”概念，但完全由协议层自动管理，对用户透明。具体实现上，系统会为分割账户在每个相关分片中维护独立的状态记录，包括余额、交易计数等关键信息，但所有子状态共享同一个账户地址。

账户分割的实现依赖于精心设计的数据结构，即新设计的分片状态树 (mSST，如图2.4所示)。它在传统账户状态信息基础上增加了“存储映射”向量。这个向量采用位图形式，精确记录账户状态在各个分片的分布情况。每个分片只维护本地存储的账户子状态，包括：

- 分片专属余额 (value)

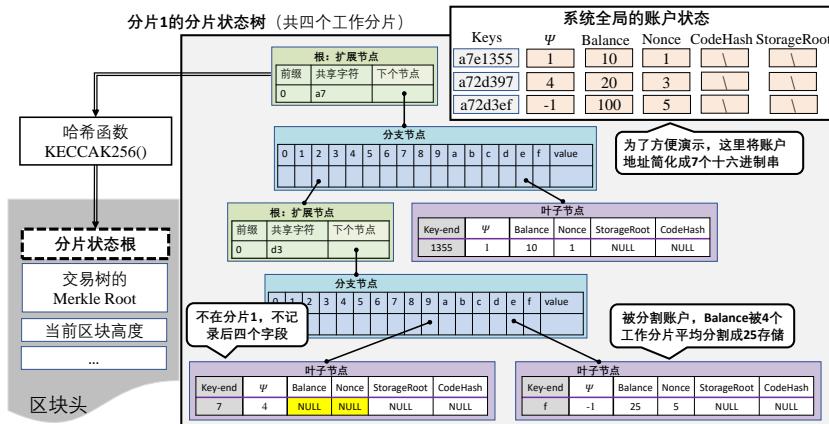


图 2.4: 分片状态树 mSST (modified sharding state tree) 原理示意图。

- 本地交易计数器 (nonce)
- 合约代码哈希 (code)

这种设计确保分片间状态相互独立, 修改一个分片中的子状态不会影响其他分片, 大大简化了并发控制。

在性能方面, 账户分割通过动态调整账户状态在分片间的分布, 从根本上解决了传统分片系统的两大瓶颈问题。首先, 该机制大幅降低了跨分片交易比例, 实验数据 [5] 显示, 相比 Monoxide 高达 98.6% 的跨分片交易率, BrokerChain 可将其控制在 7.4% 的极低水平。其次, 通过将高频交易账户智能拆分到多个分片, 有效消除了热分片现象, 使系统吞吐量提升近 10 倍, 交易确认延迟降低至 300 毫秒以内。这种性能优化并非以牺牲用户体验为代价, 用户仍通过单一地址操作系统, 完全无需感知后台复杂的分割逻辑。

安全性设计上, 账户分割机制融合了多重防护措施。状态转移采用严谨的两阶段提交协议, 确保分割操作的原子性; 设置合理的交易锁定周期

防范“双重支付攻击”(double-spending attacks)；引入非对称验证机制防止余额重复计算；定期审计机制保障各分片子状态的一致性。这些安全设计使得账户分割在提升性能的同时，仍能维持区块链系统应有的安全特性。

在实际应用场景中，账户分割展现出极强的适应性。在数字货币交易所等高频交易场景中，热门交易对的账户会自动分散到多个分片，避免形成处理瓶颈；在支付系统中，商户账户可以根据客户分布模式智能优化分片位置。这种动态调整能力使 BrokerChain 能够适应各种复杂的商业应用场景，为 DeFi、供应链金融等需要高吞吐量的区块链应用提供了理想的基础架构。

BrokerChain 的账户分割机制代表了区块链账户管理的范式转变，从“账户固定归属”演进到“状态智能分布”，为分片区块链的可扩展性提供了全新思路。这种设计不仅解决了当前分片区块链的技术瓶颈，也为未来更复杂的分布式账本应用奠定了架构基础。

### 2.1.5 跨分片交易机制

BrokerChain 跨分片机制，来自于 INFOCOM2022 收录的论文“BrokerChain: A Cross-Shard Blockchain Protocol for Account/Balance-based State Sharding”。该论文针对目前分片区块链系统中存在大量跨分配交易的问题提出了 BrokerChain 跨分片交易机制，即，将选定为 Broker 的账户的状态进行分割，使其存在于每个分片之中，当分片间出现跨分片交易的时候，就可以使用 Broker 来进行处理。

图 2.5 描述的是一笔由 Broker 服务的跨分片交易的执行流程时序图，具体流程描述如下：

- **Op1** : Sender A 发送原始交易消息  $\theta_{raw}$  给 Broker C。

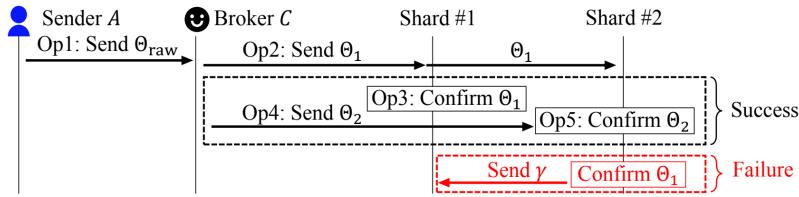


图 2.5: 交易执行时序图。

- **Op2**: BrokerC 收到原始消息  $\theta_{raw}$  后，分别给原始交易发送方账户和接收方账户所在分片（图中 Shard1、Shard2）发送  $\theta_1$ 。
- **Op3**: Shard1 节点接收到  $\theta_1$  后，判断消息合法性，构建 A 和 C 之间的交易 (Tx1)，如果交易上链，则向 Broker 发送 Confirm  $\theta_1$  消息。
- **Op4**: Broker 接收到 Confirm  $\theta_1$  消息，验证消息的有效性后，向 Shard2 发送  $\theta_2$ 。
- **Op5**: Shard2 节点接收到  $\theta_2$  后，判断消息合法性，构建 C 和 B 之间的交易 (Tx2)，如果交易上链，则向 Broker 发送 Confirm  $\theta_2$  消息完成跨分片交易。

图 2.5 示例的系统中存在两种类型的节点：Sender 和 Broker。两种角色在系统运行时承担的任务是不同的。

**Sender**: 目前 Sender 为**区块链中共识节点担任的角色** (如 PBFT 中主节点，负责接收客户端发来的交易和给“从节点”(即 **follower** 节点)和 broker 客户端发消息)。

- 负责检测接收的交易是否为**跨分片交易**，如果是跨分片交易，则生成  $\theta_{raw}$ ，并发送给 Broker。

- 负责处理来自 Broker 的  $\theta_1$ ，主要步骤为验证消息、生成 Tx1 片内交易、添加 Tx1 到交易池。
- 负责处理来自 Broker 的  $\theta_2$ ，主要步骤为验证消息、生成 Tx2 片内交易、添加 Tx2 到交易池。
- 当新的区块打包上链后，负责筛选 BrokerTx，并分别生成 Confirm  $\theta_1$  和 Confirm  $\theta_2$  发送给 Broker 客户端。

**Broker**: Broker 作为独立客户端，专门负责跨分片交易请求。在分片区块链网络中，Broker 账户将会被分割到各个分片中，也就是说 Broker 将拥有和分片数量对应的 accounts (记做 BrokerAccounts)，且 BrokerAccounts 不参与区块链网络的状态迁移。举个例子，假如给定一个跨分片交易: CTX:  $\langle A \rightarrow B \rangle$ ，其中，账户 A 位于分片 shard#1，账户 B 位于分片 shard#2。当分片区块链检测到该跨分片交易 CTX 后，经过一系列信息交换，**该跨分片交易 CTX 被转换为两个片内子交易**，即 Tx1:  $\langle A \rightarrow \text{BrokerAccount1} \rangle$  和 Tx2:  $\langle \text{BrokerAccount2} \rightarrow B \rangle$  (其中，BrokerAccount1 位于分片 shard#1, BrokerAccount2 位于分片 shard#2)。

在跨分片交易处理方面，BrokerChain 提出了创新的“做市商账户”解决方案。这些特殊账户的状态被**有意分割存储在多个分片中**，充当跨分片交易的桥梁。通过将原本需要在多个分片间协调的复杂交易，拆分为两个可以在单个分片内完成的简单交易，系统大幅降低了跨分片交易的处理难度和延迟。配合精心设计的交易锁定机制，既**保证了交易的原子性**，又有效**防范了双重支付等恶意行为**。

## 2.2 BrokerChain 网络可扩展性

Question: 共识节点接入 BrokerChain 网络时，会发生什么事？

### 2.2.1 共识网络中两种分片类型

为提升 BrokerChain 共识网络的可扩展性与任务分工效率，BrokerChain 网络将包含两类分片：

- **Junior Shard** (普通分片，我们形象地称之为普通桌)：主要参与交易共识，适合中低资源配置的共识节点参与；
- **Senior Shard** (长老分片，我们形象地称之为长老桌)：负责交易共识、跨片协调、分片重组等任务，要求加入的节点具备更强的性能与超稳定的网络连接质量。

当加入共识网络时，用户将会面临以下两个选项，可加入任何一种分片（这里形象地称之为“购入一个席位”）：

- **长老桌席位**：席位费 1 万-100 万 BKC，费用将会根据网络参数动态地浮动调整；优点：单位时间内的收益率高，可享受 BrokerChain 链上某些 DeFi 产品的“优先质押”金融服务；缺点：一旦退出长老桌，就失去了席位费。
- **普通桌席位**：席位费固定为 2 BKC；缺点：单位时间内的收益率低，不可享受 BrokerChain 链上某些 DeFi 产品的“优先质押”金融服务；优点：自由度高，可随时加入或者退出一个普通桌，损失的席位费代价较低。

## 2.2.2 节点加入共识网络

随着加入网络的共识节点的增加，BrokerChain 分片数量会自动延伸扩展。这样可以保证共识网络几乎可以扩展到很大的量级。但当网络中共识节点规模过大时，会导致产生极高比例的“跨分片交易（Cross-shard Transactions, CTXs）”。此时，越加突显出 Broker 账户角色与 BrokerChain 原始跨分片协议 [3] 的重要性，因为来自于 broker 账户质押的 tokens 可以帮助“消灭”广泛存在于分片区块链的跨分片交易。

需要提示一点：由于分片区块链在“共识节点组成新的分片”这个特殊的阶段，新节点加入共识网络时，可能需要稍微等待一小段时间，跟别的其他共识节点“凑成一桌”后，才会真正地进入某一个“长老桌”或者“普通桌”。这是因为，新节点需要等待满足了协议层预先设置的分片内共识节点的最小数量要求时，才可以开始参与共识并获取出块奖励。

## 2.2.3 分片节点的组织规则

在分片区块链横向扩展形成新的分片时（无论是“长老桌”还是“普通桌”），协议会从 `waiting list` 中随机选取一定数量的候选共识节点组成一个新的分片。During all those consensus node candidates of a given new shard that will be formed, one will be selected randomly as the leader for this new shard's PBFT consensus.

# 第三章 通证经济模型与激励机制

本章介绍 BrokerChain 区块链的通证发行机制以及对相关角色的激励机制设计。

## 3.1 通证机制设计

BrokerChain 区块链的通证总量设定为 **12 亿枚 BKC**，采用“初始阶段预分配（25%）+ 长期出块奖励（75%）”的混合发行模式：初始阶段预分配方案用于激励早期参与者、保障网络安全并推动生态发展；长期出块奖励方案激励参与共识的节点。BrokerChain 区块链的通证沿用以太坊的 `wei` 为最小计量单位，即  $1 \text{ BKC} = 10^{18} \text{ wei}$ 。

### 3.1.1 初始通证分配

协议在初始阶段通过 genesis block 预分配总计 3 亿枚通证（占总数量 25%），具体用途如下：

- **1 亿枚**：预存入面向开发者的“水龙头账户”，支持面向开发者社区的开发测试；
- **0.5 亿枚**：用于激励活跃的 BrokerChain 社区用户与贡献者；
- **1.5 亿枚**：用于激励应用生态。这部分代币发放后，将会在每一个目标账户锁定 3 年。

### 3.1.2 其余通证的长期释放

其余通证将通过共识节点参与共识获得出块奖励而发行。占据总量 75% 的通证将通过“出块奖励”逐年释放，每四年释放量减半。每次区块完成共识后共识节点会收到奖励，奖励规则请参考公式(3.6)。

## 3.2 长期出块奖励机制

### 3.2.1 片内共识奖励规则

首先，每个分片内部执行 PBFT 协议（未来会升级片内共识协议），所以每个分片内的节点有些会成为 `leader`，有些会成为 `follower` 节点。每个分片内每轮共识只有一个节点会成为 `leader`，其他节点属于 `follower`。

那么，这些不同角色的节点的奖励分配规则由协议层执行。而且，协议可动态地调整每个分片内部对共识节点的出块奖励（即 `block reward`）的奖励值，并遵循以下设计原则：

- **奖励来源：**每个新产生区块的出块奖励 `block reward` 自两部分：i) `base reward`, ii) 该 `block` 内部包含的交易的手续费之和。
- **通胀控制：** `base reward` 每四年减半，确保 token 发行总量可控。
- **行为导向：**激励真实参与共识过程的关键角色：即 `leader` 节点与积极参与投票的 `follower` 节点。
- **长期激励：**每个 `follower` 的收益与其持续在线时长正相关。
- **响应效率：**鼓励 `follower` 节点快速响应 `leader` 的提议。

### 3.2.2 跨片共识的奖励规则

在 BrokerChain 中，一笔“跨分片交易 (CTX)”会被拆分为两个“片内子交易 (Intra-shard Transaction, ITX)”，分别在发送方账户所在分片和接收方账户所在分片执行。每个子交易分别由对应分片内的某个 `leader` 共识节点打包上链。因此，**跨分片交易的奖励也需要在参与其处理的两个分片之间合理分配**，从而激励对应分片的共识节点积极推进跨分片交易的共识上链。

#### CTX 的手续费均分规则

对于一笔合法的跨分片交易  $\text{CTX}: \langle A \rightarrow B, \text{fee}_{\text{CTX}} \rangle$ ，其中， **$\text{fee}_{\text{CTX}}$  是用户支付的交易手续费**； $A$  账户在分片 shard#1， $B$  账户在分片 shard#2。BrokerChain 协议会将 CTX 拆分为两笔片内交易，即：Tx1 (由分片 shard#1 执行，其手续费分成为  $\text{fee}_{\text{Tx1}}$ ) 和 Tx2 (由分片 shard#2 执行，其手续费分成为  $\text{fee}_{\text{Tx2}}$ )。该交易的手续费  $\text{fee}_{\text{CTX}}$  在 BrokerChain (academic) Testnet 中被等分，并划归为两笔子交易 Tx1 与 Tx2 的手续费，作为对各自打包它们的两个分片的区块奖励。Thus, we have

$$\text{fee}_{\text{Tx1}} = \text{fee}_{\text{Tx2}} = \frac{1}{2} \cdot \text{fee}_{\text{CTX}}, \forall \text{CTX } (\text{CTX} \rightarrow \text{Tx1} \& \text{Tx2}). \quad (3.1)$$

每笔子交易 (Tx1 与 Tx2) 分别由其所在分片的共识节点负责打包。因此，参与这些交易共识过程的 `leader` 与 `followers` 的手续费奖励将计入其打包区块的区块奖励，区块奖励分配规则与普通片内交易一致。

### 3.3 共识节点激励函数设计

为实现公平透明的激励机制并控制代币通胀，区块链的设计者需要设计一个基于共识行为和在线历史的即时奖励函数。每个分片  $s$  ( $s \in S$ ,  $S$  是当前网络中所有活跃分片的集合) 在高度  $h$  ( $h \in \mathbb{N}$ ,  $\mathbb{N}$  表示自然数集合) 产出区块的奖励包括出块奖励与交易手续费，由该区块的 leader 和部分积极参与共识的 followers 共享。

为表述简洁，记分片  $s$  在高度  $h$  上产出的区块为  $\text{block}(s, h)$ ，适用于任意  $s \in S, h \in \mathbb{N}$ 。

#### 3.3.1 区块总奖励定义

每个区块  $(s, h)$  的总奖励由“出块奖励”与“交易手续费”两部分组成，奖励的计算方式如下：

$$R(s, h) = B(s, h, \beta_{s,h}, W(S)) + F(s, h), \quad \forall (s, h). \quad (3.2)$$

其中：

- $s \in S$ ：表示当前 BrokerChain 网络中的一个活跃分片；
- $h \in \mathbb{N}$ ：表示出块高度；
- $\beta_{s,h}$ ： $\text{block}(s, h)$  的出块时间戳；
- $W(S)$ ：当前网络中所有分片的权重之和，定义见公式 (3.4)；
- $B(s, h, \beta_{s,h}, W(S))$ ： $\text{block}(s, h)$  的基础出块奖励，具体计算见公式 (3.5)；
- $F(s, h)$ ： $\text{block}(s, h)$  中所有交易产生的手续费总和；

-  $R(s, h)$  : block  $(s, h)$  在当前共识轮可获得的 token 总奖励 (含出块奖励与手续费)。

### 3.3.2 分片类型与权重机制

如章节2.2.1所述，为了提升网络的可扩展性与适应不同性能配置的共识节点，BrokerChain 网络将所有分片划分为两类：

- **Senior Shard**: 负责交易共识、跨片协调、分片重组等任务，要求加入的共识节点具备更强的性能与稳定的网络质量；
- **Junior Shard**: 负责交易共识，适合中低资源配置的节点参与。

每个分片  $s \in S$  被赋予一个静态或动态设定的权重  $w_s \in \mathbb{R}^+$ ，表示其基础出块奖励的计算权重。其中，**Senior Shard** 的权重为 **Junior Shard** 的  $\alpha$  倍 ( $\alpha \in \mathbb{R}^+, \alpha \geq 1$ )。

$$w_s = \begin{cases} \alpha, & \text{if } s \text{ is a Senior Shard} \\ 1, & \text{if } s \text{ is a Junior Shard} \end{cases}, \quad \forall s \in S. \quad (3.3)$$

以上公式定义了每个分片的权重，反映了不同类型分片在奖励分配中的差异。

记所有分片的总权重用符号  $W(S) \in \mathbb{R}^+p$  表示，则它的计算方式为：

$$W(S) = \sum_{s \in S} w_s. \quad (3.4)$$

该权重总和将用于计算“基础出块奖励”。

### 3.3.3 基础出块奖励的时间减半机制

为控制代币总量增长，BrokerChain 区块链采用逐周期减半的出块奖励机制。在每轮共识中，区块  $(s, h)$  的基础出块奖励计算如下：

$$B(s, h, \beta_{s,h}, W(S)) = B_0 \cdot 2^{-\lfloor \frac{\beta_{s,h} - t_0}{T} \rfloor} \cdot \frac{w_s}{W(S)}, \quad \forall (s, h). \quad (3.5)$$

其中：

- $B_0$ ：BrokerChain 区块链在创世时设定的基础出块奖励的速度参数，该参数用于控制整个区块链中 token minting 的新增速度；
- $t_0$ ：BrokerChain 创世区块的时间戳；
- $T$ ：出块奖励减半周期，单位为毫秒；
- $\lfloor \cdot \rfloor$ ：向下取整函数，控制按周期逐级减半；
- $w_s$ ：分片  $s$  的出块权重，取决于其类型（Senior Shard 或 Junior Shard）；
- $W(S)$ ：当前所有活跃分片的权重之和，见公式 (3.4)。

该公式确保基础出块奖励随着时间推移逐步递减，并依据分片类型及职责进行加权分配，从而实现通胀控制与激励公平性的平衡。

### 3.3.4 奖励分配函数

区块总奖励将按节点在共识过程中的贡献进行分配。我们定义  $\text{Reward}(i, s, h)$  为共识节点  $i$  ( $\in I$ ) 在产出 block  $(s, h)$  时获得的实际奖励，计算如下：

$$\text{Reward}(i, s, h) = \frac{S(i, s, h)}{\sum_{i \in I} S(i, s, h)} \cdot R(s, h), \quad \forall (s, h), \forall i \in I. \quad (3.6)$$

其中：

- $I$  是 BrokerChain 网络中所有共识节点的集合;
- $S(i, s, h)$  : 节点  $i$  ( $\in I$ ) 在产出 block  $(s, h)$  时的角色贡献得分, 详见公式 (3.7);

该公式表示节点  $i$  ( $\in I$ ) 在产出 block  $(s, h)$  时实际获得的奖励与其得分在本轮的占比成正比。

### 分片内不同共识节点的得分函数

根据共识节点在共识过程中担任的角色 (leader 或 follower), 它们将会获得不同的得分。具体来讲, 得分函数设计如下:  $\forall i \in I, \forall (s, h)$ ,

$$S(i, s, h) = \begin{cases} 2 \times \phi(t_{i,s,h}), & \text{若 } i \text{ 是 block } (s, h) \text{ 的出块节点 (i.e., leader) ;} \\ \phi(t_{i,s,h}), & \text{若 } i \in \text{TopW}(s, h); \\ 0, & \text{其它.} \end{cases} \quad (3.7)$$

其中:

- $\text{TopW}(s, h)$  : 为 block  $(s, h)$  签名的时间戳最早的前  $w$  ( $\geq 1$ ) 个 followers 节点的集合。其中, 我们定义  $w$  ( $\in \mathbb{N}$ ) 为一个合法区块 (即, block  $(s, h)$ ) 所需要的最小签名需求数。当使用 PBFT 协议时,  $w = 2f + 1$ ,  $f (\in \mathbb{N})$  是 PBFT 共识协议正常执行时共识节点中“拜占庭节点 (Byzantine nodes)” 的最大数量。

- $\phi(t_{i,s,h})$  : 为节点  $i$  为产出 block  $(s, h)$  的在线时长得分函数, 详见公式 (3.8)。

对于 block  $(s, h)$  的出块者 (即 leader 节点) 来说, 该公式可以衡量响应该 leader 节点的前  $w$  名 followers 的激励优先级。

## 在线时长得分函数

我们首先定义  $\phi(t_{i,s,h})$  为节点  $i$  的上线时长  $t_{i,s,h}$  对应的奖励档位, 它的取值范围是  $\{0.1, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ , 并按照如下公式来计算:

$$\phi(t_{i,s,h}) = \begin{cases} k, & \text{若 } \Delta t_i \in [2^{k-1}H, 2^kH), \quad 1 \leq k \leq 8; \\ 9, & \text{若 } \Delta t_i \geq 256 \times H; \\ 0.1, & \text{其它情况.} \end{cases} \quad \forall i \in I, \forall (s, h). \quad (3.8)$$

其中:

-  $t_{i,s,h} = \beta_{s,h} - \hat{t}_i$  : 节点  $i$  ( $\in I$ ) 相对于 block  $(s, h)$  的在线时长, 时间单位是毫秒 (ms),  $\beta_{s,h}$  是 block  $(s, h)$  的出块时间戳,  $\hat{t}_i$  是节点最近一次加入 BrokerChain 区块链的时间戳。

-  $H = 3600000$  : 一小时的毫秒数 ( $3600 \times 1000 = 3600000$ )。

一个节点持续在线时长越久, 它就越可能落入高档位, 从而获得更高得分。

## 3.4 可能会遭受的攻击

### 3.4.1 Attacks of Launching A large Number of Miner Clients by a Single Person

由于 BrokerChain 的共识协议属于“低算力资源需求 (low-CPU hungry)”类型, 所以每台普通电脑可以运行多个 miner clients, 每个 client 使用一个单独的账户来参与共识网络挖矿, 以期获得尽量多的共识奖励。

严格来讲，这种行为不是一种攻击，因为它是设计的共识协议规则允许的一种情形。但是，这种行为可能会造成一种后果：如果有人投入很多资源，运行并掌控了大量的 miner clients，那么他/她就可以通过某种方式同时启动大量的 miner clients，有一定概率可以在新生成的分片中占据绝对主导的共识节点份额（例如，占据了一个分片内超过三分之二的份额）。那么，这种行为对于分片区块链是一种潜在的威胁。

那么如何抵御这种攻击呢？当尽量多的 Miner clients 加入共识网络时，这种攻击成功的概率就会降低。这是因为，当新分片形成时，分片节点是从一个候选名单（waiting list）中挑选出来一部分来组成一个分片。因此，只要 waiting list 有足够的 miner clients 在等待加入某个新分片，这种攻击成功的可能性就越低。

### 3.4.2 Attacks of Keeping Randomizing to be Leaders

在内测的早期阶段，有的朋友会发现一个问题：成为 leader 角色，是不是能领取更多收益？有没有什么方法可以增大成为 leader 的几率呢？

BrokerChain 测试网早期的共识奖励机制设计会让 leader 的挖矿收益比 follower 角色的收益高几倍。事实上，在内测开始不久就已经有人利用这个漏洞，使用脚本发起了这个新型攻击：持续地关闭并重开大量的 miner 客户端，直到随机成为 leader 角色。

这个攻击会造成共识网络的动荡。例如，本来 32 个人“一桌打掼蛋”（即，参与同一个分片内的共识）很顺利，突然有一拨人离开了，结果祸害了“同一桌”的其他玩家。为了堵上这个漏洞，我们技术团队紧急讨论出来了一个**异构共识分片**的框架，即“长老桌”与“普通桌”共存的架构。

## 第四章 使用手册

根据图1.2 所示的 BrokerChain (academic) 的几类参与者角色，本章分别整理了他们对应的使用指南。

### 4.1 Researcher：基于 BrokerChain 进行链上应用生态的创新

Researchers 可以“魔改”开源的代码，只要能保证修改后的客户端可以与 BrokerChain Testnet 兼容即可。例如，研究者可以继续扩展共识节点的功能或者结合构建在链上 dApp 做跨层创新 (cross-layer innovation)。比如，研究者可以在 BrokerChain (academic) Testnet 研究“三明治攻击”或者“交易优先级排序”相关的研究课题。值得注意的是，如果研究者们想进一步对 BrokerChain (academic) 进行共识协议、跨分片交易流程优化、状态存储优化等链底层的功能进行创新，我们推荐您使用黄华威教授研究组开源的另一个相关的项目 BlockEmulator<sup>1</sup> [4]。

另外，研究团队鼓励 researchers 使用第 6 章展示的 3 个代表性 dApps demo 示例，进行二次开发并开展应用层的创新，读者也可以尝试基于这些 demo 构建新的 dApps。

---

<sup>1</sup><https://www.blockemulator.com/>

## 4.2 Miner：运行共识节点

Question: Miner 如何启动运行一个共识节点?

Miner 共识节点可能运行在不同的操作系统上。本节以 Windows 和 MacOS 系统为例来展示如何运行一个共识节点。此外，Github 项目页面也提供了 Linux 的预编译可执行文件，以及详细的操作流程指引，请读者自行探索。

### 4.2.1 安装 BrokerChain Miner 客户端

根据您的操作系统，从 BrokerChain Github 的 Release 页面<sup>1</sup> 下载对应的 BrokerChain 压缩包。下面以 Windows 操作系统为例，将 BrokerChain 压缩包解压到本地 BrokerChain 文件夹中。如图 4.1 所示，brokerchain.exe 是 BrokerChain 的主程序，html 文件夹包含了 BrokerChain 的钱包网页。

名称	修改日期	类型	大小
html	2025/5/23 11:02	文件夹	
brokerchain.exe	2025/5/24 10:17	应用程序	56,906 KB

图 4.1: 解压后的 BrokerChain (academic) miner 客户端程序。

### 4.2.2 在 Windows 系统运行 BrokerChain (academic)

双击 brokerchain.exe 运行 BrokerChain (academic)，进入 BrokerChain 欢迎页，如图 4.2 所示，您可以看到如下选项：

1. Join BrokerChain as a consensus node & Open the wallet.
2. Open a wallet (using your private key).

<sup>1</sup><https://github.com/HuangLab-SYSU/brokerchain-academic/releases/>

3. Query an account and its balance if given an address.
4. Transfer tokens to another account.
5. Claim BKC academic tokens through faucets.



图 4.2: 启动运行 BrokerChain 的共识节点，可看到 5 个选项。输入 1，选择成为共识节点并生成一个新账户的私钥。

### Choice 1: Generate the private key for a new account

如图 4.2 所示，再输入 1 后回车，配置共识节点所使用的私钥。可以看到如下新的选项：

1. Generate a pair of (public/private) keys for a new account (为一个新账户生成新的私钥).
2. Use the private key of an existing account (使用已有账户的私钥).

这里我们假设您是第一次加入 BrokerChain network，那么需要输入 1，系统则会为您生成一个新账户的公私钥对 (The client executable program will create a new account's (public/private) keys for you)。

如图 4.3 所示，系统提示我们输入私钥的保存位置。如果输入 pk22 (注意这里没有添加文件后缀)，系统将会把新生成的私钥 (例如 7691\*\*\*782)

```

MacBook-Pro-6:brokerchain_academic_macos hwhuang$ ./brokerchain_academic_macos
[-----] [-----] [-----] [-----] [-----] [-----] (academic)
[-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] [-----] (academic)

BrokerChain仅供学术交流使用，用户不得使用BrokerChain从事任何非法活动。  

用户使用BrokerChain所产生的任何直接或间接后果，均由BrokerChain创始团队无关。  

BrokerChain创始团队保留随时修改、更新或终止BrokerChain的权利，且无需事先通知用户。  

用户在使用BrokerChain时，应自行承担风险，并同意放弃对创始团队的任何索赔权利。  

本免责声明接受中华人民共和国法律管辖，并按照其解释。
Welcome. Please enter an option:  

1: Join BrokerChain as a consensus node && Open the wallet.  

2: Open a wallet.  

3: Query an account and its balance if given an address.  

4: Transfer tokens to another account.  

5: Claim BKC tokens through faucets.  

1  

Please enter an option:  

1: Generate a pair of (public/private) keys for a new account  

2: Use the private key of an existing account  

1  

Please enter the filename to save the generated private key:  

pk22  

Private key generated: 7691*****  

The private key is successfully saved to file: pk22 This private key is saved on your PC.
Please enter an option:  

S: Join a Senior Shard  

Otherweise: Join a Junior Shard Choose to join a Junior Shard.  

3  

Your account address is: [74d58ed2f831154be92786efca524b1535296ee8]  

The browser wallet URL is: [http://127.0.0.1:36588] Copy this URL to your browser, and you will see your wallet in a webpage.  

Start trying to join BrokerChain network... Failure to join, due to the stake threshold.  

{"error": "Join failed. Your balance is less than 2 BKC."}  

[poS failed]. Your account balance is not enough to join junior shard. Program will exit after 10 seconds.  

{"error": "Join failed. Your balance is less than 2 BKC."}
MacBook-Pro-6:brokerchain_academic_macos hwhuang$ 

```

图 4.3: Choose to generate a new private key and save it on your PC initially. However, a new account fails to join the junior shard network due to the proposed **stake threshold** design, in which each new account must hold at least 2 BKC before joining the mining network of a junior shard. This **stake threshold** aims to prevent Sybil attackers from generating a huge number of new accounts without much cost.

保存到可执行文件当前所在文件夹的 `pk22` 文件中，并且在私钥生成时该文件只存储在您的本地磁盘而不会发送到网络中。需要说明的是，此 BrokerChain (academic) Testnet 只是一个学术测试版，您的私钥是明文显示与存储。请您妥善保管好您的私钥，不要和任何人分享，否则您账户的资产将处于风险之中。若您丢了私钥，则无法找回账户中的资产。

在生成的私钥信息下方，可执行程序还自动展示了您的私钥对应的“账户地址 (account address)”，例如 74d58ed2f831154be92786efca524b1535296ee8。

此账户地址与刚刚生成的私钥是绑定在一起的。另外，还可以看到输出信息中还显示了 `browser wallet URL`。您可以把这个 URL 粘贴复制到浏览器，就可以看到您这个账户对应的钱包页面了。

如图4.3所示，在生成私钥之后，`miner` 客户端程序还会自动尝试将此账户加入共识网络。但是，会提示失败信息：`{"error": "Join failed. Your balance is less than 2 BKC."}`。这是因为我们为每一个新账户设计了一个加入共识网络的门槛（叫做 `stake threshold`），目的是为了防止恶意攻击者无成本地生成大量新账户加入网络，这种行为本质上是对共识网络发起了“女巫攻击（Sybil attack）”。所以，我们在这里强制让每个新账户必须“支付”2个BKC之后，才有资格加入 `Junior Shard` 的分片共识网络。

那么，新账户如何获取这2个BKC的“启动资金”呢？您可以去图4.2所示的第五选项（即，5: `Claim BKC academic tokens through faucets.`）探索一下，从测试网预设的一个“水龙头（Faucets）”账户中领取少量BKC。当然，您也可以找已经持有BKC的朋友借2个BKC。

#### Choice 2: Use the private key of an existing account

如图4.4所示，若您想要使用一个已有账户的私钥，请输入2。如果您有一个账户余额足够的账户（例如图中所示的私钥文件 `pk1`），那么可以有两个选择：继续输入S，加入一个Senior Shard；或者输入其他任意字符，加入一个Junior Shard。

如果您输入S，并且这个账户的余额足够的话，客户端就会开始尝试加入一个“长老桌”，如图4.5所示。

```
MacBook-Pro-6:brokerchain_academic_macos hwhuang$ ./brokerchain_academic_macos
[-----] [-----] [-----] [-----] [-----]
[-----] [-----] [-----] [-----] [-----] [-----]
[-----] [-----] [-----] [-----] [-----] [-----] [-----] (academic)

BrokerChain仅供学术交流使用，用户不得使用BrokerChain从事任何非法活动。  
用户使用BrokerChain所产生的任何直接或间接后果，均与BrokerChain创始团队无关。  
BrokerChain创始团队保留随时修改、更新或终止BrokerChain的权利，且无需事先通知用户。  
用户在使用BrokerChain时，应自行承担风险，并同意放弃对创始团队的任何索赔权利。  
本免责声明受中华人民共和国法律管辖，并按照其解释。

Welcome. Please enter an option:  
1: Join BrokerChain as a consensus node && Open the wallet.  
2: Open a wallet.  
3: Query an account and its balance if given an address.  
4: Transfer tokens to another account.  
5: Claim BKC tokens through faucets.  
1  
Please enter an option:  
1: Generate a pair of (public/private) keys for a new account  
2: Use the private key of an existing account  
2  
Please enter the filename for the private key:  
pk1  
Please enter an option:  
S: Join a Senior Shard  
Otherwise: Join a Junior Shard
```

Two choices.

```
Please enter the filename for the private key:  
pk1  
Please enter an option:  
S: Join a Senior Shard  
Otherwise: Join a Junior Shard  
S  
-----*****  
Your account address is: [c38171707e1f23323442eaacf10f495cd2d48c00]  
The browser wallet URL is: [http://127.0.0.1:53819]  
-----*****  
  
Start trying to join BrokerChain network...  
Join a senior shard of BrokerChain network successfully.  
Connect successfully, waiting construct new shard...  
2025/07/13 10:00:55 Consensus gets started...  
proxy: academic.broker-chain.com  
port: 56743  
Generating a new blockchain &{0xc0000ae240}  
There is no existed blockchain in the database.  
The newest block is updated  
Block is added  
New genesis block
```

图 4.4: Choose to use the private key of an existing account. Then, you can join either a Senior Shard or a Junior Shard.

### 4.2.3 执行共识协议，获取出块奖励



```
BrokerChain仅供学术交流使用，用户不得使用BrokerChain从事任何非法活动。  
用户使用BrokerChain所产生的任何直接或间接后果，均与BrokerChain创始团队无关。  
BrokerChain创始团队保留随时修改、更新或终止BrokerChain的权利，且无需事先通知用户。  
用户在使用BrokerChain时，应自行承担风险，并同意放弃对创始团队的任何索赔权利。  
本免责声明受中华人民共和国法律管辖，并按照其解释。  
  
Welcome. Please enter an option:  
1: Join BrokerChain as a consensus node && Open the wallet.  
2: Open the wallet.  
3: Query an account and its balance if given an address.  
4: Transfer tokens to another account.  
5: Claim BKC academic tokens through faucets.  
1  
Please enter an option:  
1: Generate a pair of (public/private) keys for a new account  
2: Use the private key of an existing account  
2  
Please enter the filename for the private key:  
pk17  
-----  
Your account address is: [2882dec6193baa0779010d68e22ac7904ff88b44]  
The browser wallet URL is: [http://127.0.0.1:45334]  
-----  
Start trying to join BrokerChain network...  
Join BrokerChain network successfully.  
Connect successfully, waiting construct new shard...  
|
```

图 4.6: 一个节点虽然加入了网络，但是需要等待与其他几个节点一起组成一个新的“分片共识组”，方可开始参与共识并获取出块奖励。您可能需要等待几分钟。

如图4.6所示，节点通过 PoW (Proof-of-Work, 工作量证明) 后，等待加入一个新的“普通桌”分片。由于一个分片执行共识协议至少需要一定数量的共识节点（例如 8 个），因此需要等待额外的其它几个节点加入区块链网络，与您的共识节点组成一个新的分片之后，您的共识节点才真正开始参与分片共识并出块。若您的客户端等待时间过久，您可以重复前面的步骤，使用不同账户的私钥启动新的共识节点加入区块链网络。这样可以减少组成一个新分片的等待时长。

如图4.7所示，组成一个新的分片后，客户端界面显示了当前节点的浏览器钱包地址，并且客户端程序会自动在浏览器中弹出一个页面展示这个

```
Welcome. Please enter an option:  
1: Join BrokerChain as a consensus node && Open the wallet.  
2: Open the wallet.  
3: Query an account and its balance if given an address.  
4: Transfer tokens to another account.  
5: Claim BKC academic tokens through faucets.  
1  
Please enter an option:  
1: Generate a pair of (public/private) keys for a new account  
2: Use the private key of an existing account  
2  
Please enter the filename for the private key:  
pk17  
-----  
Your account address is: [2882dec6193baa0779010d68622ac7904ff88b44]  
The browser wallet URL is: [http://127.0.0.1:45334]  
-----  
Start trying to join BrokerChain network...  
Join BrokerChain network successfully.  
Connect successfully, waiting construct new shard...  
2025/06/27 14:10:06 Consensus gets started...  
Generating a new blockchain &{0xc0001167e0}  
There is no existed blockchain in the database.  
The newest block is updated  
Block is added  
New genesis block  
[23 12293 12293:23]  
received the PrePrepare ...
```

Get started !

图 4.7: 该新节点与其他若干 miner 节点共同组成了一个新的分片并开始共识出块。其中，读者可以看到界面上显示了两个重要的信息：1) Your account address；2) The browser wallet URL。读者可以复制 browser wallet URL 到浏览器，就可以看到图4.8所示的网页钱包页面了。

钱包的界面。同时，共识节点将会开始执行共识协议进行出块，该共识节点的账户将获得出块奖励。

如图4.8所示，浏览器钱包展示了当前节点的账户地址和账户余额。随着节点执行共识协议出块，该共识节点的账户将持续获得出块奖励，账户余额会不断增加。另外，浏览器钱包还提供了转账的功能，在输入框中分别输入“收款账户地址 (Recipient Address)”、“转账金额 (Amount)”、与“交易手续费 (Fee)”，即可点击 Transfer 按钮发起一笔转账交易。

# BrokerChain Wallet

## Account

Your Account's Balance: 6.366040572035113468

Your Account's Address:  
2882dec6193baa0779010d68622ac7904ff88b44

## Transfer

Recipient Address
Amount
Fee
<b>Transfer</b>

图 4.8: 浏览器钱包界面。

### 4.2.4 在 MacOS 系统运行 BrokerChain Miner Client

如果您的操作系统是 MacOS, 请按照以下解释与操作步骤来启动 BrokerChain (academic) miner 客户端程序。

1. 下载 BrokerChain(academic) executable file 的 release 压缩文件, 并解压文件到任意文件夹, 不要删除任何文件夹内的文档与子文件夹。而且, 如果您不要双击启动 `brokerchain_academic_macos` 文件, 系统会提示“无法打开”。可以打开系统设置中的“隐私与安全”页面, 找到“安全性”一栏, 系统会提示“是否继续打开 `brokerchain_academic_macos`”点击“继续打开”。
2. 打开一个 terminal 终端; 手动输入命令 “`cd {directory of the executable program}`”, 进入 `brokerchain_academic_macos` 所

在的文件夹；在终端中输入如下命令：

“`chmod 777 brokerchain_academic_macos`”，赋予权限给该可执行程序。

3. 然后输入 “`./brokerchain_academic_macos`”（注意：`./`后没有空格），即可开始运行程序。
4. 您需要手动复制钱包的浏览器地址，方可在浏览器中访问钱包页面。

在 Mac OS 启动运行 `brokerchain_academic_macos` 的过程中，您可能会遇到的一些“阻力”，其原因解释如下。

- 注意不能直接双击 `brokerchain_academic_macos` 文件运行它，因为这会因为出现路径错误而导致程序运行错误。需要在 `terminal` 终端中进入该可执行文件所在的文件夹，再使用命令  
`“./brokerchain_academic_macos”` 开始运行它。
- 双击 `brokerchain_academic_macos` 时，可能 MacOS 会弹出无法识别“UTF-8”的提示框。这是因为此时 MacOS 系统是在尝试以文稿的形式打开该程序文件。您可以关闭错误提示窗口后，按照上述章节4.2.4中的操作指引重新运行。
- 由于 IP 列表种子服务器部署在国外，所以可能会出现无法连接上服务器，或者网络连接 `time out`。这时，需要您多次尝试，直至连接上服务器为止。
- 打开 `terminal` 终端，如果直接拖动 `brokerchain_academic_macos` 到 `terminal` 终端后“回车”运行它，会让该程序在操作系统默认的根目录下执行，生成的私钥等信息也会保存在系统默认的根目录。系统默认的根目录往往是当前 MacOS 登录用户名所在的路径。

- 由于区块链需要良好的网络链接，在出现网络连接问题的提示时，请尝试调整 IP 或者 DNS 为默认设置，或者尝试切换多种不同的网络连接方式，例如您可以切换使用别的 WiFi 热点、手机热点等等。

#### 4.2.5 运行客户端程序注意事项

You must use a dedicated, unique private key to run a miner client. Any attempt to run multiple clients using the same private key will trigger a warning message “join failed, try again later”.

Don’t worry if your executable client shows the `view change` message, which means your executable client is trying to find a new leader of the shard committee. It will try to join a new shard when the “`view change`” lasts more than 11 iterations. So, please don’t turn off your terminal if you see “`view change`” messages in your miner client. Just keep it running. Everything is okay.

If you use an old version of the executable client program, your account balance may remain unchanged. If so, you must check whether the client has an updated version from the GitHub project.

#### 4.2.6 Miner 客户端程序常见的误操作及相应处理

在运行 Miner 共识节点时，您可能会遇到如下几种常见的“误操作”：

- 不小心把运行共识节点的笔记本电脑合上了屏幕，那么笔记本电脑的程序都会进入暂停状态；
- 误点击了“退出程序”按钮；

- 网络掉线了；
- 台式机断电了；
- 一直提示 view change 相关信息；
- 客户端程序卡住不动了，原因不明；…

当您参与到 Juinor Shard 时，如果遇到上述情形或者等等类似情况时，可以重新启动客户端程序即可，不用担心自己的账户余额受到损失。因此，我们建议读者使用 Miner 功能时，要尽量保障运行共识节点的机器运行良好、网络畅通，而且最好一直持续运行客户端程序，否则会影响参与共识的收益，具体原因请参照激励机制相关章节的表述。

但如果当您参与到 Senior Shard 时，任何掉线、脱离了共识网络，都会被扣除掉 Stake threshold 的“门票”费。因此，参与 Senior Shard 时，更要尽一切可能保障运行共识节点的机器运行良好、网络畅通。

## 4.3 Developer: 部署智能合约

Question: 开发者如何往 BrokerChain 上部署智能合约?

### 4.3.1 访问 Remix 在线 IDE 网站

Remix 是一个主流的支持智能合约开发的在线 IDE。如图4.9所示，打开浏览器，访问 Remix 网站<sup>1</sup>。

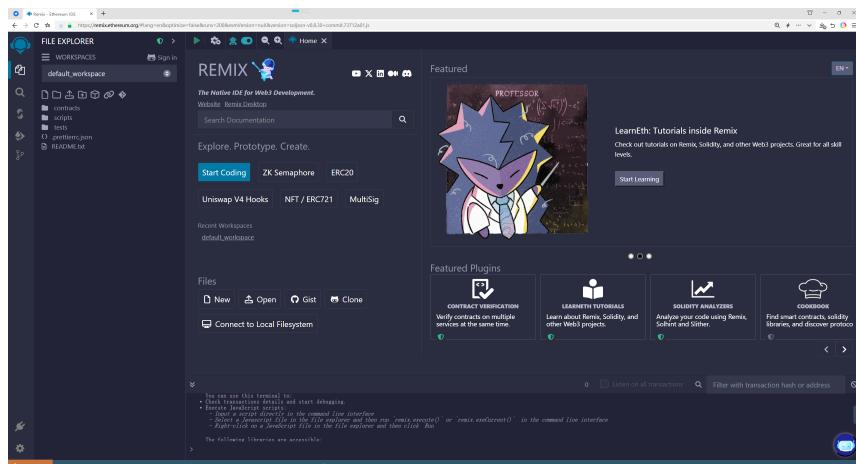


图 4.9: 访问 Remix 在线 IDE 网站。

### 4.3.2 编写智能合约

如图4.10所示，点击 File Explorer -> Create New File ，输入 A.sol，新建智能合约。

<sup>1</sup><https://remix.ethereum.org/>

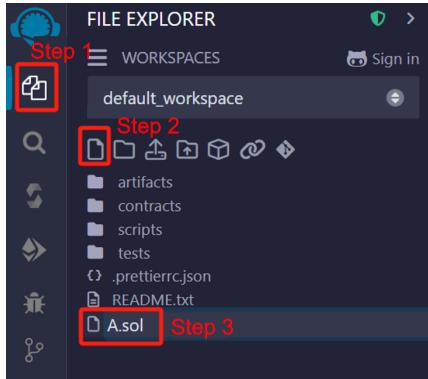


图 4.10: 新建一个智能合约。

```
1 contract A {
2     uint256 public a;
3
4     constructor() payable{
5         a=10;
6     }
7
8     function addA() payable public returns(uint256){
9         a++;
10        return a;
11    }
12
13    function getA() payable public returns(uint256){
14        return a;
15    }
16 }
```

图 4.11: 编写智能合约。

如图4.11所示，在 A.sol 编写一个简单的智能合约。这个智能合约包含了构造器、一个 uint256 类型的状态变量和两个读写这个状态变量的函数。

### 4.3.3 编译智能合约

如图4.12所示，点击左侧第三个按钮，跳转至合约编译页面。选择 Solidity 版本为 0.8.26，点击 Compile A.sol 按钮编译智能合约。

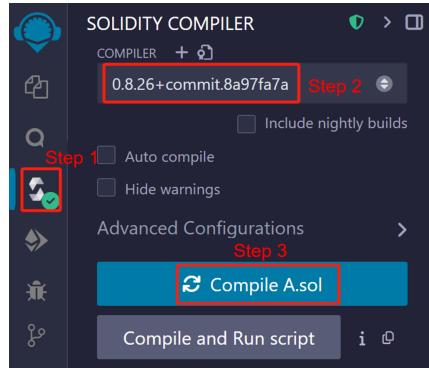


图 4.12: 编译智能合约。

#### 4.3.4 连接 BrokerChain

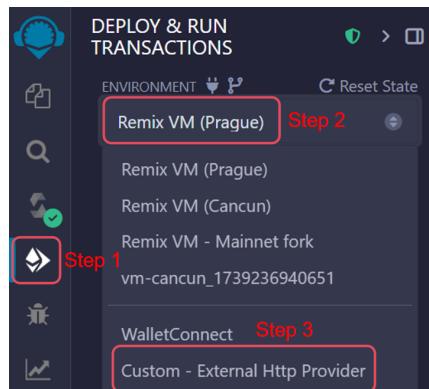


图 4.13: 连接 BrokerChain。

如图4.13所示，点击左侧 Deploy & run transactions 按钮，点击 Environment 按钮，在下拉列表中选择 Custom - External Http Provider。

如图4.14所示，在 Custom - External Http Provider 的 Endpoint 中填写您启动 BrokerChain 时得到的浏览器钱包的地址，然后点击 OK。

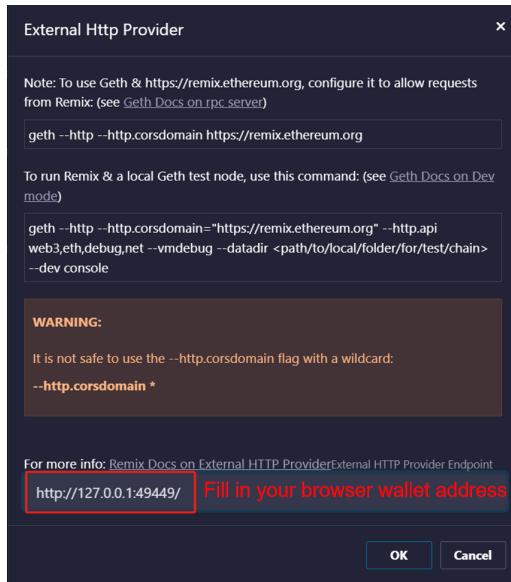


图 4.14: 填写 BrokerChain 浏览器钱包的地址。

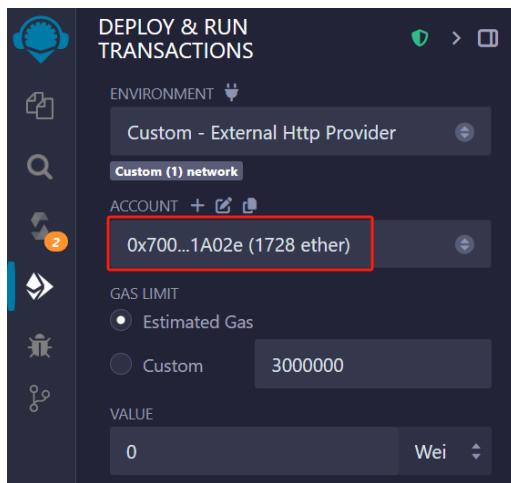


图 4.15: 查看您输入的账户的地址和余额。

如图4.15所示，Account 框中展示了您账户的地址和余额，这些余额可用于支付调用智能合约时消耗的 Gas（Gas 可理解为调用智能合约需要付出的“燃料费”）。

#### 4.3.5 部署智能合约

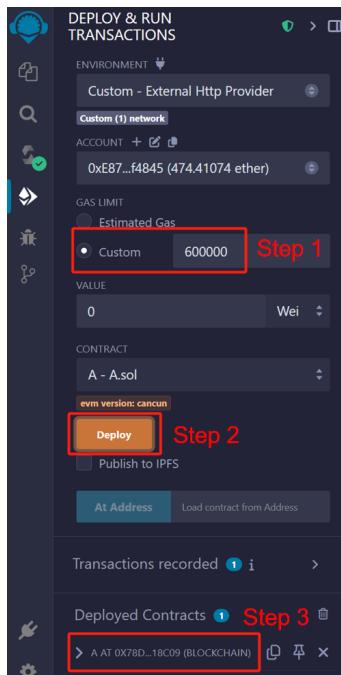


图 4.16：部署智能合约。

如图4.16所示，在 GAS LIMIT 选项中选择 Custom 并输入 600000（若 GAS LIMIT 过低可能会导致部署智能合约失败），点击 Deploy 按钮部署智能合约。部署成功后，下方的 Deployed Contracts 中就会展示该智能合约的地址。

#### 4.3.6 调用智能合约

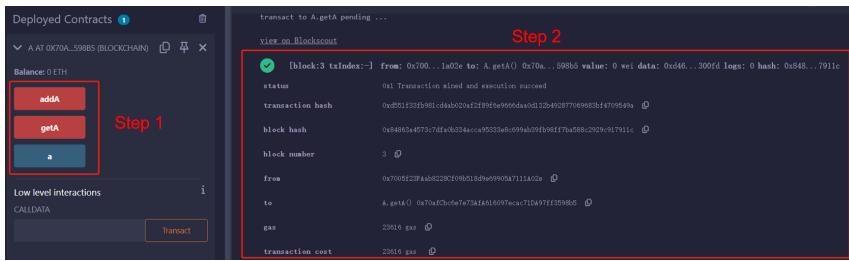


图 4.17: 调用一个智能合约的示例。

如图4.17所示，点击 Deployed Contracts 中的合约函数，调用该智能合约。调用成功后，控制台展示了该次调用的相关信息，如状态、交易哈希、区块哈希、消耗的 GAS 量等。

## 4.4 User: 使用 BrokerChain Wallet 体验 dApps

### 4.4.1 下载 BrokerChain Wallet

如图4.18所示, 从 BrokerChain Wallet 的 Github 仓库<sup>1</sup> Releases 页面下载编译好的 apk 文件, 发送到手机进行安装。目前 BrokerChain Wallet 仅支持安卓手机操作系统, 未来会发布 iOS 版本的钱包软件。

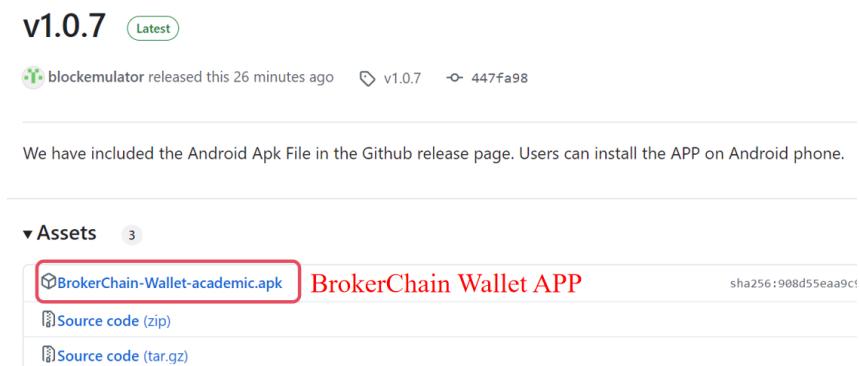


图 4.18: 下载 BrokerChain Wallet。

### 4.4.2 创建钱包

如图4.19所示, 打开 BrokerChain Wallet, 输入密码和确认密码创建钱包。该密码将作为您以后解锁使用钱包的唯一凭证, 请妥善保管。

<sup>1</sup><https://github.com/HuangLab-SYSU/brokerwallet-academic>

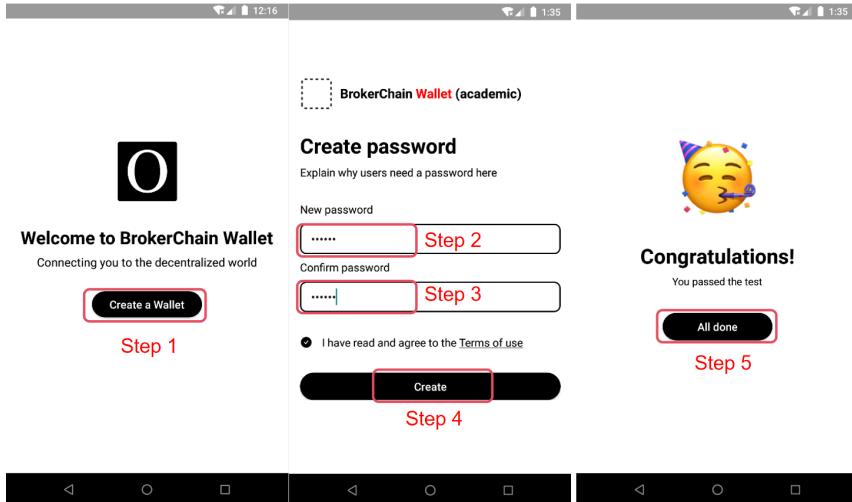


图 4.19: 创建钱包。

#### 4.4.3 导入账户

如图4.20所示，进入 BrokerChain Wallet 主页面后，点击右上角进入菜单列表，点击 `Settings` 选项进入设置页面，点击 `Accounts` 选项进入账户管理页面。

如图4.21所示，进入账户管理页面后，点击 `Add account` 按钮，在 `Account privatekey` 输入框中输入要导入的账户的私钥。点击 `Save` 按钮保存这个账户，账户列表中会显示出刚刚导入的账户的地址及其余额，点击该账户，将该账户选中，后续将以该账户的身份作为付款者账户。

#### 4.4.4 发起转账交易

如图4.22所示，进入 BrokerChain Wallet 主页面。点击 `Send` 按钮，进入转账页面。在 `Send to` 输入框中输入收款者账户地址，在 `Amount` 输入

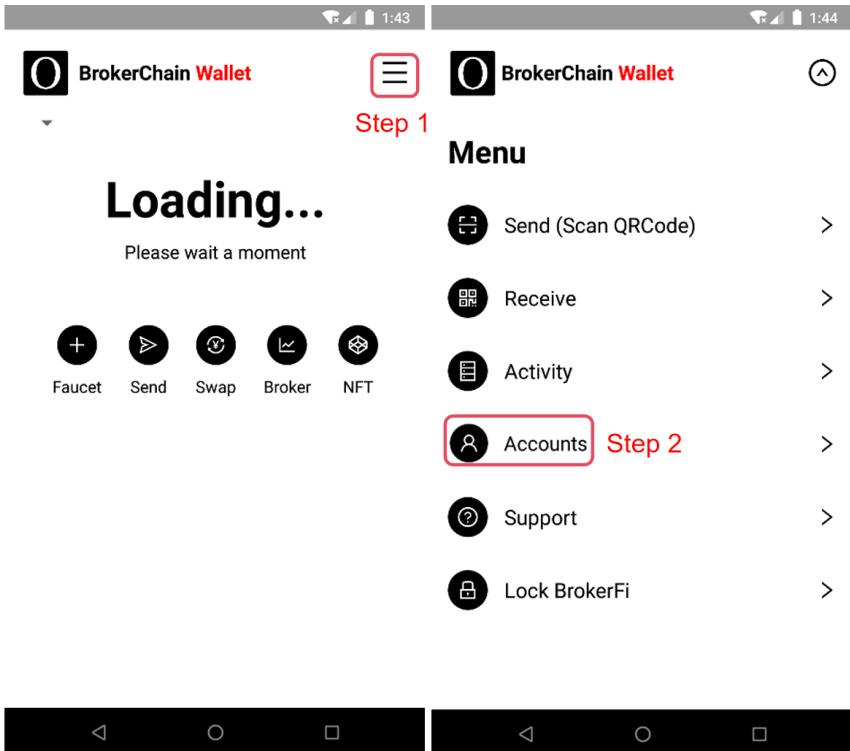


图 4.20: 进入账户管理页面。

框中输入转账金额（单位为 BKC），在 Fee 输入框中输入转账手续费（单位为 BKC）。点击 Send 按钮发起转账交易。转账成功后，返回 BrokerChain Wallet 主页面，能看到转账金额和手续费已从账户余额中扣除。转账金额将在收款者账户到账，转账手续费将会作为 Broker 的收益。如何成为 Broker 并赚取收益，请查看6.1章节。

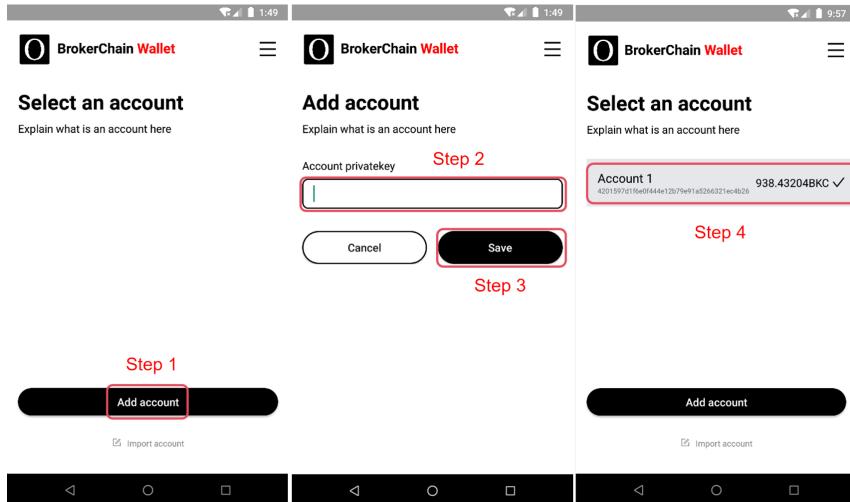


图 4.21: 导入账户。

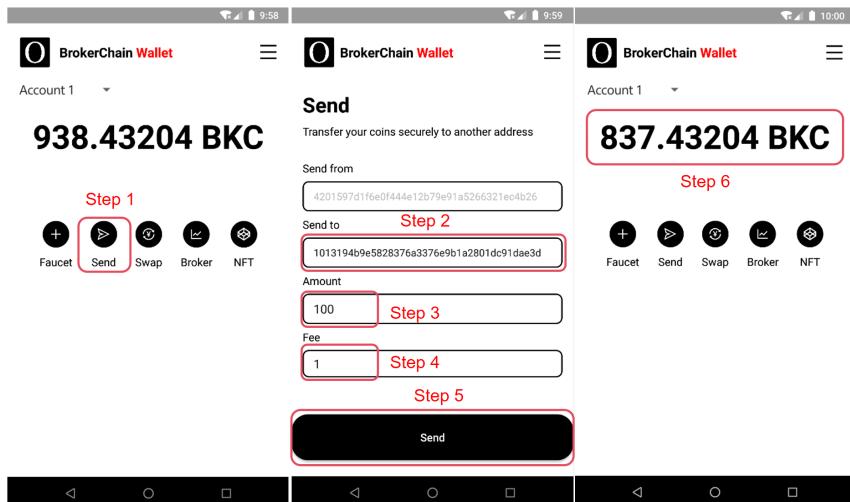


图 4.22: 发起一笔转账交易。

#### 4.4.5 钱包软件每日打卡奖励

当您使用手机端的钱包软件时，如果确保钱包一直在运行（而没有被杀死进程），您的钱包中的账户会每日自动领取一个 BKC。尽管您的钱包中导入了多个账户，每日签到领取的 BKC 数量仍然只有一个。这是为了避免有人生成大量的账户导入钱包后领取大量的签到奖励。

### 4.5 使用“水龙头”领取少量 BKC

BrokerChain Wallet 提供了“水龙头 (Faucet)”的功能，任何用户均可免费领取少量 BKC，可用于体验、做测试、或者当做加入 miner 网络的“启动资金”。如图 4.23 所示，点击 **Faucet** 按钮，进入水龙头页面。点击 **Claim** 按钮即可领取一定数量的通证。用户每个账户每天只能领取少量次数。领取成功后，返回 Wallet 主页面，就能看到账户余额增加了。

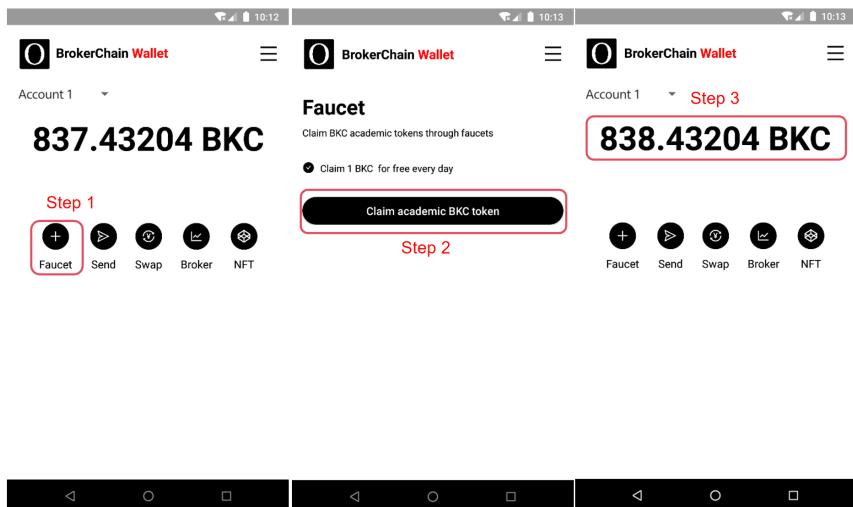


图 4.23: 通过“水龙头 (Faucet)”领取少量 BKC。

## 4.6 BrokerChain 的区块链浏览器

为了方便公众查询 BrokerChain 链上的一些数据，我们技术团队开发了面向公众的 BrokerChain 区块链浏览器（BrokerChain Dashboard），网址为：<http://academic.broker-chain.com:56741/>

在 BrokerChain 的区块链浏览器页面，任何人可以执行如下操作：

- Query the number of tokens minted in recent few hours.
- Query an account and its balance if given an address.
- Claim a few BKC tokens through faucets.

此外，为了支持开发者（developers）在 BrokerChain 上构建 dApp，我们还设置了面向开发者的“水龙头”代币领取功能，即 **Faucet for developers**。通过这个功能，developers 可以提交申请来领取一定数量 BKC 代币用于开发测试，但是需要阐述具体原因与用途，以及提交一些必要的沟通方式（例如 email，接收 BKC 的 account address 等信息）。

该浏览器页面的功能会逐渐完善与丰富，请读者自行探索。

## 第五章 未来升级计划概述

关于 BrokerChain 的未来路线简介，这里使用图5.1来简单阐释。

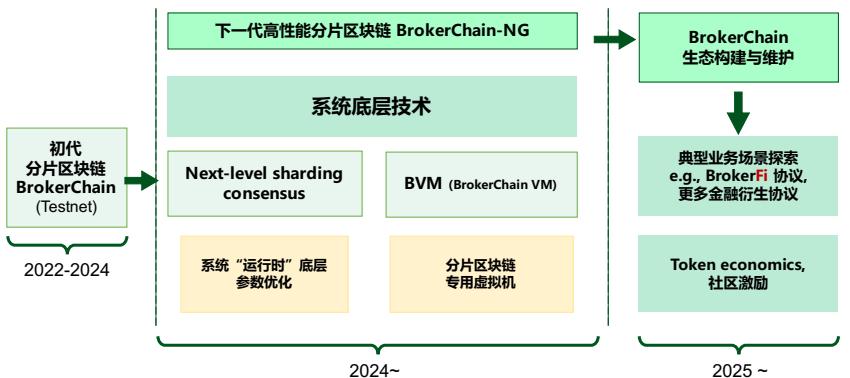


图 5.1: Roadmap of BrokerChain and its next-generation version.

## 第六章 代表性 dApp Demo 示例

本章展示在 BrokerChain 上构建的 3 个典型 dApp 的 demo 示例、说明与复用代码。

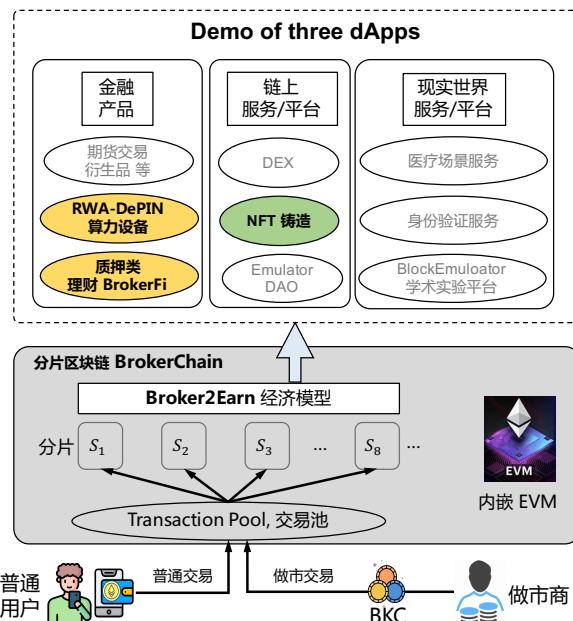


图 6.1: BrokerChain (academic) 链上构建的 3 个 dApps 的 Demo 示例。

## 6.1 BrokerFi：构建在 BrokerChain 上的流动性质押 DeFi 协议

Slogan: *Anyone can become a market maker if using BrokerFi.*

### 6.1.1 BrokerFi 简介

BrokerFi [7] 是构建在 BrokerChain 分片区块链之上的首个 DeFi 理财产品，旨在为普通用户提供类似“余额宝”的低风险、稳定收益投资机会。该协议通过创新的 Broker2Earn 机制，让普通用户能够成为“做市商”，参与跨分片交易处理并获得稳定收益，有效降低了用户参与 DeFi 的门槛。

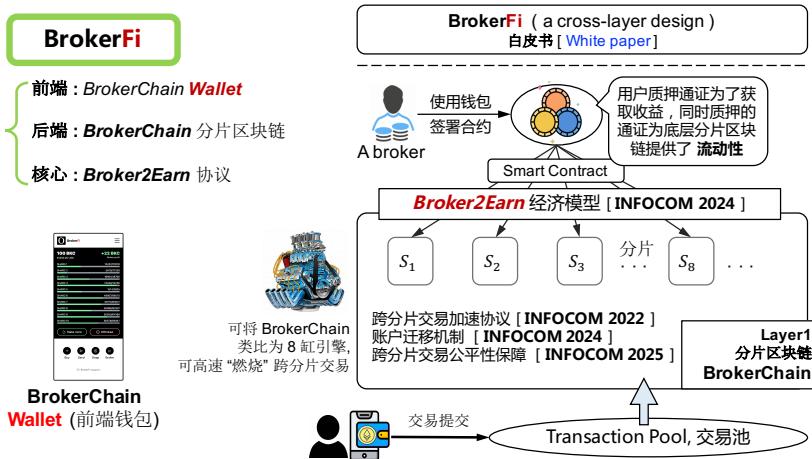
与传统 DeFi 协议相比，BrokerFi 具有以下特点：

- **低门槛**：无需复杂的金融知识，操作简单直观。
- **低风险**：资金仅质押在协议中，不会外流。
- **稳定收益**：通过参与跨分片交易获得手续费分成。

这些特点的设计使得 BrokerFi 能够有效降低普通用户参与 DeFi 的门槛。传统 DeFi 协议往往需要用户具备深入的金融知识和对复杂机制的理解，而 BrokerFi 采用了类似“余额宝”的设计理念，用户只需要进行简单的质押操作即可参与，整个流程被设计得极其直观。在风险控制方面，相比于传统 DeFi 协议中资金可能被借贷给未知第三方或投入高风险策略的情况，BrokerFi 中用户的资金仅在协议内进行质押，专门用于支持跨分片交易的流动性需求，大大降低了本金损失的风险。在收益稳定性方面，BrokerFi 的收益来源于 BrokerChain 网络中跨分片交易的手续费分成，由

于跨分片交易是网络运行的基础需求，具有相对稳定的频率和规模，因此用户能够获得更加可预期和稳定的收益回报。

## BrokerFi 架构设计



[ White paper ] BrokerFi: A DeFi dApp Built Upon Broker-Based Blockchain, IEEE ICPADS 2023.  
[1] BrokerChain: A Cross-Shard Blockchain Protocol for Account/Balance-based State Sharding (INFOCOM 2022)  
[2] Broker2Earn: Towards Maximizing Broker Revenue and System Liquidity for Sharded Blockchains (INFOCOM 2024)  
[3] Account Migration across Blockchain Shards using Fine-tuned Lock Mechanism (INFOCOM 2024)  
[4] Justitia: An Incentive Mechanism towards the Fairness of Cross-shard Transactions (INFOCOM 2025)

图 6.2: BrokerFi 架构——a cross-layer design.

如图6.2所示，BrokerFi 采用三层架构设计：

1. 前端：自研 BrokerFi [7] 数字钱包作为用户交互的主要界面，提供完整的数字资产管理功能。该钱包不仅支持基础的转账、收款、兑换等操作，更重要的是集成了独特的 Broker 质押功能，让用户能够通过简单的界面操作参与到 BrokerChain 网络的流动性提供中。钱包的 UI 设计，确保用户体验的流畅性和直观性。前端的具体介绍会在本章节之后进行详细的讲解。

2. **后端**: BrokerChain [3] 分片区块链作为整个系统的技术基础设施，提供高性能、低延迟的区块链服务。相比传统单链架构，BrokerChain 通过分片技术实现了更高的吞吐量和更低的交易成本。该层负责处理**所有的链上交易、智能合约执行以及跨分片交易**的协调，为上层应用提供稳定可靠的区块链服务支撑，更多介绍可以参照本书第 2.1 章节的内容。
3. **核心**: Broker2Earn [1] 经济协议是连接前端用户需求和后端技术能力的关键桥梁。该协议定义了用户质押、收益分配、风险控制等核心机制，通过精心设计的激励模型确保系统的长期稳定运行。协议基于严格的数学模型和经济学原理，能够在保障用户利益的同时优化整个网络的性能表现。

如图6.2所示，BrokerFi 的架构清晰地展现了三层设计的协同工作机制。在前端层面，用户通过 BrokerFi 去中心化金融应用与系统进行交互，Broker 账户持有者可以通过钱包界面进行质押操作以获取收益。在后端基础设施层面，Layer2 展示了 BrokerChain 分片区块链的核心架构，包含多个分片 (S1、S2、S3...Sn)，这些分片共同构成了高性能的区块链网络。在核心协议层面，Broker2Earn 经济模型作为连接用户和底层区块链的关键机制，负责管理质押资金在各个分片间的分配，并协调跨分片交易的处理。这种设计确保了 BrokerFi 既能发挥 BrokerChain 分片技术的高性能优势，又能兼容现有的 DeFi 生态，为用户提供更广泛的应用场景。

### 6.1.2 BrokerFi 的核心机制 - Broker2Earn 协议

本章节会简要介绍 BrokerFi 的核心机制 Broker2Earn 协议 [1]。在 BrokerFi 中，Broker2Earn 协议主要起如下几个作用：

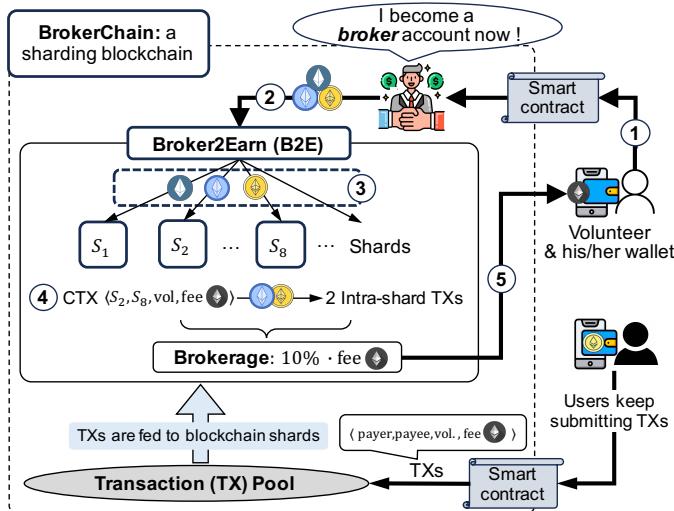


图 6.3: 一个用户 (volunteer) 使用钱包可以参与 Broker2Earn 协议，通过操作步骤 1-5，即可完成完整的流程：选择成为 broker 角色；质押 tokens 到 Broker2Earn protocol；tokens 被 B2E 算法匹配到 CTXs；tokens 用来“消灭”CTXs；B2E protocol 回馈一定百分比的交易手续费给 broker。

- 最大化 “Broker (做市商账户)” 的质押收益；
- 质押的 tokens 为后端分片区块链提供 “流动性”，可加速交易的上链；
- 让用户自愿质押成为 Broker，参与进 BrokerFi 生态；
- 与 BrokerChain 分片区块链在共识协议层面进行协作。

图6.3介绍了 Broker2Earn 协议是如何为一个钱包用户和 BrokerChain 互动的。此处我们以一个普通钱包用户为视角简单描述：

1. 一个普通用户使用钱包选择成为 broker 账户
2. Broker 将 token 抵押为 B2E 协议；

3. B2E 协议将 token 分配给区块链分片；
4. Broker 的 token 将一个跨分片交易 (CTX, Cross-shard Transaction) 转换为 2 个片内交易 (ITX, Intra-shard Transaction)。注：这里的  $\langle S_2, S_8, \text{vol}, \text{fee} \rangle$  表示交易是从位于分片  $S_2$  的付款人账户向位于分片  $S_8$  的另一个收款人账户发出的，交易额为 vol，同时支付数目为 fee 的手续费；
5. Broker 因其流动性服务而收取手续费  $10\% \cdot \text{fee}$ 。注：佣金率 10% 是返回给 broker 的手续费分成的可调参数。

### 6.1.3 BrokerFi 的安装使用

从 Github 下载安装 BrokerChain Wallet<sup>1</sup>到安卓手机上。此 Wallet 包含了 BrokerFi 的功能。

### 6.1.4 申请成为 Broker

如图6.4所示，进入 BrokerChain Wallet 主页面后，点击 Broker 按钮，进入申请成为 Broker 的页面。页面中展示了申请成为 Broker 的智能合约。点击 Apply 按钮后，系统将执行这个智能合约，当前账户将成为 Broker。

### 6.1.5 质押 token 到 Broker2Earn

如图6.5所示，申请成为 Broker 成功后，进入质押资金页面。在 Amount 输入框中输入要质押到 Broker2Earn 的 token 数额，点击 Stake 按钮确认质押。系统将会从账户余额中扣除相应的数额。

---

<sup>1</sup><https://github.com/HuangLab-SYSU/brokerwallet-academic>

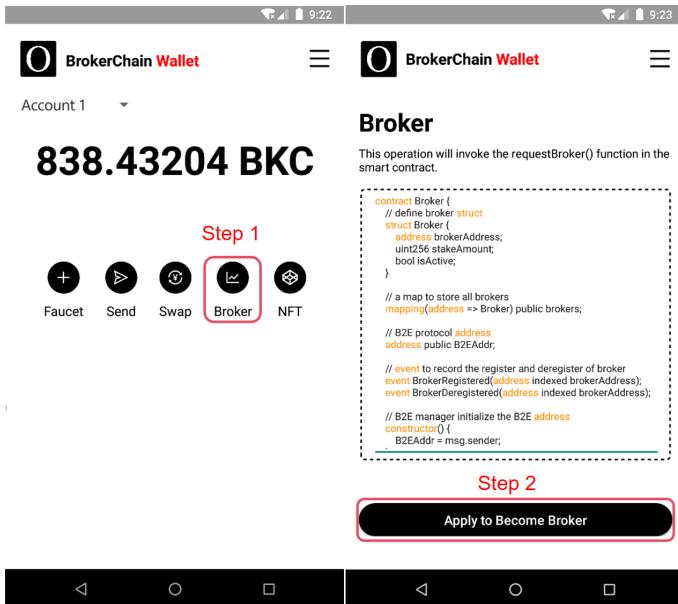


图 6.4: 申请成为 Broker

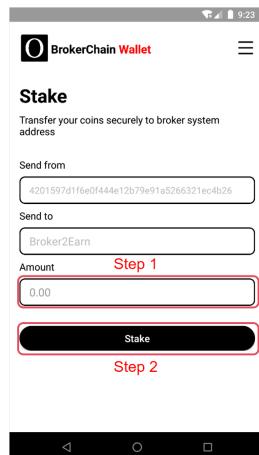
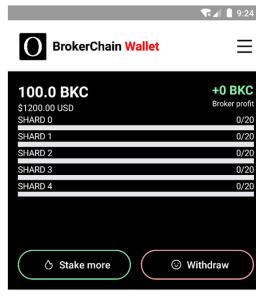


图 6.5: 质押资金到 Broker2Earn



© BrokerFi support



图 6.6: 查看 Broker 收益

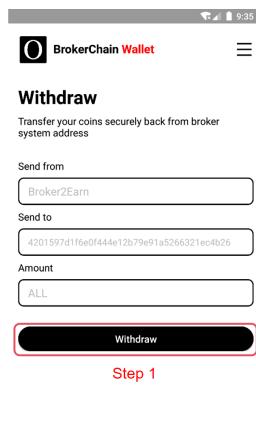


图 6.7: 取消 Broker 身份

### 6.1.6 查看 Broker 收益

如图6.6所示，质押成功后，进入 Broker 收益查看页面。可以查看质押金额和收益。在中部列表中可以查看 Broker 在各个分片的收益明细情况。用户提交转账交易到系统后，Broker 将有机会获取交易的手续费作为收益。

### 6.1.7 取消 Broker 身份

用户赚取收益后，若想要取消 Broker 身份，可以点击图6.6的 Withdraw 按钮，跳转至图6.7的取消 Broker 身份页面。点击 Withdraw 按钮，确认取消 Broker 身份，Broker 的本金和收益将一并返还到账户。

### 6.1.8 演示视频

可在线查看 BrokerFi 的演示视频<sup>1</sup>。

---

<sup>1</sup><https://j4s9d119cd.feishu.cn/docx/AIa3dA1P7oCUUQx54Rcc2fZinRe>

## 6.2 NFT-Fi：构建在 BrokerChain 上的 NFT 流通性增强方案

Slogan: 让 NFT 碎片化，增强 NFT 的流通性。

### 6.2.1 项目背景

非同质化代币（Non-Fungible Token, NFT）作为一种基于区块链技术的数字资产，因其唯一性、不可分割性及不可篡改性，确保每个 NFT 具有独特的链上标识，从而为创作者和持有者的提供确权依据，增强资产安全性。这些特性使 NFT 成为数字艺术品、虚拟资产确权和收藏品市场的核心工具，并逐渐拓展到游戏、音乐、身份认证等多个领域，成为区块链技术应用的重要代表。

然而，在 NFT 行业快速发展的过程中，仍存在诸多挑战，尤其是在用户体验、交易效率与创新机制方面。首先，现有 NFT 平台的操作门槛较高，普通用户难以入门。现有 NFT 平台通常要求用户具备一定的区块链知识，才能完成 NFT 的铸造、存储和交易。例如，用户需要配置数字货币钱包、理解智能合约的工作机制，并熟悉 Gas Fee 的支付方式。这对于缺乏技术背景的用户而言，不仅学习成本高，而且可能因误操作导致资产损失，从而影响 NFT 的普及率。

其次，交易成本高且效率受限。当前大多数 NFT 的铸造和交易依赖于以太坊（Ethereum）等智能合约公链。尽管以太坊已从工作量证明（PoW）转向权益证明（PoS），降低了能源消耗，但其网络吞吐量仍受限制，交易处理速度较慢，且 Gas 费用较高。尤其在交易量激增时，网络拥堵导致的交易延迟更为明显，严重影响了交易的效率和用户体验。

最后，现有 NFT 平台在交易形式上的创新性相对不足。当前主流平台普遍采用“单一用户买入-持有-卖出”的传统线性交易逻辑，难以支持“多人共同持有单个 NFT”的复杂资产共享机制。以 OpenSea 为例，该平台主要支持单一用户对 NFT 的上架与出售，并未提供多人持有、按持有份额交易或共享收益等功能。这些功能缺失在很大程度上源于区块链底层共识机制的特性：所有交易需经过节点间一致性确认方可上链，从而导致交易吞吐量受限，难以支持更复杂的多人协同逻辑。这不仅限制了 NFT 在交易结构上的延展性，也阻碍了其在多元化社区应用场景中的进一步发展。

因此，针对 NFT 交易门槛高、交互复杂、持有模式单一等问题，本项目致力于构建一个创新的 NFT 应用平台，用户可通过该平台便捷地铸造、查看、交易 NFT，并实现多人共同持有。为实现这一目标，本项目围绕用户交互体验、区块链架构及 NFT 持有模式进行了优化：

- 1. 降低技术门槛，优化用户交互体验。：**本平台基于数字钱包 Broker-Chain Wallet 开发，它提供了简洁直观的用户界面，即使是没有区块链基础的用户，也能轻松完成 NFT 的铸造与交易。用户可通过手机钱包上传实物藏品照片，并将其一键转换为链上 NFT 资产，简化了 NFT 创建与管理的流程，降低了用户的认知成本。
- 2. 优化底层架构，提高交易吞吐量与成本效率。：**本平台采用 Broker-Chain 的高效跨分片交易机制，能够存储大量 NFT 持有数据，提高交易吞吐量，并降低交易延迟与 Gas 费用。相比传统公链，Broker-Chain 的架构更适用于多人共同持有 NFT 的交易模式，使 NFT 的确权、转让和共享持有更加高效。
- 3. 引入多人持有机制，实现 NFT 的共享与灵活转让。：**本平台通过智

能合约支持多人共同持有单一 NFT，即多个用户可按比例共享 NFT 资产，并可自由转让其持有份额。这一机制不仅降低了单个 NFT 的持有成本，使更多用户能够参与，还提升了 NFT 的交易流动性。此外，智能合约确保 NFT 数据安全上链、所有权数据不可篡改、交易透明可追溯，为多人持有模式提供安全保障。

本项目通过上述优化措施，有效提升了 NFT 交易的效率，降低了交易成本和用户准入门槛，为 NFT 行业发展出更具共享性和多样性的交易模式提供了新的探索方向。

### 6.2.2 系统架构

整个系统由三个核心部分组成：

1. **BrokerChain Wallet (钱包前端)**: 提供用户交互入口。用户可以通过移动端进行 NFT 的铸造、上架、购买等操作，所有动作最终会通过 Web3 接口提交到区块链上。
2. **BrokerChain (区块链平台)**: 兼容 EVM 的智能合约执行环境，支持 Solidity 合约的部署与调用。它负责接收用户操作、处理交易、记录状态变更，并通过区块记录所有交易历史，确保数据不可篡改。
3. **NFT 智能合约**: 这是系统的核心逻辑层，定义了 NFT 相关操作的规则。用户的每次操作最终都会触发合约内部状态的更新。

### 6.2.3 系统核心逻辑：多人持有

本项目的核心特色之一是单个 NFT 可供多人持有，这一机制通过“份数”(shares) 实现。**铸造者在 NFT 生成时决定其总份数，且总份数在铸造**

后不可更改，NFT 持有者无法自行拆分份数。如果 NFT 仅有 1 份，则为单人持有；若份数大于 1，则可由不同用户分别持有该 NFT 的一定份数，实现多人共同持有，从而提升 NFT 的流动性与市场适应性。此外，“份数制”也优化了后端的数据存储，后端系统使用 uint 类型存储 NFT 的份数，便于增减数量，简化交易记录的管理。

在交易过程中，NFT 持有者可以选择自己拥有的部分份数进行上架，并为不同部分设定不同价格。买家则可以购买不超过挂单份数的 NFT 份额，最终的交易价格为份数乘以单价加上 Gas 费用，交易完成后，该 NFT 的相应份数所有权转移给买家，原持有者的份额同步减少。

这一机制依赖智能合约确保所有权明确且不可篡改，所有 NFT 的份数信息均记录在区块链上，以确保资产的唯一性且可追溯。智能合约采用事件机制实时广播交易变更，防止数据不同步，同时通过哈希映射高效存储和更新持有者的份额信息。每笔交易都由智能合约自动结算，避免超额出售或份数计算错误，使多人持有机制在高并发环境下依然安全可靠。

#### 6.2.4 NFT 模块使用说明

从 Github 下载安装 BrokerChain Wallet<sup>1</sup>到安卓手机上。此 Wallet 包含了 NFT-Fi 的功能。

如图6.8所示，运行 BrokerFi，打开应用，点击 *NFT* 按钮进入 NFT 导航主界面。点击 *Mint* 按钮进入 NFT 铸造页面。

进入“NFT 铸造”页面后，用户可通过拍照或从相册中选择图片作为 NFT 素材，并填写相关信息进行铸造操作。点击 *Camera* 按钮后，系统将跳转至拍照界面，用户拍摄完成的照片将实时显示在上传区域内。

---

<sup>1</sup><https://github.com/HuangLab-SYSU/brokerwallet-academic>

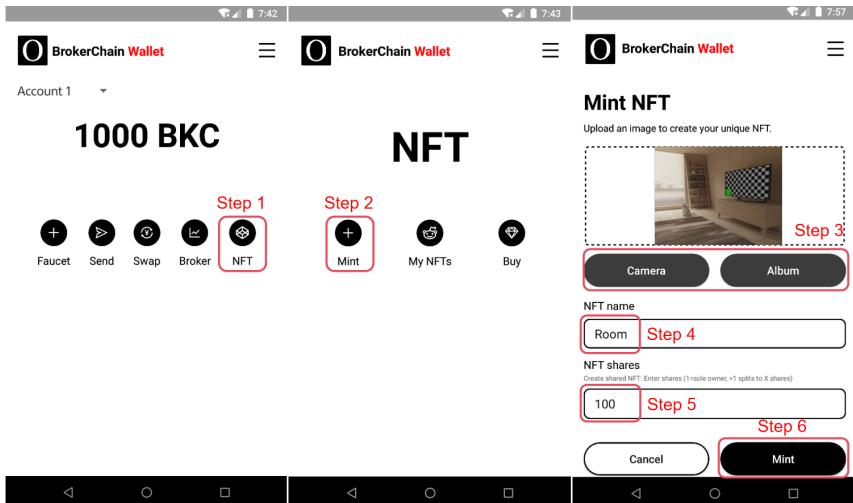


图 6.8: 铸造 NFT

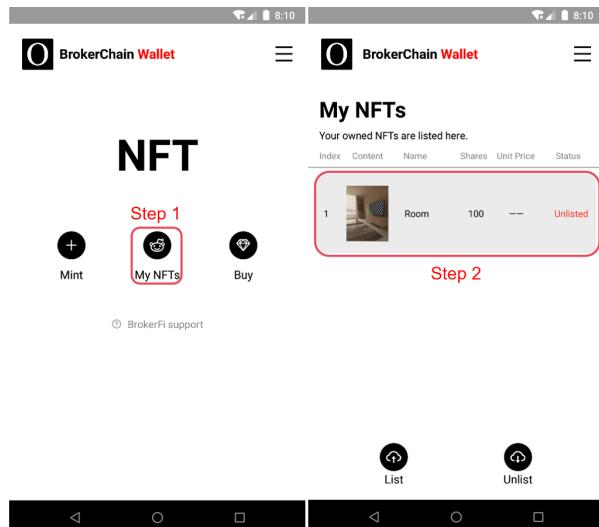


图 6.9: 查看拥有的 NFT

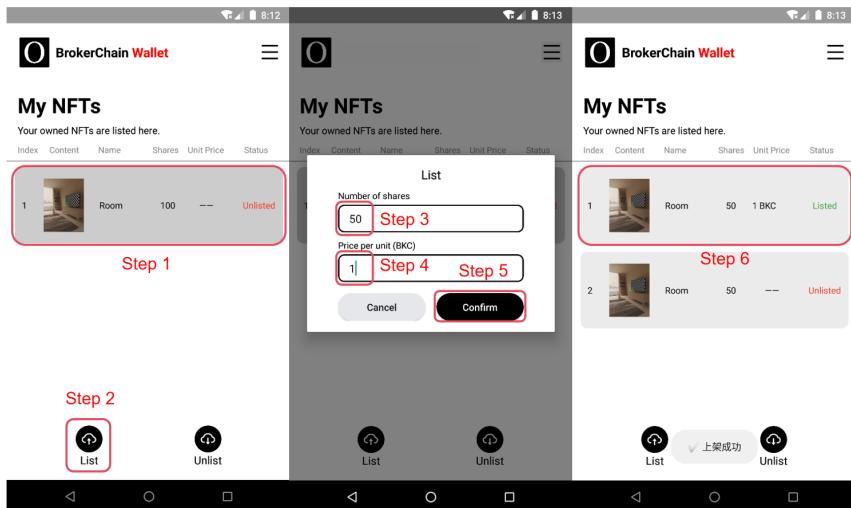


图 6.10: 上架 NFT

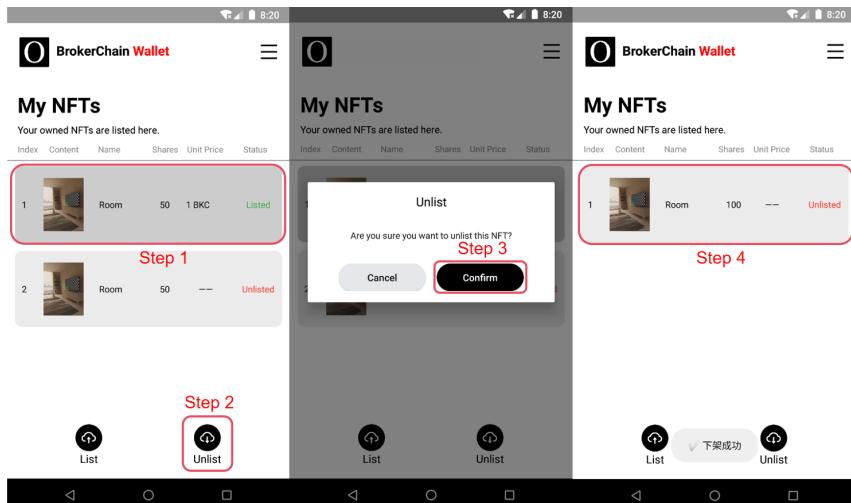


图 6.11: 下架 NFT

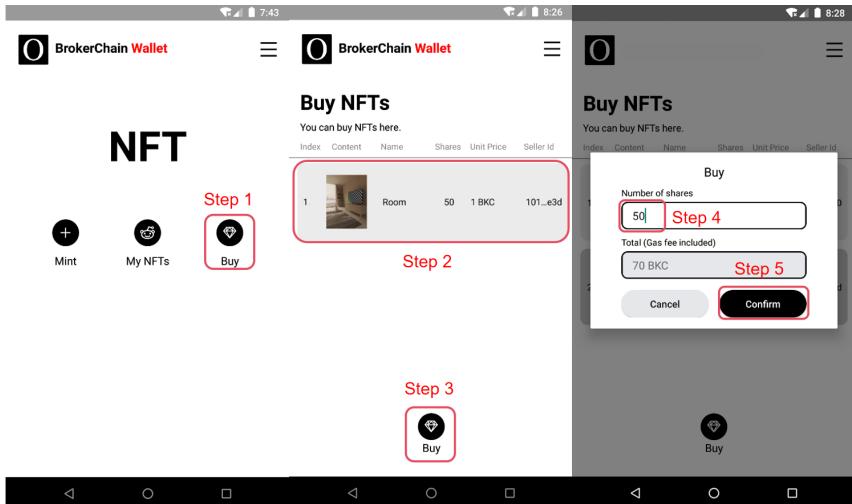


图 6.12: 购买 NFT

若点击 *Album* 按钮，则可从本地相册中选择一张图片进行上传。上传成功后，选中的图片将展示在虚线框内。接着，用户需填写 NFT 的基本信息，包括名称以及总份额。其中，份额为 1 表示该 NFT 为独占式持有；份额大于 1 则允许多人按比例共享所有权。完成信息填写后，点击 *Mint* 按钮，即可发起铸造请求。铸造操作默认收取 10 BKC 的 Gas 费用。

如图6.9所示，进入“我的 NFT”页面后，用户已成功铸造的 NFT 会在此展示。

如图6.10所示，用户可对自己持有的 NFT 执行上架或下架操作。选择想要上架的 NFT，点击 *List* 按钮后，会弹出上架操作窗口。用户需输入上架份数（不得超过可用份数）以及每份 NFT 的价格，点击 *Confirm* 按钮完成上架。成功后，对应的上架信息将出现在界面中。

平台支持对同一 NFT 以不同价格和份数进行多次上架操作。当用户继续上架同一个 NFT，并设置不同的价格与份数时，系统将生成新的挂单

信息。

若再次以上一次相同价格上架该 NFT，系统不会新增挂单，而是自动合并份数，将相同价格的挂单进行更新。这一机制有效减少了存储冗余与界面信息的重复展示。

如图6.11所示，用户点击已上架 NFT 的 *Unlist* 按钮，在弹出的确认框中点击 *Confirm* 即可完成下架操作。下架成功后，后台将删除该挂单记录，并将对应 NFT 份额返还至用户的可用份数中。前端界面会以价格为“——”、状态为“Unlisted”的方式展现记录，可见原未上架的 50 份 NFT 与下架的 50 份 NFT 自动合并为一条记录。同理，用户可对其他 NFT 进行上架或下架操作，所有持有记录均在“我的 NFT”页面统一显示。

如图6.12所示，在“NFT 交易市场”页面中，用户可以浏览平台上所有已成功挂单的 NFT 信息（不包括自己发布的挂单）。当用户选择心仪的 NFT 并点击 *Buy* 按钮后，系统将弹出购买操作窗口。用户需输入购买的份数（不得超过当前挂单份额），系统将自动计算总价，包括所购 NFT 总价、默认的 10 BKC 铸造者版税以及 10 BKC 的 Gas 费用。确认无误后，点击 *Confirm* 按钮即可发起购买操作。

购买成功后，后台将自动减少该挂单记录的剩余 NFT 份数，并及时将交易信息反映在界面中。与此同时，用户所购买的 NFT 将同步显示至“我的 NFT”页面中，供用户进一步查看与操作。

### 6.2.5 前端 BrokerChain Wallet 中 NFT 代码说明

本系统的前端主要由导航页面和功能页面组成。每个页面由一个 Java 控制类与对应的 XML 布局文件配合完成，实现逻辑处理与界面渲染的分离，页面之间通过 Intent 实现跳转。以下为本项目新增或修改的 Java 或布局文件内容。

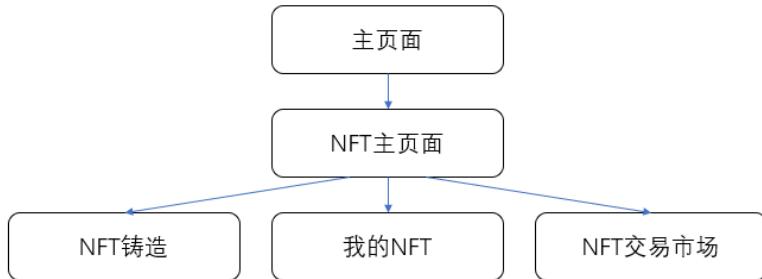


图 6.13: 前端系统架构图

### 1). MainActivity.java

新增代码在原先主界面的基础上添加了 NFT 跳转按钮，按下可跳转 NFTMain 界面。

```

1 nft.setOnClickListener(view -> {
2     //创建意图对象
3     Intent intent = new Intent();
4     intent.setClass(MainActivity.this,NFTMainActivity.class);
5     //跳转
6     startActivity(intent);
7 });

```

### 2). NFTMainActivity.java

此 javay 文件以及相应的 xml 文件对应了导航界面。界面中具有三个按钮，用于跳转不同的 NFT 功能界面。

### 3). MintActivity.java

NFT 铸造界面，用户输入信息进行 NFT 铸造。

```
1 private void initResultLaunchers() {
2     // 相机结果监听
3     cameraLauncher = registerForActivityResult(
4         new ActivityResultContracts.
5             StartActivityForResult(),
6         result -> {
7             if (result.getResultCode() == RESULT_OK) {
8                 handleCameraImage();
9             }
10        });
11
12     // 相册结果监听
13     galleryLauncher = registerForActivityResult(
14         new ActivityResultContracts.
15             StartActivityForResult(),
16         result -> {
17             if (result.getResultCode() == RESULT_OK &&
18                 result.getData() != null) {
19                 handleGalleryImage(result.getData());
20             }
21        });
22 }
```

以上函数用于初始化相机/相册返回结果监听。

```
1 btn_doCamera.setOnClickListener(v -> handleCameraClick());
2 btn_doFile.setOnClickListener(v -> handleGalleryClick());
```

点击相机/相册按钮后，会跳转到请求权限函数。

```
1 private void handleCameraClick() {
2     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
```

```
3         if (checkSelfPermission(Manifest.permission.CAMERA)
4             != PackageManager.PERMISSION_GRANTED)
5         {
6             requestPermissions(new
7                 String[]{Manifest.permission.CAMERA},
8                 PERMISSION_CAMERA);
9
10        return;
11    }
12
13    startCamera();
14
15 }
16
17 private void handleGalleryClick() {
18     // Android 13+ 使用新权限
19     if (Build.VERSION.SDK_INT >=
20         Build.VERSION_CODES.TIRAMISU) {
21
22         if (checkSelfPermission(Manifest.permission
23             .READ_MEDIA_IMAGES)
24             != PackageManager.PERMISSION_GRANTED) {
25
26             requestPermissions(new
27                 String[]{Manifest.permission.READ_MEDIA_IMAGES},
28                 PERMISSION_STORAGE);
29
30         return;
31     }
32
33     // Android 6.0+ 使用旧权限
34     else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
35
36         if (checkSelfPermission(Manifest.permission.
37             READ_EXTERNAL_STORAGE) !=
38             PackageManager.PERMISSION_GRANTED) {
39
40             requestPermissions(new
```

```
    String []{Manifest.permission.  
        READ_EXTERNAL_STORAGE}, PERMISSION_STORAGE);  
    25         return;  
    26     }  
    27 }  
    28 // 直接启动相册  
    29 startGallery();  
  30 }
```

```
 1 public void onRequestPermissionsResult(int requestCode,  
  2     @NonNull String[] permissions, @NonNull int[]  
  3     grantResults) {  
  4     super.onRequestPermissionsResult(requestCode,  
  5         permissions, grantResults);  
  6     if (requestCode == PERMISSION_STORAGE) {  
  7         if (grantResults.length > 0 && grantResults[0] ==  
  8             PackageManager.PERMISSION_GRANTED) {  
  9             // 权限通过后直接启动相册  
 10             startGallery();  
 11         } else {  
 12             Toast.makeText(this, "需要权限才能访问相册",  
 13                 Toast.LENGTH_SHORT).show();  
 14         }  
 15     }  
 16     if (requestCode == PERMISSION_CAMERA) {  
 17         if (grantResults.length > 0 && grantResults[0] ==  
 18             PackageManager.PERMISSION_GRANTED) {  
 19             startCamera();  
 20         } else {  
 21             Toast.makeText(this, "需要权限访问相机",
```

```
        Toast.LENGTH_SHORT).show();  
    }  
}  
}
```

此处代码用于处理请求返回。

若请求权限成功，会打开相机/相册：

```
1 private void startCamera() {  
2     try {  
3         File photoFile = createImageFile();  
4         imageUri = FileProvider.getUriForFile(this,  
5             "com.example.brokerfi.fileprovider", photoFile);  
6         Intent cameraIntent = new  
7             Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
8         cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT,  
9             imageUri);  
10        cameraLauncher.launch(cameraIntent);  
11    } catch (IOException e) {  
12        Toast.makeText(this, "创建文件失败",  
13            Toast.LENGTH_SHORT).show();  
14    }  
15 }  
16  
17 private void startGallery() {  
18     try {  
19         // 使用更兼容的 ACTION_GET_CONTENT  
20         Intent intent = new Intent(Intent.ACTION_GET_CONTENT);  
21         intent.setType("image/*");  
22         intent.addCategory(Intent.CATEGORY_OPENABLE);  
23         galleryLauncher.launch(intent);  
24     } catch (Exception e) {  
25     }  
26 }
```

```
21 // 备用方案：使用文件浏览器
22 Intent fallback = new Intent(Intent.ACTION_VIEW);
23 fallback.setData(Uri.parse(Environment.
24         getExternalStoragePublicDirectory(Environment.
25             DIRECTORY_PICTURES).toString()));
26 startActivity(fallback);
27 }
28 }
```

```
1 // 处理相机图片
2 private void handleCameraImage() {
3     try {
4         Bitmap bitmap = MediaStore.Images.Media.getBitmap(
5             getContentResolver(), imageUri);
6         uploadView.setImageBitmap(bitmap);
7         hasImage = 1;
8     } catch (IOException e) {
9         Toast.makeText(this, "加载图片失败",
10            Toast.LENGTH_SHORT).show();
11    }
12 }
13
14 // 处理相册图片
15 private void handleGalleryImage(Intent data) {
16     if (data != null && data.getData() != null) {
17         imageUri = data.getData();
18         try {
19             Bitmap bitmap =
20                 MediaStore.Images.Media.getBitmap(
21                     getContentResolver(), imageUri);
22             uploadView.setImageBitmap(bitmap);
23         }
24     }
25 }
```

```
19         hasImage = 1;
20     } catch (IOException e) {
21         Toast.makeText(this, "加载图片失败",
22                         Toast.LENGTH_SHORT).show();
23     }
24 }
```

以上代码用于将图片加载进虚线框中。

```
1 private void intEvent() {
2     btn_mint.setOnClickListener(view -> {
3
4         String nftname = edt_nft_name.getText().toString();
5         String sharesStr = edt_shares.getText().toString();
6
7         //判断是否有NFT图片
8         if (hasImage == 0) {
9             Toast.makeText(this, "请先上传图片",
10                         Toast.LENGTH_SHORT).show();
11
12             return;
13
14         //验证名字是否输入
15         if (nftname.isEmpty()) {
16             edt_nft_name.setError("必填字段");
17             edt_nft_name.requestFocus();
18             Toast.makeText(MintActivity.this,
19                         "请输入NFT名字", Toast.LENGTH_SHORT).show();
20             return;
21
22         // 验证份数
23     }
24 }
```

```
21     if (sharesStr.isEmpty()) {
22         edt_shares.setError("必填字段");
23         edt_shares.requestFocus();
24         Toast.makeText(MintActivity.this,
25             "请输入NFT总份数", Toast.LENGTH_SHORT).show();
26         return;
27     }
28
29     try {
30         int share = Integer.parseInt(sharesStr);
31         if (share < 1) {
32             edt_shares.setError("最小为1份");
33             Toast.makeText(MintActivity.this,
34                 "份数不能小于1",
35                 Toast.LENGTH_SHORT).show();
36             return;
37         }
38         if (share > 10000) {
39             edt_shares.setError("超过最大限制");
40             Toast.makeText(MintActivity.this,
41                 "最多将NFT分为10000份",
42                 Toast.LENGTH_SHORT).show();
43             return;
44         }
45     } catch (NumberFormatException e) {
46         edt_shares.setError("格式示例: 100");
47         Toast.makeText(MintActivity.this,
48             "请输入有效的整数份数",
49             Toast.LENGTH_SHORT).show();
50         return;
51     }
52 }
```

```
45      .....
46  });
47 }
```

用户点击 mint 按钮后，系统先判断输入信息是否符合格式要求。

```
1 try {
2     // 1. 直接读取图片字节
3     // 以下注释的是不压缩图片，直接传输的代码
4     // InputStream in =
5         // getContentResolver().openInputStream(imageUri);
6     // ByteArrayOutputStream out = new
7         // ByteArrayOutputStream();
8     // byte[] buffer = new byte[1024];
9     // int len;
10    // while ((len = in.read(buffer)) != -1) {
11        //     out.write(buffer, 0, len);
12    // }
13    // byte[] imageBytes = out.toByteArray();
14
15    InputStream in =
16        // getContentResolver().openInputStream(imageUri);
17    BitmapFactory.Options options = new
18        // BitmapFactory.Options();
19    options.inSampleSize = 4; // 直接缩小4倍
20    Bitmap bitmap = BitmapFactory.decodeStream(in, null,
21        options);
22    in.close();
23
24    //质量压缩，确保图片大小不超过200KB
25    ByteArrayOutputStream out = new ByteArrayOutputStream();
26    int quality = 70; // 初始质量
```

```
22     bitmap.compress(Bitmap.CompressFormat.JPEG, quality, out);
23     while (out.toByteArray().length > 200 * 1024 && quality >
24         50) {
25         out.reset(); // 清空输出流
26         quality -= 10; // 降低质量
27         bitmap.compress(Bitmap.CompressFormat.JPEG, quality,
28             out);
29     }
30
31     // 2. 转换为Base64字符串
32     String base64Image = Base64.encodeToString(imageBytes,
33         Base64.NO_WRAP);
34
35     // 3. 生成交易数据 (使用ABIUtils)
36     BigInteger share = new BigInteger(sharesStr);
37     String data = ABIUtils.encodeMint(nftname, base64Image,
38         share);
39
40     // 4. 发送交易
41     sendMintTransaction(data, BigInteger.TEN);
42
43     // 添加日志
44     Log.d("IMAGE_COMPRESS", "压缩后尺寸: " +
45         bitmap.getWidth() + "x" + bitmap.getHeight());
46     Log.d("IMAGE_COMPRESS", "压缩后大小: " +
47         imageBytes.length + " bytes");
48     Log.d("IMAGE_COMPRESS", "Base64长度: " +
49         base64Image.length());
50
```

```
46 } catch (Exception e) {
47     Toast.makeText(this, "处理失败: " + e.getMessage(),
48         Toast.LENGTH_LONG).show();
49 }
```

输入无误后，将图片压缩（注释掉的代码即不压缩图片的写法），并生成符合 JSON-RPC 规范的交易数据（参考后文 ABIUtils.java），并发送交易“sendMintTransaction”。

```
1 private void sendMintTransaction(String data, BigInteger
2     value) {
3     try {
4         // 构造交易参数
5         JSONObject txParams = new JSONObject();
6         //txParams.put("from",
7             "0x107D73D8A49EEB85d32Cf465507Dd71D507100C1");
8         txParams.put("from",
9             getAccountNumberFromSharedPreferences());
10        txParams.put("to",
11            "0x1855c485845Aa12ab2dFbaB7c29aE2cC4669eF61");
12        txParams.put("data", data);
13        txParams.put("value", "0x" + value.toString(16));
14
15        // 包装请求参数
16        JSONArray params = new JSONArray();
17        params.put(txParams);
18
19        // 构造请求
20        JSONObject request = new JSONObject();
21        request.put("method", "eth_sendTransaction");
22        request.put("params", params);
```

```
19     request.put("id", RequestIdGenerator.getNextId());
20     request.put("jsonrpc", "2.0");
21
22     // 记录完整请求
23     Log.d("MINT_DEBUG", "完整请求 JSON:\n" +
24           request.toString());
25
26     // 记录关键字段
27     Log.d("MINT_DEBUG", "value: " + value);
28     Log.d("MINT_DEBUG", "data length: " + data.length());
29
30     OkhttpUtils.getInstance().doPost(
31         BaseUrl.Base_nft,
32         request.toString(),
33         new MyCallBack() {
34             @Override
35             public void onSuccess(String result) {
36                 try {
37                     // 先检查是否是有效的 JSON
38                     if (result.trim().startsWith("{")) {
39                         JSONObject response = new
40                             JSONObject(result);
41                         if (response.has("error")) {
42                             Toast.makeText(MintActivity.
43                                 this,
44                                 "铸造失败: " + response.
45                                     getString("error"),
46                                     Toast.LENGTH_LONG).show();
47                         } else {
48                             String txHash = response.
49                                 getString("result");
50                         }
51                     }
52                 }
53             }
54         });
55     }
56 }
```

```
45                     checkTransactionStatus(txHash);
46
47             }
48
49         } else {
50
51             // 处理非 JSON 响应 (如 404)
52
53             runOnUiThread(() ->
54
55                 Toast.makeText(MintActivity.
56
57                     this,
58
59                     "服务器错误: " + result,
60
61                     Toast.LENGTH_LONG).show()
62
63             );
64
65         }
66
67     @Override
68
69     public Void onError(Exception e) {
70
71         Toast.makeText(MintActivity.this,
72
73             "Transaction
74
75             Failed", Toast.LENGTH_LONG).show();
76
77         return null;
78
79     }
80
81 }
82
83 );
```

```
73
74     } catch (Exception e) {
75
76         e.printStackTrace();
77
78         Toast.makeText( this, "交易请求失败",
79                         Toast.LENGTH_SHORT).show();
80     }
81 }
```

此函数用于向智能合约发送交易。注意，这里手动填入了预先部署的智能合约地址。如果实际部署的智能合约地址有更改，需要替换掉“to”之后的智能合约地址，也即替换“0x1855c485845Aa12ab2dFbaB7c29aE2cC4669eF61”为新的合约地址。后文的所有交易数据同理。

交易发送成功返回交易哈希后，自动调用 checkTransactionStatus：

```
1 private void checkTransactionStatus(String txHash) {
2
3     try {
4
5         JSONArray params = new JSONArray();
6
7         params.put(txHash);
8
9         JSONObject request = new JSONObject();
10
11        request.put("method", "eth_getTransactionReceipt");
12
13        request.put("params", params);
14
15        request.put("id", RequestIdGenerator.getNextId());
16
17        request.put("jsonrpc", "2.0");
18
19
20        OkhttpUtils.getInstance().doPost(BaseUrl.Base_nft,
21                                         request.toString(),new MyCallBack() {
22
23             @Override
24
25             public void onSuccess(String result) {
26
27                 try {
28
29                     JSONObject response = new
30                     JSONObject(result);
31
32                     if(response.getString("status").equals("0"))
33
34                         checkTxSuccess(result);
35
36                     else
37
38                         checkTxError(result);
39
40                 }
41             }
42
43         });
44
45     }
46
47 }
```

```
16         JSONObject receipt =
17             response.optJSONObject("result");
18
19         if (receipt != null) {
20             String status =
21                 receipt.optString("status", "0x0");
22             if ("0x1".equals(status))
23                 runOnUiThread(() ->{Toast.makeText(
24                     MintActivity.this, "铸造成功",
25                     Toast.LENGTH_SHORT).show();
26
27                 //创建意图对象，跳转祝贺界面
28                 Intent intent = new Intent();
29                 intent.setClass(MintActivity.this,
30                     CongratulationsMintActivity.
31                     class);
32
33                 //跳转
34                 startActivity(intent);
35             });
36         }
37
38         else{
39             runOnUiThread(() ->Toast.makeText(MintActivity.
40                 this, "交易失败！",
41                 Toast.LENGTH_SHORT).show()
42             );
43         }
44     }
45
46     else {
47         // 交易尚未上链，继续轮询
48         new Handler().postDelayed(() ->checkTransactionStatus(txHash),
49             1000);
50     }
51 }
```

```
                2000);
        }
    }
    catch (JSONException e)
    {
        e.printStackTrace();
    }
}

public Void onError(Exception e) {
    Toast.makeText(MintActivity.this,
        "Transaction
Failed",Toast.LENGTH_LONG).show();
    return null;
}
}

);

} catch (JSONException e) {
    e.printStackTrace();
    Toast.makeText(this, "查询请求失败",
        Toast.LENGTH_SHORT).show();
}
}
```

这个函数对应了“getTransactionReceipt”的 RPC 请求。如果返回信息 status==0x1，则表明操作成功，系统跳转到祝贺界面。

#### 4). CongratulationsMintActivity.java

此界面用于标识 NFT 铸造成功。

## 5). MyNFTsActivity.java

“我的 NFT 界面”，用户可以在此查看并管理自己拥有的 NFT，进行上架、下架操作。

```
1 protected void onCreate(Bundle savedInstanceState) {  
2     super.onCreate(savedInstanceState);  
3     setContentView(R.layout.activity_my_nfts);  
4  
5     initView();  
6     intEvent();  
7  
8     // 启动 NFT 数据获取  
9     fetchMyNFTs();  
10 }
```

```
1 private void fetchMyNFTs() {  
2     try {  
3         JSONObject txParams = new JSONObject();  
4         //txParams.put("from",  
5         "0x107D73D8A49EEB85d32Cf465507Dd71D507100C1");  
6         txParams.put("from",  
7             getAccountNumberFromSharedPreferences());  
8         txParams.put("to",  
9             "0x1855c485845Aa12ab2dFbaB7c29aE2cC4669eF61");  
10        //智能合约地址  
11        txParams.put("data", ABIUtils.encodeGetMyNFTs());  
12        txParams.put("value", "0x0");  
13        txParams.put("gas", "0x1");  
14  
15        // 包装请求参数
```

```
13     JSONArray params = new JSONArray();
14     params.put(txParams);
15
16     // 构造请求
17     JSONObject request = new JSONObject();
18     request.put("method", "eth_call");
19     request.put("params", params);
20     request.put("id", RequestIdGenerator.getNextId());
21     request.put("jsonrpc", "2.0");
22
23     // 记录完整请求
24     Log.d("MYNFTS_DEBUG", "获取 NFT 请求 JSON:\n" +
25           request.toString(4));
26
27     OkhttpUtils.getInstance().doPost(BaseUrl.Base_nft,
28         request.toString(), new MyCallBack() {
29
30         @Override
31         public void onSuccess(String result) {
32
33             try {
34
35                 // 先检查是否是有效的 JSON
36
37                 if (result.trim().startsWith("{")) {
38
39                     JSONObject response = new
```

```
        response.getString("result");

40    ABIUtils.MyNFTsResult nfts =
        ABIUtils.
            decodeGetMyNFTs(hexData);

41
42    List<NFT> nftList = new
        ArrayList<>();
43    for (int i = 0; i <
        nfts.nftIds.length; i++) {
44        NFT nft = new NFT(
45            nfts.nftIds[i],
46            getAccountNumberFromSharedPreferences(),
47            nfts.images[i],
48
49            nfts.names[i],
50            nfts.sharesList[i],
51            nfts.pricesList[i],
52            nfts.listingStatus[i],
53            nfts.listingIds[i]
54        );
55        nftList.add(nft);
56    }

57
58    runOnUiThread(() -> {
59        if (nftList.isEmpty()) {
60            Toast.
61                makeText(MyNFTsActivity.
62                    this, "暂无NFT数据",
63                    Toast.LENGTH_LONG)
64                .show();
65    }
66}
```

```
63             NFTData.clear();
64
65             NFTData.addAll(nftList);
66
67             adapter.notifyDataSetChanged();
68         });
69     }
70
71 } else {
72
73     // 处理非 JSON 响应 (如 404)
74
75     runOnUiThread(() ->
76
77         Toast.makeText(MyNFTsActivity.
78
79             this,
80
81             "服务器错误：" + result,
82
83             Toast.LENGTH_LONG).
84
85             show()
86
87         );
88     }
89
90 } catch (JSONException e) {
91
92     e.printStackTrace();
93
94     Log.e("NFT_FETCH", "响应格式错误", e);
95
96     runOnUiThread(() ->
97
98         Toast.makeText(MyNFTsActivity.this,
99
100            "数据解析失败：" + e.getMessage(),
101
102            Toast.LENGTH_LONG).show());
103
104     }
105
106
107     @Override
108
109     public void onError(Exception e) {
110
111         runOnUiThread(() ->
112
113             Toast.makeText(MyNFTsActivity.this,
114
115                 "网络请求失败：" + e.getMessage(),
116
117                 Toast.LENGTH_LONG).show());
118     }
119 }
```

```
        Toast.LENGTH_LONG).show());
86    return null;
87  }
88}
89} catch (Exception e) {
90    runOnUiThread(() ->
91        Toast.makeText(MyNFTsActivity.this,
92        "请求构造失败: " + e.getMessage(),
93        Toast.LENGTH_LONG).show());
94}
95}
```

在初始化 onCreate() 中，系统会自动调用 fetchMyNFTs()，向后端发送请求用户所持有的 NFT 信息。成功获取信息后系统会将其加载到 NFTData 中。

```
1 private void initView() {
2     menu = findViewById(R.id.menu);
3     actionBar = findViewById(R.id.action_bar);
4     btn_list = findViewById(R.id.btn_list);
5     btn_unlist = findViewById(R.id.btn_unlist);
6
7     // 配置 RecyclerView
8     recyclerView = findViewById(R.id.rv_nft_list);
9     recyclerView.setLayoutManager(new
10        LinearLayoutManager(this));
11    adapter = new NFTAdapter(NFTData, false);
12    recyclerView.setAdapter(adapter);
13}
```

initView() 的特色在于使用了 RecyclerView，它是多条数据的动态展示的

承载体，确保页面能够实时渲染用户拥有的 NFT 资产，无需预设固定条目数量。为此，我在项目中新增了适配器“NFTAdapter.java”用于将每条 NFT 信息绑定至 recyclerView，从而在 UI 界面展示 NFT 信息。

“MyNFTs”界面具有两个按钮，一个是“上架”(List)，另一个是“下架”(Unlist)。若用户正确选中 NFT，即可进行上架/下架操作。

```
1 //上架按钮
2 btn_list.setOnClickListener(view -> {
3     NFT selected = adapter.getSelectedItem();
4     int position = adapter.getSelectedPosition();
5
6     if (selected == null) {
7         Toast.makeText(this, "请先选择要上架的NFT",
8             Toast.LENGTH_SHORT).show();
9         return;
10    }
11
12    if (selected.isListed()) {
13        Toast.makeText(this, "此NFT已上架",
14            Toast.LENGTH_SHORT).show();
15        adapter.clearSelection();
16        return;
17    }
18
19    // 创建弹窗
20    AlertDialog dialog = new AlertDialog.Builder(this)
21        ..setView(R.layout.dialog_confirm)
22        .create();
23
24    dialog.setOnShowListener(dialogInterface -> {
```

```
24     TextView tv_title =
25         dialog.findViewById(R.id.tv_title);
26     EditText et_shares =
27         dialog.findViewById(R.id.et_shares);
28     EditText et_price =
29         dialog.findViewById(R.id.et_price);
30     Button btnConfirm =
31         dialog.findViewById(R.id.btn_confirm);
32     Button btnCancel =
33         dialog.findViewById(R.id.btn_cancel);
34
35     // 设置提示信息
36     tv_title.setText("List");
37
38     // 确认操作
39     btnConfirm.setOnClickListener(v -> {
40
41         // 获取输入值
42         String shares =
43             et_shares.getText().toString().trim();
44         String price =
45             et_price.getText().toString().trim();
46
47         // 验证 shares
48         if (shares.isEmpty()) {
49
50             et_shares.setError("必填字段");
51             et_shares.requestFocus();
52             Toast.makeText(MyNFTsActivity.this,
53                 "请输入需要上架的NFT份数",
54                 Toast.LENGTH_SHORT).show();
55
56             return;
57         }
58     }
```

```
46
47 // 验证 price
48 if (price.isEmpty()) {
49     et_price.setError("必填字段");
50     et_price.requestFocus();
51     Toast.makeText(MyNFTsActivity.this,
52         "请输入每份的价格",
53         Toast.LENGTH_SHORT).show();
54
55     return;
56
57     BigInteger shareValue = new BigInteger(shares);
58     BigInteger priceValue = new BigInteger(price);
59     try {
60         if (shareValue.compareTo(BigInteger.ONE) < 0)
61             {
62                 et_shares.setError("最小为1份");
63                 Toast.makeText(MyNFTsActivity.this,
64                     "份数不能小于1",
65                     Toast.LENGTH_SHORT).show();
66
67             return;
68         }
69
70         if (shareValue.
71             compareTo(selected.getShares()) > 0) {
72             et_shares.
73                 setError("超过您所拥有的NFT份数");
74             Toast.makeText(MyNFTsActivity.this,
75                 "超过您所拥有的NFT份数",
76                 Toast.LENGTH_SHORT).show();
77
78             return;
79         }
80     }
81 }
```

```
68     }
69
70     if (priceValue.compareTo(BigInteger.ZERO) <=
71         0) {
72         et_price.setError("最小0.01");
73         Toast.makeText(MyNFTsActivity.this,
74             "价格必须大于0",
75             Toast.LENGTH_SHORT).show();
76         return;
77     }
78 } catch (NumberFormatException e) {
79     et_shares.setError("格式示例：10");
80     et_price.setError("格式示例：0.05");
81     Toast.makeText(MyNFTsActivity.this,
82         "请输入有效数值",
83         Toast.LENGTH_SHORT).show();
84     return;
85 }
86
87 //后端接口逻辑
88 //生成交易数据
89 String data
90 = ABIUtils.encodeList(selected.getId(),
91     shareValue, priceValue);
92
93 //进行交易
94 listNFT(data);
95
96 // 上架成功
97 adapter.clearSelection();
98 dialog.dismiss();
```

```
92     });
93
94     // 取消操作
95     btnCancel.setOnClickListener(v -> {
96         Toast.makeText(this, "已取消上架",
97             Toast.LENGTH_SHORT).show();
98         adapter.clearSelection();
99         dialog.dismiss();
100    });
101
102    // 显示弹窗并设置背景变暗
103    dialog.show();
104    dialog.getWindow().setDimAmount(0.5f);
105});
```

以上架操作为例。用户点击“上架”按钮后，若错误选中 NFT 或未选中 NFT，系统会自动向用户报告错误信息。成功点击“上架”按钮后，系统会弹出确认弹窗（使用了 dialog\_confirm.xml 作为载体）。用户需要输入正确格式的需要上架的 NFT 份额以及价格，并点击“确认”按钮。系统自带输入信息格式判断，若判断正确，则向后端发送交易数据。“adapter.clearSelection()”会清空选中信息，dialog.dismiss() 会关掉弹窗。

```
1 // 上架NFT的后端接口调用
2 private void listNFT(String data) {
3     try {
4
5         // 构造交易参数
6         JSONObject txParams = new JSONObject();
7         //txParams.put("from",
8         "0x107D73D8A49EEB85d32Cf465507Dd71D507100C1");
```

```
8         txParams.put("from",
9             getAccountNumberFromSharedPreferences());
10        txParams.put("to",
11            "0x1855c485845Aa12ab2dFbaB7c29aE2cC4669eF61"); //智能合约地址
12        txParams.put("data", data); // 编码上架数据
13        txParams.put("value", "0x0"); // 无需转账
14        txParams.put("gas", "0x1");
15
16        // 包装请求参数
17        JSONArray params = new JSONArray();
18        params.put(txParams);
19
20        // 构造请求
21        JSONObject request = new JSONObject();
22        request.put("method", "eth_sendTransaction");
23        request.put("params", params);
24        request.put("id", RequestIdGenerator.getNextId());
25        request.put("jsonrpc", "2.0");
26
27        // 记录完整请求
28        Log.d("LIST_DEBUG", "完整请求JSON:\n" +
29              request.toString(4));
30
31        OkhttpUtils.getInstance().doPost(BaseUrl.Base_nft,
32            request.toString(), new MyCallBack() {
33                @Override
34                public void onSuccess(String result) {
35                    try {
36                        // 先检查是否是有效的 JSON
37                        if (result.trim().startsWith("{")) {
```

```
34                 JSONObject response = new
35                     JSONObject(result);
36
37             if (response.has("error")) {
38
39                 Toast.makeText(MyNFTsActivity.this,
40                     "上架失败: " + response.
41                         getString("error"),
42                         Toast.LENGTH_LONG).show();
43
44             // 刷新 NFT 数据
45             fetchMyNFTs();
46
47         } else {
48
49             String txHash =
50                 response.getString("result");
51
52             checkListTransactionStatus(txHash);
53
54         }
55
56     }
57
58     @Override
```

```
57     public Void onError(Exception e) {
58         runOnUiThread(() ->
59             Toast.makeText(MyNFTsActivity.this,
60                 "上架失败: " + e.getMessage(),
61                 Toast.LENGTH_LONG).show());
62         return null;
63     }
64 }
65 }
66
67 // 检查上架交易状态
68 private void checkListTransactionStatus(String txHash) {
69     try {
70         JSONArray params = new JSONArray();
71         params.put(txHash);
72         JSONObject request = new JSONObject();
73         request.put("method", "eth_getTransactionReceipt");
74         request.put("params", params);
75         request.put("id", RequestIdGenerator.getNextId());
76         request.put("jsonrpc", "2.0");
77
78         OkhttpUtils.getInstance().doPost(BaseUrl.Base_nft,
79             request.toString(), new MyCallBack() {
80
81             @Override
82             public void onSuccess(String result) {
83                 try {
```

```
82         JSONObject response = new
83             JSONObject(result);
84
85         JSONObject receipt =
86             response.optJSONObject("result");
87
88         if (receipt != null) {
89             String status =
90                 receipt.optString("status",
91                     "0x0");
92
93             if ("0x1".equals(status)) {
94                 // 上架成功，更新本地数据
95                 runOnUiThread(() -> {
96                     Toast.makeText(MyNFTsActivity.
97                         this, "上架成功",
98                         Toast.LENGTH_SHORT).
99                         show();
100
101                 });
102             } else {
```

```
103                         // 交易尚未上链，继续轮询
104                         new Handler().postDelayed(() ->
105                             checkListTransactionStatus(txHash),
106                             2000);
107                     }
108
109                 } catch (JSONException e) {
110                     e.printStackTrace();
111                 }
112
113             @Override
114             public Void onError(Exception e) {
115                 runOnUiThread(() ->
116                     Toast.makeText(MyNFTsActivity.this,
117                         "上架失败：" + e.getMessage(),
118                         Toast.LENGTH_LONG).show();
119
120                     return null;
121                 });
122             } catch (JSONException e) {
123                 e.printStackTrace();
124                 runOnUiThread(() ->
125                     Toast.makeText(MyNFTsActivity.this,
126                         "查询请求失败", Toast.LENGTH_SHORT).show());
127             }
128         }
129     }
```

ListNFT() 用于向后端发送符合格式的上架请求。与铸造 NFT 发送的信息请求类似，这里也有查看交易状态的函数：checkListTransactionStatus()。其中，上架成功与否都会调用 fetchMyNFTs 函数，相当于刷新

本地 NFT 信息。

下架操作 unlistNFT() 与上架操作同理。

## 6). adapter 文件夹中的 NFTAdapter.java & model 文件夹中的 NFT.java & nft\_holder.xml

```
1 package com.example.brokerfi.xc.model;
2
3 import java.math.BigInteger;
4 import java.util.Objects;
5
6 //数据模型类，存放单条NFT视图数据
7 public class NFT {
8     private final String imageBase64;
9     private final BigInteger NFTId;
10
11    private final String accountNumber;
12
13    private final String name;
14    private final BigInteger shares;
15    private final BigInteger price;
16    private final boolean isListed;
17    private final BigInteger listingId;
18
19    public NFT(BigInteger nftId, String accountNumber, String
20               imageBase64, String name,
21               BigInteger shares, BigInteger price, boolean
22               isListed, BigInteger listingId) {
23
        this.NFTId = nftId;
        this.accountNumber = accountNumber;
        this.imageBase64 = imageBase64;
```

```
24         this.name = name;
25
26         this.shares = shares;
27
28         this.isListed = isListed;
29
30         this.listingId = listingId;
31
32
33     }
34
35
36     // Get 方法
37
38     public BigInteger getId() { return NFTId; }
39
40
41     public String getAccountNumber() { return accountNumber; }
42
43
44     public String getName() { return name; }
45     public BigInteger getPrice() { return price; }
46     public BigInteger getShares() { return shares; }
47     public boolean isListed() { return isListed; }
48     public BigInteger getListingId(){ return listingId;}
49
50     @Override
51
52     public boolean equals(Object o) {
53
54         if (this == o) return true;
55
56         if (o == null || getClass() != o.getClass()) return
57
58             false;
59
60         NFT nft = (NFT) o;
61
62
63     }
64
65     public void setName(String name) { this.name = name; }
66
67     public void setShares(BigInteger shares) { this.shares = shares; }
68
69     public void setPrice(BigInteger price) { this.price = price; }
70
71     public void setIsListed(boolean isListed) { this.isListed = isListed; }
72
73     public void setListingId(BigInteger listingId) { this.listingId = listingId; }
```

```

54         return isListed == nft.isListed &&
55             Objects.equals(price, nft.price) &&
56             NFTId == nft.NFTId &&
57             Objects.equals(accountNumber,
58                 nft.accountNumber) &&
59             Objects.equals(shares, nft.shares);
60     }
61
62     @Override
63     public int hashCode() {
64         return Objects.hash(name, accountNumber, isListed,
65             price, shares);
66     }
67 }
```

model 文件夹中的 NFT.java 用于存放展示在 nft\_holder 中的单条 NFT 信息。它并不等同于后端存储的 NFT 元数据，而是为了 UI 展示方便而将所有 NFT 持有信息的原子化集合。具体而言，不同用户所持有的每个具有不同状态（未上架、以不同价格上架）的 NFT 都会作为前端的一条“NFT”数据存储。

```

1 public class NFTAdapter extends
2     RecyclerView.Adapter<NFTAdapter.ViewHolder> {
3
4     private List<NFT> nftData;
5
6     //存放本地测试数据
7
8     private int lastSelectedPosition = -1;
9
10    private final boolean
11        fromBuyNFTsActivity;//标示是否从 BuyNFTsActivity 进入
12
13    public NFTAdapter(List<NFT> localData, boolean
14        fromBuyNFTsActivity) {
```

```
8         this.nftData = localData;
9
10        this.fromBuyNFTsActivity = fromBuyNFTsActivity;
11
12    }
13
14    //ViewHolder类，用于存储和显示NFT的视图
15    public static class ViewHolder extends
16        RecyclerView.ViewHolder {
17
18        ImageView ivImage;
19
20        TextView tvId, tvName, tvShares, tvPrice, tvSaleStatus;
21
22
23        public ViewHolder(@NonNull View itemView) {
24
25            super(itemView);
26
27            tvId = itemView.findViewById(R.id.textView_id);
28            ivImage =
29                itemView.findViewById(R.id.iv_thumbnail);
30            tvName = itemView.findViewById(R.id.tv_name);
31            tvShares = itemView.findViewById(R.id.tv_shares);
32            tvPrice = itemView.findViewById(R.id.tv_price);
33            tvSaleStatus =
34                itemView.findViewById(R.id.tv_sale_status);
35
36        }
37
38    }
39
40    //把nft_holder.xml布局文件转换为View对象，
41    //就能加载进RecyclerView（也就是parent）
42
43    @NonNull
44    @Override
45
46    public ViewHolder onCreateViewHolder(@NonNull ViewGroup
47
48        parent, int viewType) {
49
50        View view = LayoutInflater.from(parent.getContext())
51
52            .inflate(R.layout.nft_holder, parent, false);
53
54    }
55
56
```

```
34         return new ViewHolder(view);
35     }
36
37     //将单个NFT数据绑定到ViewHolder上
38     @Override
39     public void onBindViewHolder(@NonNull ViewHolder holder,
40             int position) {
41         NFT item = nftData.get(position);
42         holder.tvId.setText(String.valueOf(position + 1));
43
44         // Base64图片加载
45         if (item.getImageBase64() != null &&
46             !item.getImageBase64().isEmpty()) {
47             try {
48                 byte[] imageBytes =
49                     Base64.decode(item.getImageBase64(),
50                         Base64.NO_WRAP);
51
52                 Bitmap bitmap =
53                     BitmapFactory.decodeByteArray(imageBytes,
54                         0, imageBytes.length);
55
56                 holder.ivImage.setImageBitmap(bitmap);
57             } catch (IllegalArgumentException e) {
58                 Log.e("BASE64_ERROR", "Base64格式错误：" +
59                     e.getMessage());
60             }
61         }
62
63         holder.tvName.setText(item.getName());
64         holder.tvShares.setText(item.getShares().toString());
65         holder.tvPrice.setText(item.isListed() ?
66             String.format("%s BKC",
67             item.getPrice().toString()) : "——");
68     }
69 }
```

```
56
57 // 若从 BuyNFTs 进入则显示账户 ,
58 // 若从 MyNFTs 进入则显示售出状态
59
60     if (fromBuyNFTsActivity) {
61
62         String account = item.getAccountNumber() != null
63             ? item.getAccountNumber() : "";
64
65         int length = account.length();
66
67         // 紧凑模式: 前3 + 末3字符
68         String compactDisplay = (length >= 6)
69             ? account.substring(0, 3) + "..." +
70                 account.substring(length - 3)
71             : account; // 对短地址显示完整内容
72
73         holder.tvSaleStatus.setText(compactDisplay);
74     } else {
75
76         holder.tvSaleStatus.setText(item.isListed() ?
77             "Listed" : "Unlisted");
78         holder.tvSaleStatus.setTextColor(item.isListed()
79             ? 0xFF4CAF50 : 0xFFFF44336);
80     }
81
82     // 设置选中状态
83
84     holder.itemView.setSelected(position ==
85         lastSelectedPosition);
86
87     holder.itemView.setBackgroundResource(
88         position == lastSelectedPosition ?
89             R.drawable.custom_nft_selected_bg
90             : R.drawable.custom_nft_background
91     );
92
93 }
```

```
80         holder.itemView.setOnClickListener(v -> {
81             int prevSelected = lastSelectedPosition;
82             int currentPosition = holder.getAdapterPosition();
83
84             if (currentPosition == RecyclerView.NO_POSITION)
85                 return;
86
87             // 切换选中状态
88             if (lastSelectedPosition == currentPosition) {
89                 lastSelectedPosition = -1;
90             } else {
91                 lastSelectedPosition = currentPosition;
92             }
93
94             // 刷新新旧选中项
95             if (prevSelected != -1) {
96                 notifyDataSetChanged(prevSelected);
97             }
98             if (lastSelectedPosition != -1) {
99                 notifyDataSetChanged(lastSelectedPosition);
100            }
101        });
102
103     // 添加获取选中项的方法
104     public NFT getSelectedItem() {
105         return lastSelectedPosition != -1 ?
106             nftData.get(lastSelectedPosition) : null;
107     }
108
109     public int getSelectedPosition() {
```

```
109         return lastSelectedPosition;
110     }
111
112     public void clearSelection() {
113         if (lastSelectedPosition != -1) {
114             int prev = lastSelectedPosition;
115             lastSelectedPosition = -1;
116             notifyDataSetChanged(prev);
117         }
118     }
119
120     //返回 NFT 数据 的 数量
121     @Override
122     public int getItemCount() {
123         return nftData.size();
124     }
125 }
```

NFTAdapter.java 用于管理 NFT 数据与视图的绑定。它使用 nft\_holder.xml 来承载 NFT 信息，通过 onBindViewHolder 将具体 NFT 信息放入 nft\_holder。

从 MyNFTsActivity 和 BuyNFTsActivity 调用此类会有不同的操作，加载不同的视图信息。若是从 MyNFTsActivity 调用 NFTAdapter 的，会显示上架状态；若是从 BuyNFTsActivity 中调用的，会显示卖家 id。

NFT 选中状态由 lastSelectedPosition 记录。点击 NFT 后，nft\_holder 会切换背景颜色，表示选中了该 NFT。再次点击该 NFT 后其背景颜色还原。

## 7). dialog\_confirm.xml & dialog\_confirm\_unlist.xml

dialog\_confirm 有两个用处，一是 MyNFTsActivity 中的上架提示框，二是 BuyNFTsActivity 中的购买提示框。

dialog\_confirm\_unlist 用于让用户确认是否下架 NFT。

## 8). BuyNFTsActivity.java

“购买 NFT 界面”，用户可以在此选择交易市场上一定份额的 NFT 进行购买。整体代码与 MyNFTsActivity 相似。特色的点在于，此处的提示框 dialog\_confirm 会根据用户输入的份数，乘上单价，并加上 20BKC 的手续费（10BKC 为交易手续费，10BKC 为铸造手续费，会交给 NFT 铸造者，此机制可以在智能合约中更改）显示出来。默认会显示 1 份 NFT 的价格加上手续费。

## 9). net 文件夹中的 ABIUtils.java & RequestIdGenerator.java (前端对后端接口的适配)

本项目采用 JSON-RPC 作为应用程序接口，以实现前端与以太坊区块链后端的交互。JSON-RPC 是一种轻量级、无状态的远程过程调用(RPC)协议，被广泛应用于不同的以太坊客户端。由于该协议与具体的客户端实现无关，开发者可以使用不同的编程语言进行调用，使得系统具有更好的兼容性和灵活性。

前端发送 HTTP 请求调用后端智能合约时需遵循特定的 JSON-RPC 格式。请求通常包括请求体 (method、params、id、jsonrpc) 和请求参数 (如 from、to、data、value、gas 等)。其中，method 字段用于标识具体的 Web3 接口调用，例如 eth\_sendTransaction、eth\_call 和 eth\_getTransactionReceipt，

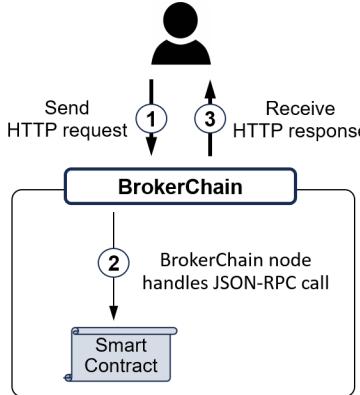


图 6.14: 前端系统交互图

不同的方法需要传入不同的请求参数。

`eth_sendTransaction` 用于发送交易并修改后端状态，其主要参数包括：`from`（用户地址）、`to`（智能合约地址）、`data`（包含智能合约函数调用信息及传入数据）、`value`（交易或 NFT 铸造费用）以及 `gas`（操作手续费）。

`eth_call` 负责查询后端信息而不对其进行修改，`data` 参数包含唯一标识符，以调用特定的后端函数。

`eth_getTransactionReceipt` 用于查询交易状态，所需参数为交易哈希值（`TransactionHash`）。查询后，后端返回 `status` 字段，`status=1` 表示交易成功，`status=0` 则表示交易失败。

由于 `BrokerFi` 本身已经拥有了 HTTP 请求的封装代码，这里仅需新增按照 JSON-RPC 格式要求格式进行二进制编码与解码的接口。这些接口封装在 `ABIUtils.java` 中。ABI (Application Binary Interface) 即应用二进制接口，负责智能合约的输入、输出数据的格式化，确保数据能够正

确传输并被合约识别。

ABIUtils 提供了多种编码函数，将输入信息转换为符合传输要求的二进制数据。例如，“encodeMint” 函数负责编码 NFT 铸造信息，而“encodeBuy” 函数负责编码购买信息。在这些编码过程中，代码内部使用了不同的静态函数选择器 (SELECTOR)，用于标识智能合约中不同的方法。例如，MINT\_SELECTOR 对应于智能合约的“铸造 NFT” 函数。

在实际使用中，每个编码函数都会将 SELECTOR 作为前缀附加到请求体数据上，使智能合约能够正确识别和执行对应的操作。例如，在“encodeMint” 函数编码 NFT 铸造数据时，会先对 NFT 的名称、图像信息及份额数据进行偏移量调整，然后与 MINT\_SELECTOR 组合，并返回编码后的数据。这部分数据会被封装到 OkhttpUtils 的“data” 字段中，并通过 doPost 请求发送至后端。

除了编码功能，ABIUtils 还实现了解码函数，用于解析智能合约返回的数据。其中包括 decodeGetMyNFTs (用于解码用户拥有的 NFT 信息) 以及 decodeGetMarketListings (用于解码市场上可售 NFT 信息)。这些解码函数首先解析数据的偏移量，随后进行数据解码，并将结果填充至多个结果数组中，以便前端的 NFTAAdapter 组件能够正确解析并更新 RecyclerView，最终在 MyNFTs 页面和 BuyNFTs 页面展示相应的 NFT 数据。

以下是具体代码详解：

```
1 public class ABIUtils {  
2     // 函数选择器（根据实际合约生成）  
3     private static final String MINT_SELECTOR = "d41d9a27";  
4     private static final String LIST_SELECTOR = "b0136c8d";  
5     private static final String UNLIST_SELECTOR = "961f0944";  
6     private static final String GET_MY_NFTS_SELECTOR =
```



并在 f12 中查看。

编码部分例子（以 encodeMint 为例）：

```
1 public static String encodeMint(String name, String
2     base64Image, BigInteger shares) {
3
4     // 计算动态参数偏移量
5     int baseOffset = 32 * 3; // 三个参数各占32字节
6     int nameContentLength = 32 + ((name.getBytes()).length +
7         31) / 32) * 32;
8
9     int imageOffset = baseOffset + nameContentLength;
10
11
12     StringBuilder data = new StringBuilder();
13     // 方法选择器
14     data.append(MINT_SELECTOR);
15     // 参数1：name偏移量
16     data.append(padLeft(Integer.toHexString(baseOffset), 64));
17     // 参数2：image偏移量
18     data.append(padLeft(Integer.toHexString(imageOffset),
19         64));
20     // 参数3：shares值
21     data.append(padLeft(shares.toString(16), 64));
22
23     // 动态参数1：name
24     data.append(encodeDynamicString(name));
25     data.append(encodeDynamicString(base64Image)); // 使用字符串编码方式
26
27     return "0x" + data.toString();
28 }
```

这是铸造信息的编码函数，根据 remix ide 发出的 HTTP 请求格式编写而

成。

具体案例：假设我要向后端发送一条 name 是 “hahaha”，image 是 “image”（测试用），shares 是 10 的铸造信息，那么我会在 MintActivity 中把这些信息传入 encodeMint()：

```
1 //生成交易数据 (使用 ABIUtils)  
2 BigInteger share = new BigInteger(sharesStr);  
3 String data = ABIUtils.encodeMint(nftname, base64Image,  
        share);
```

encodeMint 会生成以下数据：

```
1 0xd41d9a2700000000000000000000000000000000000000000000000000000000000000  
2 000000000060000000000000000000000000000000000000000000000000000000000000  
3 000000000000a0000000000000000000000000000000000000000000000000000000000  
4 0000000000000000a000000000000000000000000000000000000000000000000000000  
5 00000000000000000000668616861686100000000000000000000000000000000000000  
6 00000000000000000000000000000000000000000000000000000000000000000000000  
7 00000000000000000000000000000000000000000000000000000000000000000000000  
8 00000000000000000000000000000000000000000000000000000000000000000000000
```

对该数据的解读：

```
1 d41d9a27 mint 函数标识符  
2 000000000000000000000000000000000000000000000000000000000000000000000000  
    60 name 偏移量, 为 96  
3 000000000000000000000000000000000000000000000000000000000000000000000000  
    a0 image 偏移量, 为 160  
4 000000000000000000000000000000000000000000000000000000000000000000000000  
    0a shares 值, 为 10 (此处已经 96 位了, 接下来就是 name 信息)  
5 000000000000000000000000000000000000000000000000000000000000000000000000  
    06 name 长度: 6 字节 (name 是 string 类型, 长度不固定,
```

```
先带有信息长度, 后是内容)
6 68616861686100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00 name 内容: hahaha+填充
7 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
05 image 长度: 5字节
8 696d6167650000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00 image 内容: image 加填充
```

接下来, MintActivity 会把这些数据作为 data 放入完整的 HTTP 请求中:

```
1 //发送交易
2     sendMintTransaction(data, BigInteger.TEN);
```

请求封装成功后是以下内容:

```
1 {"method":"eth_sendTransaction","params": [
2 "from": "0x107D73D8A49EEB85d32Cf465507Dd71D507100C1",
3 "to": "0x1855c485845Aa12ab2dFbaB7c29aE2cC4669eF61",
4 "data": "0xd41d9a2700000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
7 000000000060000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
8 0000000000000a000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
9 000000000000000a0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
10 000000000000000000006686168616861000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
11 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
12 000000000000000000000000000000000000005696d616765000000000000000000000000000000000000000000000000000000000000000000000000000000000
13 00000000000000000000000000000000",
14 "value": "0x8",
15 "gas":
```

```
17 "0x1"}], "id":612, "jsonrpc":"2.0"}
```

如此，就可以通过 OKHTTPUtils 将请求发送到后端了。

```
1 public static String encodeList(BigInteger nftId, BigInteger
2     shares, BigInteger price) {
3
4     StringBuilder data = new StringBuilder();
5     data.append(LIST_SELECTOR);
6     data.append(padLeft(nftId.toString(16), 64));
7     data.append(padLeft(shares.toString(16), 64));
8     data.append(padLeft(price.toString(16), 64));
9
10
11    public static String encodeUnlist(BigInteger listingId) {
12
13        StringBuilder data = new StringBuilder();
14        data.append(UNLIST_SELECTOR);
15        data.append(padLeft(listingId.toString(16), 64));
16
17        return "0x" + data.toString();
18
19    public static String encodeBuy(BigInteger listingId,
20        BigInteger shares) {
21
22        StringBuilder data = new StringBuilder();
23        data.append(BUY_SELECTOR);
24        data.append(padLeft(listingId.toString(16), 64));
25        data.append(padLeft(shares.toString(16), 64));
26
27        return "0x" + data.toString();
28    }
```

```
27
28     public static String encodeGetMyNFTs() {
29         return "0x" + GET_MY_NFTS_SELECTOR;
30     }
31
32     public static String encodeGetListedNFTs() {
33         return "0x" + GET_LISTED_NFTS_SELECTOR;
34 }
```

同理，其他编码函数也是类似的操作。如果是修改后端数据的操作，data 字段就要包括操作数据，HTTP 请求要使用 eth\_sendTransaction 交互。如果是查看后端数据的操作，data 字段只需要放函数标识符，HTTP 请求要使用 eth\_call 交互。

由于 eth\_sendTransaction 的返回状态只会存在交易成功与否，所以不需要复杂的解码操作。eth\_sendTransaction 会返回交易哈希，再将交易哈希放入 HTTP 请求并使用 eth\_getTransactionReceipt 与后端交互，即可得知交易是否成功。（见 3.3）

eth\_call 则会返回比较复杂的数据，前端用到的两种 eth\_call 请求分别是获取用户持有的 NFT 信息以及市场上所有挂单 NFT 的信息，数据较为复杂，需要适配的解码操作。

解码部分例子（以 decodeGetMyNFTs 为例）：

```
1  public static class MyNFTsResult {
2      public BigInteger[] nftIds;
3      public String[] names;
4      public String[] images;
5      public BigInteger[] sharesList;
6      public BigInteger[] pricesList;
7      public boolean[] listingStatus;
```

```
8     public BigInteger[] listingIds;  
9 }
```

这里首先声明了一个类，用于装解码结果。

具体例子，假设前端 MyNFTsActivity 调用了 fetchMyNFTs，向后端获取用户持有的 NFT，后端返回以下内容：

```
1 { "jsonrpc": "2.0", "id": 906, "result":  
2 "0x0000000000000000000000000000000000000000000000000000000000000000  
3 000e00000000000000000000000000000000000000000000000000000000000000000000  
4 00001200000000000000000000000000000000000000000000000000000000000000000000  
5 0000001a00000000000000000000000000000000000000000000000000000000000000000000  
6 00000000022000000000000000000000000000000000000000000000000000000000000000000000  
7 0000000000026000000000000000000000000000000000000000000000000000000000000000000000  
8 00000000000002a0000000000000000000000000000000000000000000000000000000000000000000  
9 0000000000000002e00000000000000000000000000000000000000000000000000000000000000000  
10 000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
11 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
12 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
13 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
14 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
15 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
16 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
17 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
18 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
19 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
20 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
21 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
22 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
23 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
24 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
25 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
26 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
27 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000" }
```

MyNFTsActivity 会把 result 中的十六进制数据交给 decodeGetMyNFTs():

```
1 String hexData = response.getString("result");  
2 ABIUtils.MyNFTsResult nfts =  
3     ABIUtils.decodeGetMyNFTs(hexData);
```

对该数据的解读：

```
1 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    e0 → 数组1偏移量 (224字节)  
2 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    20 → 数组2偏移量 (288字节)  
3 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    a0 → 数组3偏移量 (416字节)  
4 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    20 → 数组4偏移量 (544字节)  
5 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    60 → 数组5偏移量 (608字节)  
6 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    a0 → 数组6偏移量 (672字节)  
7 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    e0 → 数组7偏移量 (736字节)  
8  
9 Nftid 内容：  
10 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000  
    01 → 数组长度 (1)  
11 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```





```
6     nfts.images[i], // 直接存储Base64图片数据
7     nfts.names[i],
8     nfts.sharesList[i],
9     nfts.pricesList[i],
10    nfts.listingStatus[i],
11    nfts.listingIds[i]
12  );
13  nftList.add(nft);
14 }
```

最后是 RequestIdGenerator，是一个全局可用的自增 Id 生成器，用于确保各个 HTTP 请求的 Id 不重复：

```
1 public class RequestIdGenerator {
2     private static AtomicInteger counter = new
3         AtomicInteger(1);
4     public static int getNextId() {
5         return counter.getAndIncrement();
6     }
7 }
```

## 6.2.6 后端 NFT 智能合约代码说明

### 1). 信息存储

```
1 contract NFT{
2     // NFT基础信息（所有持有者共享）
3     struct NFTMetadata {
4         uint256 nftId;
5         string name;
6         string image;
```

```
7         uint256 totalShares; //NFT的总份额，不会被更改
8         address minter; // NFT铸造者，后期可收取交易手续费
9     }
10
11    // 用户基础持有信息
12    struct UserHolding {
13        uint256 totalShares; //用户拥有的份额
14        uint256 availableShares; //未上架的份额
15    }
16
17    // 上架订单结构
18    struct Listing {
19        uint256 listingId;
20        uint256 nftId;
21        address payable seller;
22        uint256 shares;
23        uint256 price;
24    }
25    .....
26 }
```

合约使用了 NFTMetadata、UserHolding 和 Listing 三个结构体，分别用于存储 NFT 的元数据、用户的持有信息以及市场上的上架记录。

NFTMetadata 存储 NFT 的基础信息，包括 nftId (NFT 的唯一 ID)、name (NFT 名称)、image (NFT 图片数据)、totalShares (NFT 总份额) 以及 minter (铸造者 ID)。这些信息在 NFT 铸造时确定，之后不可更改，并由所有用户共享。

UserHolding 管理用户的 NFT 持有情况，每条记录的唯一标识是 nftId 和 userId 的组合，确保每个用户对每个 NFT 的持有信息都能被唯一存储

和快速查询。UserHolding 结构体包含两个字段：totalShares 记录用户对某个 NFT 的总持有份额，availableShares 记录用户未上架的份额。

Listing 用于管理市场上的 NFT 上架信息，包括 listingId（上架 ID）、nftId（NFT ID）、seller（卖家 ID）、shares（出售的 NFT 份额）和 price（NFT 份额单价）。为了减少冗余数据，合约在 NFT 上架时会检查是否已有相同卖家、相同价格的相同 NFT 在市场中。如果存在，合约不会创建新的 Listing，而是直接增加原 Listing 的 shares 数量，从而减少市场信息的重复存储。

```
1 // 计数器
2 uint256 private nftIdCounter;
3 uint256 private listingIdCounter;
4
5 mapping(uint256 => NFTMetadata) public nftMetadata; //  
    NFT基础信息
6 mapping(uint256 => Listing) public listings; // 所有上架订单
7 mapping(uint256 => mapping(address => UserHolding)) public  
    userHolding; // 所有用户持有详情
8
9 // 索引映射 便于用户查询
10 mapping(address => uint256[]) private userOwnedNFTs; //  
    用户地址 => 持有的 NFT ID数组
11 mapping(address => uint256[]) private userListings; //  
    用户的上架订单
```

在三个主要结构体之外，两个计数器（nftIdCounter、listingIdCounter）用于生成唯一 ID，两个索引信息（userOwnedNFTs、userListings）用于快速定位用户资产和挂单，避免全量遍历，提升查询效率。

## 2). 核心函数

```
1 //铸造NFT 返回铸造的NFT的Id
2 function mint(string calldata name, string calldata image,
   uint256 totalShares) external payable returns(uint256) {
3
4     uint256 gas = 10; //假设铸造费用为10
5
6     //检查用户是否有足够多的钱
7     require(msg.value >= gas, "Insufficient balance for
   minting");
8
9     nftIdCounter++;
10    nftMetadata[nftIdCounter] =
11        NFTMetadata(nftIdCounter, name, image, totalShares, msg.
   sender);
12    userHolding[nftIdCounter][msg.sender] =
13        UserHolding(totalShares, totalShares);
14    userOwnedNFTs[msg.sender].push(nftIdCounter);
15    return nftIdCounter;
16 }
```

**铸造 (mint)** 本方法用于铸造 NFT，调用时需支付 10 BKC 的 Gas 费用，合约会检查用户余额是否足够。铸造成功后，系统通过自增计数器 `nftIdCounter` 生成新的 NFT ID。这种顺序生成方式借助智能合约的上链机制保障唯一性，无需额外哈希操作，简化了 NFT 的管理与索引。随后，系统将元数据存入 `NFTMetadata`，并在 `UserHolding` 中初始化持有信息，记录用户份额。

```
1 //返回用户拥有的NFT的所有信息（包括上架的和未上架的）
```

```
2 function getMyNFTs() external view returns (
3     uint256[] memory nftIds,
4     string[] memory names,
5     string[] memory images,
6     uint256[] memory sharesList,
7     uint256[] memory pricesList,
8     bool[] memory listingStatus,
9     uint256[] memory listingIds
10 ) {
11     address user = msg.sender;
12
13     // 第一阶段：预加载元数据到结构体数组
14     NFTMetadata[] memory metadataCache = new
15         NFTMetadata[](userOwnedNFTs[user].length);
16
17     uint256[] storage ownedNFTIds = userOwnedNFTs[user];
18
19     for(uint i = 0; i < ownedNFTIds.length; i++) {
20         uint256 nftId = ownedNFTIds[i];
21
22         metadataCache[i] = NFTMetadata({
23             nftId: nftId,
24             name: nftMetadata[nftId].name,
25             image: nftMetadata[nftId].image,
26             totalShares: nftMetadata[nftId].totalShares,
27             minter: nftMetadata[nftId].minter
28         });
29     }
30
31     // 第二阶段：计算总条目数
32     uint256 totalItems = 0;
33
34     // 1. 统计有效挂单数
```

```
32     for(uint i = 0; i < userListings[user].length; i++) {
33         if(listings[userListings[user][i]].shares > 0) {
34             totalItems++;
35         }
36     }
37
38     // 2. 统计未上架数
39     for(uint i = 0; i < ownedNFTIds.length; ) {
40         uint256 nftId = ownedNFTIds[i];
41
42         if(userHolding[nftId][user].availableShares > 0) {
43             totalItems++;
44         }
45
46         unchecked { i++; }
47     }
48
49
50     // 第三阶段：填充数据
51     nftIds = new uint256[](totalItems);
52     names = new string[](totalItems);
53     images = new string[](totalItems);
54     sharesList = new uint256[](totalItems);
55     pricesList = new uint256[](totalItems);
56     listingStatus = new bool[](totalItems);
57     listingIds = new uint256[](totalItems);
58
59     uint256 index = 0;
60
61     // A. 处理挂单
62     for(uint i = 0; i < userListings[user].length; i++) {
```

```
63     Listing storage l = listings[userListings[user][i]];
64     if(l.shares == 0) continue;
65
66     // 从缓存数组查找元数据
67     NFTMetadata memory meta;
68     for(uint j = 0; j < metadataCache.length; j++) {
69         if(metadataCache[j].nftId == l.nftId) {
70             meta = metadataCache[j];
71             break;
72         }
73     }
74
75     nftIds[index] = l.nftId;
76     names[index] = meta.name;
77     images[index] = meta.image;
78     sharesList[index] = l.shares;
79     pricesList[index] = l.price;
80     listingStatus[index] = true;
81     listingIds[index] = l.listingId;
82     index++;
83 }
84
85 // B. 处理未上架
86 for(uint i = 0; i < ownedNFTIds.length; ) {
87     uint256 nftId = ownedNFTIds[i];
88
89     if(userHolding[nftId][user].availableShares > 0) {
90         NFTMetadata memory meta = metadataCache[i];
91
92         nftIds[index] = nftId;
93         names[index] = meta.name;
```

```
94         images[index] = meta.image;
95         sharesList[index] =
96             userHolding[nftId][user].availableShares;
97         pricesList[index] = 0;
98         listingStatus[index] = false;
99         listingIds[index] = 0; //标识未上架
100        index++;
101    }
102    unchecked { i++; }
103 }
104 }
```

**查询当前用户的 NFT 资产 (getMyNFTs)** 本方法用于获取用户当前持有的所有 NFT 信息。系统首先读取该用户拥有的所有 NFT ID，并提取对应的元数据；随后分别统计用户的上架订单和未上架份额，并将这些数据整合返回。返回的数据结构与前端展示一一对应，每个数组单元包含图片、名称、单价、持有份额和上架状态等信息，其中不同的上架订单会作为独立条目返回。未上架部分则作为价格为 0、份额为 availableShares 的数组项，并标记为“未上架”。该方法实现了持仓信息与市场挂单的同时查询，便于用户一次性获取完整的数据。

```
1 function listNFT(uint256 nftId,uint256 shares,uint256 price)
2     external {
3         require(userHolding[nftId][msg.sender].availableShares >=
4             shares, "Insufficient available shares");
5         // 检查是否已有相同价格的挂单，有则合并
6         bool merged = false;
```

```
6     for(uint i=0; i<userListings[msg.sender].length; i++){
7         Listing storage existing =
8             listings[userListings[msg.sender][i]];
9         if(existing.nftId == nftId && existing.price == price
10            && existing.seller == msg.sender) {
11             existing.shares += shares;
12             merged = true;
13             break;
14         }
15     }
16     // 创建新订单
17     if(!merged) {
18         listingIdCounter++;
19         listings[listingIdCounter] = Listing({
20             listingId: listingIdCounter,
21             nftId: nftId,
22             seller: payable(msg.sender),
23             shares: shares,
24             price: price
25         });
26         userListings[msg.sender].push(listingIdCounter);
27     }
28     // 更新持有量
29     userHolding[nftId][msg.sender].availableShares -= shares;
30 }
```

上架（listNFT） 用户可通过本方法将所持 NFT 的部分份额上架市场。执行前，合约会检查其可用份额（availableShares）是否充足，并遍历

该用户现有上架订单，尝试合并相同价格的订单，以减少重复与存储开销。若未找到相同价格订单，则生成新的 Listing 结构体并写入映射，同时更新索引信息。上架成功后，用户的可用份额将减少相应数量，以反映其挂售状态。

```
1 function unlistNFT(uint256 listingId) external {
2     Listing storage listing = listings[listingId];
3     require(listing.seller == msg.sender, "Not owner");
4
5     // 恢复持有量
6     userHolding[listing.nftId][msg.sender].availableShares += listing.shares;
7
8     // 从用户挂单列表移除
9     removeFromArray(userListings[msg.sender], listingId);
10    delete listings[listingId];
11 }
```

**下架 (unlistNFT)** 用户可通过本方法撤回已上架的 NFT 份额。合约会先确认调用者是否为订单的创建者，确认无误后，会将该订单中的 NFT 份额退回至用户的 availableShares，并删除 Listing 中的相关数据，确保市场不会存留无效挂单。

```
1 function getAllMarketListings() external view returns (
2     address[] memory sellers,
3     uint256[] memory nftIds,
4     string[] memory names,
5     string[] memory images,
6     uint256[] memory sharesList,
7     uint256[] memory pricesList,
```

```
8     uint256[] memory listingIds
9 ) {
10    // 先计算有效挂单数量
11    uint validCount = 0;
12    for (uint listingId = 1; listingId <= listingIdCounter;
13        listingId++) {
14        if (isValidListing(listingId)) {
15            validCount++;
16        }
17    }
18    // 初始化返回数组
19    sellers = new address[](validCount);
20    nftIds = new uint256[](validCount);
21    names = new string[](validCount);
22    images = new string[](validCount);
23    sharesList = new uint256[](validCount);
24    pricesList = new uint256[](validCount);
25    listingIds = new uint256[](validCount);
26
27    // 阶段2: 填充有效数据
28    uint index = 0;
29    for (uint listingId = 1; listingId <= listingIdCounter;
30        listingId++) {
31        if (isValidListing(listingId)) {
32            Listing storage l = listings[listingId];
33            NFTMetadata storage meta = nftMetadata[l.nftId];
34
35            sellers[index] = l.seller;
36            nftIds[index] = l.nftId;
37            names[index] = meta.name;
38        }
39    }
40}
```

```
37         images[index] = meta.image;
38         sharesList[index] = l.shares;
39         pricesList[index] = l.price;
40         listingIds[index] = listingId;
41         index++;
42     }
43 }
44 }
```

**查询市场挂单信息（`getAllMarketListings`）** 本方法用于获取市场中除当前用户自身挂单以外的所有有效 NFT 挂单。系统会遍历所有 Listing 信息，通过 `isValidListing` 方法筛选出仍有可售份额、卖家地址有效且非当前调用者的订单，避免返回无效或过期数据。统计有效订单数量后，方法会初始化返回数组并填充每一条订单的详细信息，确保用户获取的是最新且准确的市场挂单情况。该方法的返回数据结构与 `getMyNFTs` 类似，便于前端视图的统一展示。

```
1 function buyNFT(uint256 listingId, uint256 shares) external
2     payable {
3     Listing storage listingNFT = listings[listingId];
4     require(listingNFT.seller != msg.sender, "Cannot purchase
5         your own listing");
6     require(listingNFT.shares >= shares, "Insufficient
7         shares");
8
9     uint256 gas = 10; //假设购买的 Gas 费为 10
10    uint256 royaltyFee = 10; //假设支付给铸造者的手续费为 10
11    uint256 NFTCost = listingNFT.price * shares;
12    require(msg.value >= NFTCost + gas + royaltyFee ,
13        "Insufficient ETH");
```

```
10
11     address minter = nftMetadata[listingNFT.nftId].minter;
12
13     payable(minter).transfer(royaltyFee);
14     payable(listingNFT.seller).transfer(NFTCost);
15
16     // 更新状态
17     listingNFT.shares -= shares;
18     updateOwnership(msg.sender, listingNFT.nftId,
19                     int256(shares));
20     updateOwnership(listingNFT.seller, listingNFT.nftId,
21                     -int256(shares));
22
23     if (listingNFT.shares == 0) {
24         removeFromArray(userListings[listingNFT.seller],
25                         listingId);
26         delete listings[listingId];
27     }
28 }
```

**购买 NFT (buyNFT)** 买家可以通过本方法购买 NFT，只需输入目标挂单及希望购买的 NFT 份额。合约首先验证挂单中剩余份额是否足够，然后计算总费用：包括 NFT 单价乘以购买份数、10 BKC 的铸造者版税，以及 10 BKC 的 Gas 费用。若支付成功，合约会将版税转给 NFT 铸造者，将 NFT 售价支付给卖家，同时更新买家与卖家的 UserHolding，记录最新的份额分配情况。如果该订单的 NFT 份额被全部购买，系统会自动从 Listing 中清除该订单，确保市场数据的准确。合约通过 updateOwnership 方法管理 NFT 持有关系。对于买家，系统会增加其 totalShares 和 availableShares；

对于卖家，则减少其 `totalShares`，从而实现所有权转换。若卖家该 NFT 份额被完全售出，则会删除其 `UserHolding` 中对应记录。

## 6.3 DePIN-RWA：算力设备代币化

Slogan: 让闲散算力代币化。

读者可以从 Github 下载安装 DePIN-RWA 的 dAPP<sup>1</sup>到安卓手机上。

具体操作步骤可在线查看演示视频<sup>2</sup>。

此 dApp 仅为一个 Toy Example，可由第6.2章节展示的 NFT-Fi 的 dApp 轻松派生得到。欢迎读者（尤其是研究者）“魔改”这个 toy example。

---

<sup>1</sup><https://github.com/HuangLab-SYSU/DePIN-RWA-academic>

<sup>2</sup><https://j4s9dl19cd.feishu.cn/docx/XCh9d24Nv0thw4xFunac64DAnDe>



## 参考文献

- [1] Qinde Chen, Huawei Huang, Zhaokang Yin, Guang Ye, and Qinglin Yang. Broker2Earn: Towards Maximizing Broker Revenue and System Liquidity for Sharded Blockchains. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, pages 251–260. IEEE, 2024.
- [2] Huawei Huang, Yue Lin, and Zibin Zheng. Account Migration across Blockchain Shards using Fine-tuned Lock Mechanism. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, pages 271–280. IEEE, 2024.
- [3] Huawei Huang, Xiaowen Peng, Jianzhou Zhan, Shenyang Zhang, Yue Lin, Zibin Zheng, and Song Guo. BrokerChain: A Cross-Shard Blockchain Protocol for Account/Balance-based State Sharding. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, pages 1968–1977. IEEE, 2022.
- [4] Huawei Huang, Guang Ye, Qinglin Yang, Qinde Chen, Zhaokang Yin, Xiaofei Luo, Jianru Lin, Jian Zheng, Taotao Li, and Zibin Zheng. BlockEmulator: An Emulator Enabling to Test Blockchain Sharding Protocols. *IEEE Transactions on Services Computing (TSC)*, 18(2):690–703, 2025.

- [5] Huawei Huang, Zhaokang Yin, Qinde Chen, Guang Ye, Xiaowen Peng, Yue Lin, Zibin Zheng, and Song Guo. BrokerChain: A Blockchain Sharding Protocol by Exploiting Broker Accounts. *IEEE/ACM Transactions On Networking (ToN)*, pages 1–16, 2025.
- [6] Jiaping Wang and Hao Wang. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. In *Proc. of 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, volume 2019, pages 95–112, 2019.
- [7] Jian Zheng, Qinde Chen, Chunhua Su, and Huawei Huang. Brokerfi: A defi dapp built upon broker-based blockchain. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1817–1825, 2023.
- [8] Jian Zheng, Huawei Huang, Yinqiu Liu, Taotao Li, Hong-Ning Dai, and Zibin Zheng. Justitia: An Incentive Mechanism towards the Fairness of Cross-shard Transactions. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, pages 1–10. IEEE, 2025.