NFT Market 项目面试指南

> 前置：
>
> 1. 对NFT及NFT Market有一定的了解；
> 2. 最好看完社区内和NFT Market相关的录播课程，包括项目架构分享，核心模块功能及源码讲解。

零、基础

1.NFT基本概念

特性	ERC20	ERC721	ERC1155
类型	同质化代币	非同质化代币	可同时支持同质化
用途	货币、奖励代币、稳定币等	数字收藏品、NFT、游戏资产等	混合资产模型，支
代币的唯一性	代币之间没有区别，每个代币都是相同的	每个代币都是独特的、不可替代的	同时支持同质化和
代币ID	没有代币ID，每个代币相同	**每个代币都有唯一的ID**	不同ID的代币可以
批量转移	不支持批量转移，需逐个发送	**不支持批量转移，需逐个发送**	支持批量转移，减
适用场景	同质化资产，如货币、投票权	数字艺术品、NFT、唯一资产	游戏资产、数字商
数据结构复杂性	简单，存储每个地址的代币余额	**复杂，每个代币有独立的元数据**	复杂，支持多种类
转移成本	低	较高，因为每个代币都是唯一的	低，可以同时转移
标准接口函数	balanceOf, transfer, approve等	ownerOf, transferFrom, approve等	balanceOf, safeTr
智能合约交互	简单，同质化	复杂，需要处理每个代币的唯一性	相对复杂，但提供

2.NFT的核心操作

如ERC721，详见：https://eips.ethereum.org/EIPS/eip-721

代码块

```
// 1 关于查询（view）

// 查询_owner拥有TokenId的数量
function balanceOf(address _owner) external view returns (uint256);
// 查询_tokenId的拥有者
function ownerOf(uint256 _tokenId) external view returns (address);
// （可选，非必须） 查询NFT集合的name和symbol
function name() external view returns (string _name);
function symbol() external view returns (string _symbol);
```

```
    // （可选，非必须） 查询指定NFT的token uri信息
    function tokenURI(uint256 _tokenId) external view returns (string);

    // 2 关于授权（approve）

    // msg.sender授权_tokenId的操作权限给_approved地址
    function approve(address _approved, uint256 _tokenId) external payable;
    // msg.sender授权(或取消授权)其所有权限给_operator地址
    function setApprovalForAll(address _operator, bool _approved) external;
    // 查询单个_tokenId的授权情况，针对approve(address,uint256)方法
    function getApproved(uint256 _tokenId) external view returns (address);
    // 查询拥有者_owner地址的授权情况,针对setApprovalForAll方法
    function isApprovedForAll(address _owner, address _operator) external view
    returns (bool);

    // 3 关于转移(transfer)

    // 将_tokenId从_from地址转移至_to地址;附加校验_to地址是否为合约地址; 附加额外数据data;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes
    data) external payable;
    // 将_tokenId从_from地址转移至_to地址;附加额外校验, 一般为onERC721Received;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId)
    external payable;
    // 将_tokenId从_from地址转移至_to地址;
    function transferFrom(address _from, address _to, uint256 _tokenId) external
    payable;
```

关于 NFT（以 ERC-721 为例）的一些基础认知：

1. **NFT 资产的核心是所有权（Ownership）**，主要通过 ownerOf 和 balanceOf 方法体现：

- ownerOf(tokenId)：查询指定 tokenId 的当前拥有者地址；
- balanceOf(address)：查询指定地址当前拥有的 NFT 数量。

2. **关于 NFT 的 name、symbol 和 tokenURI 等元信息（Metadata）字段并非强制要求实现**，即在 ERC-721 标准中它们是 *可选的* 扩展接口（如 ERC721Metadata）。
3. **有些同学存在误解，认为 NFT 交易的是 tokenURI，这是错误的**。实际交易的是 **NFT 的所有权**，也就是 ownerOf(tokenId) 返回的地址发生了变化。
4. 尽管不是强制要求，但在实际项目中，**绝大多数 NFT 合约都会实现 name、symbol 和 tokenURI**：

- name 和 symbol 通常用于标识整个 NFT 合集（Collection）；
- tokenURI(tokenId) 则为每一个 tokenId 提供唯一的元数据链接（通常用于指向链下图片、属性等信息）。

5. **不同的 NFT 合约可以定义相同的 name、symbol，甚至使用相同的 tokenURI**。因此在进行 NFT 交易时，**务必要确认该合约是否为“官方发行”的版本**，即确保合约地址（Contract Address, CA）的唯一性与可信性。
6. **明确一点：NFT 的交易本质是所有权的转移。**围绕所有权的管理，NFT 提供了两种核心操作：

- **授权（Approval）**
- **转移（Transfer）**

7. **授权（Approve）**：指 NFT 的当前拥有者允许其他地址操作自己持有的 NFT，可以通过以下两种方式授权：

- **approve**(address to, uint256 tokenId)：授权某个地址对**特定 tokenId** 进行转移；
- **setApprovalForAll**(address operator, bool approved){msg.sender}：授权某个地址可操作这个NFT合约下**所有 tokenId**，适用于如交易市场、批量管理等场景。

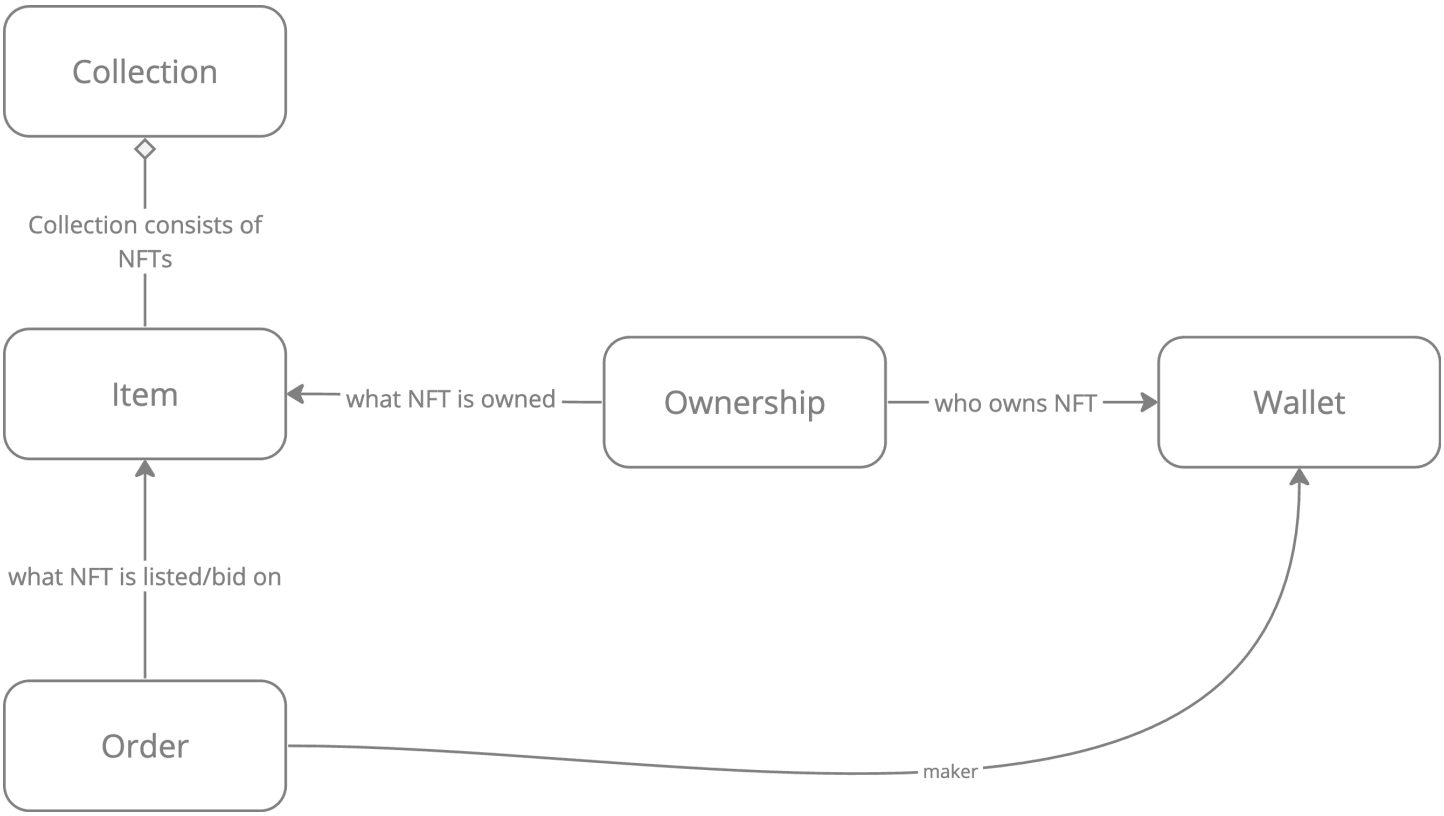
8. **转移（Transfer）**：是指将某个 tokenId 从一个地址转移到另一个地址。ERC-721 中提供了两种常用的转移函数：

- transferFrom(from, to, tokenId)：最基础的转移方法，通常由授权地址或拥有者调用；
- safeTransferFrom(from, to, tokenId)：在转移的同时进行接收方合约检查，避免 NFT 被转入不支持 ERC-721 的合约地址中。

9. **实际交易流程**（以 NFT 市场为例）：

- 用户首先调用 approve 授权给NFT市场合约（即 Market 的 CA）；
- 市场合约在完成交易撮合后，调用 transferFrom 或 safeTransferFrom 将 NFT 从卖家地址转移至买家地址；
- 该过程不涉及 tokenURI 的变化，**唯一变化的是 NFT 的所有权地址**。

3.NFT数据模型



- **Collection（集合）** ——— 一组 NFT 的集合，一个 Collection 下可以包含多个 NFT（Item）；
- **Item（NFT）** ——— 每一个具体的 NFT, 属于某个 Collection, 是NFT交易系统中的基本单位；
- **Ownership（所有权）** ——— 代表NFT的所有权，某个钱包地址对某个 Item 的持有关系，也就是Item的Owner；
- **Wallet（钱包）** ——— 表示用户的钱包地址，NFT 的最终所有者；可作为 Order 的发起者（Maker）
- **Order（订单）** ——— 表示用户希望出售或购买 NFT 的意愿，可包含出价、购买或挂单信息；
- **Activity（活动）** ——— 表示在NFT交易系统上发生的与 NFT 相关的事件记录，用于追踪行为历史，包含：transfer（转移）, list（挂单）, buy（出售）等

4.NFT交易模式

在 NFT（非同质化代币）交易中，主要存在以下两种订单发布与撮合方式：

10. **链下订单（Off-chain Order Book）**

- **模式特点**：用户在**链下平台**（如**中心化服务或链下订单簿**系统）创建和管理 NFT 挂单订单，订单的签名由用户签署并保存在服务器或数据库中。
- **撮合方式**：当 Taker 想要成交订单时，将订单与签名一并提交至链上智能合约，由合约验证签名合法性并完成交易。
- **优势**：省 Gas（挂单不发交易）、撮合灵活、适合高频交易。

11. **链上订单（On-chain Order Book）**

- **模式特点**：用户将 NFT 挂单直接提交至链上智能合约，所有订单数据**保存在链上**，确保完全的**去中心化**与可验证性。
- **撮合方式**：任何人（包括协议撮合器）都可调用合约的撮合逻辑，实现订单匹配与成交。
- **优势**：不依赖中心化服务、去中心化强、透明度高、数据完整。

5.NFT交易机制

订单簿（Order Book）模型 （90%）

- **参与角色**：
  - **Maker**：创建订单的一方，通常是卖家或挂单方。
  - **Taker**：接受订单的一方，通常是买家。
- **定价机制**：由**订单**直接给出具体价格。
- **匹配方式**：一对一撮合，价格**由用户指定**。
- **优点**：灵活定价、可构建复杂交易策略。

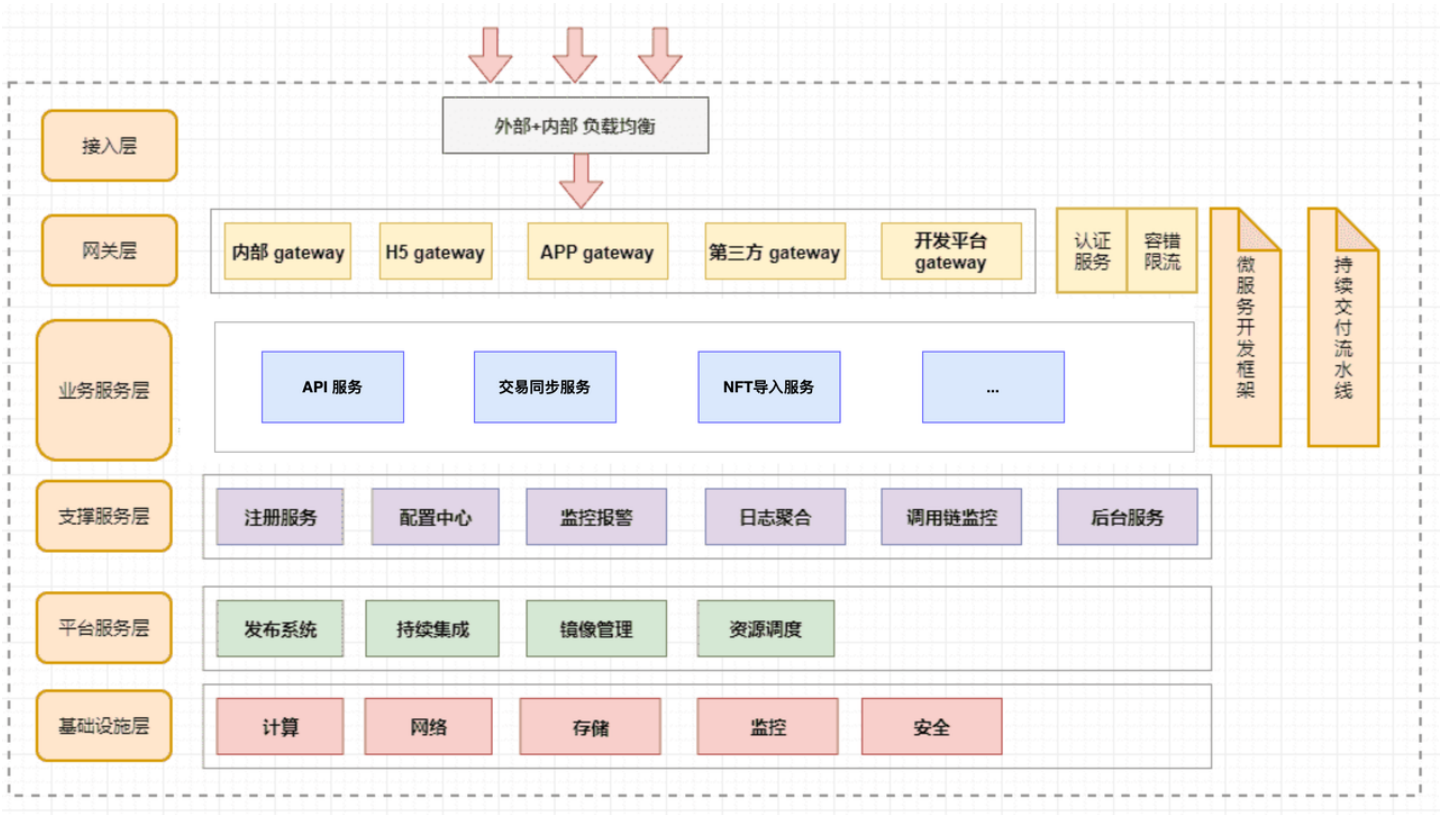
自动化做市商（AMM）模型 （10%）

- **参与角色**：
  - **Maker**：AMM 池子（由流动性提供者组成）。
  - **Taker**：用户与池子进行交易。
- **定价机制**：价格根据池子中资产数量通过**算法动态调整**（如 bonding curve）。
- **匹配方式**：用户与池子间进行一对池交易，价格由合约算法决定。
- **优点**：持续流动性高、无需等待挂单匹配。

6.NFT交易对比

模式类型	存储位置	撮合方式	定价方式	优势
链下订单簿	链下	提交链上成交	用户指定	成本低、灵活
链上订单簿	链上	链上撮合	用户指定	去中心化、透明
NFT-AMM 模式	链上	与池子交易	算法自动定价	高流动性、低等待

7.服务架构举例



1. **API服务：**为C端用户提供核心的API接口，支持用户在平台上进行 NFT信息查看，NFT交易（挂单， 购买，转移）等操作。通过高效、安全的API接口，确保用户与平台的数据交互顺畅，并支持高并发的请求处理。
2. **交易同步服务：**实时监控并同步区块链上的NFT Market交易合约事件，捕获包括MatchOrder，CancelOrder等关键事件。同步后的数据将被及时更新到链下的数据库中，以保持链上和链下数据的一致性。通过处理交易回执和事件日志，确保每笔交易的完整性和准确性，为后续的查询、分析提供可靠的数据支撑。
3. **NFT导入服务**：定期从第三方API（如Alchemy，Moralis，NFTSCAN API等）获取NFT相关的链上和链下数据，包括**NFT的元数据、所有权、媒体等信息**，并将其导入到**本地数据库**中，方便后续的**搜索和展示**。同时，服务会处理和验证获取的数据，确保数据的有效性和完整性，并支持NFT集合的批量导入与更新。
4. **订单中继服务：**通过与其他交易平台（如LooksRare、X2Y2等）的API进行交互，获取跨平台的订单数据，包括买卖报价、成交记录等信息。通过订单中继服务，平台能够将其他平台的订单信息进行聚合展示，扩大用户的选择范围，并为平台提供更多的交易流动性支持。
5. **定时任务模块：**负责定期执行平台内的各类数据更新任务，包括计算和更新Collection Rank（NFT合集排名）、历史交易量统计、活跃用户分析等。通过灵活配置的定时任务机制，确保平台的统计数据和排行榜信息保持最新，支持用户进行有效的决策和判断。

1.NFT基本概念

特性	ERC20	ERC721
类型	同质化代币	非同质化代币
用途	货币、奖励代币、稳定币等	数字收藏品、NFT、游戏资产等
代币的唯一性	代币之间没有区别，每个代币都是相同的	每个代币都是独特的、不可替代的
代币ID	没有代币ID，每个代币相同	**每个代币都有唯一的ID**
批量转移	不支持批量转移，需逐个发送	**不支持批量转移，需逐个发送**
适用场景	同质化资产，如货币、投票权	数字艺术品、NFT、唯一资产
数据结构复杂性	简单，存储每个地址的代币余额	**复杂，每个代币有独立的元数据**
转移成本	低	较高，因为每个代币都是唯一的
标准接口函数	balanceOf, transfer, approve等	ownerOf, transferFrom, approve等
智能合约交互	简单，同质化	复杂，需要处理每个代币的唯一性

> **面试准则：**
>
> 1. 以下“面试回答”仅供参考；
> 2. 不局限于社区内项目，在“项目基础”及“面试回答”上增加自己的理解；
> 3. 面对不同岗位方向和要求，在回答中体现不同的重点；
> 4. 最后，提升并体现对行业的理解和表达；

一、项目通用理解与背景

1.⭐你能简单介绍一下这个 NFT 市场项目吗？它的核心功能有哪些？

项目描述: NFT Market项目是一个基于区块链的NFT**去中心化**交易平台，平台的核心功能包括用户能够方便地交易NFT（挂单、购买、转移），同时保证链上数据与链下数据的同步，提升用户的交易体验。

核心功能包括：

1. **链上订单簿 DEX**：挂单上链，任何人都可以撮合成交，形成公开透明的去中心化交易逻辑。
2. **链上数据同步服务**：后端服务监听链上事件（如 MatchOrder，CancelOrder），同步到数据库，为前端提供实时交易数据。
3. **RESTful API 服务**：对前端提供查询接口，支持 NFT 列表、详情、交易记录等功能。
4. ...

（补充问答）**针对去中心化的解释？**

1. **先解释合约的构成**：有两个核心合约：OrderBook合约和Vault合约，
    a. OrderBook合约：实现了完整的订单簿交易逻辑，包括多个模块，如
        i. OrderStorage: 用于**存储订单**信息的模块
        ii. OrderValidator: 用于处理**订单逻辑验证**的模块
        iii. ProtocolManager: 用于**管理协议费**的模块
        iv. ...（版税管理）
    b. **Vault合约：**实现了一个资金和 NFT 资产管理库，用于去中心化订单系统的**资产管理**。
2. **再回答：** 其中OrderBook的OrderStorage模块和Vault合约实现了订单信息存储及资产管理；由此实现 挂单（买单 or 卖单）上链，可以不通过链下中心化后端任何人都可以通过合约查询订单数据，完成撮合成交，形成公开透明的去中心化交易逻辑。

（补充问答）**链上订单 vs 链下订单?**

1. NFT订单在链下；maker授权并链下签名， taker检索链下订单并链上确认撮合，完成交易； 需要依赖链下服务，但成本低；
2. NFT订单在链上：maker链上记录订单并转移资产到Vault, taker检索订单并链上确认撮合，完成交易；可以不需要链下服务（纯dex）， 但成本高；

2.项目的整体架构是怎样的？链上和链下分别负责什么？

项目主要就是两大模块，链上和链下

1. **链上合约**实现了完整的核心去中心化交易逻辑： 有两个核心合约：OrderBook合约和Vault合约，
    a. OrderBook合约：实现了完整的订单簿交易逻辑，包括多个模块，如
        i. OrderStorage: 用于存储订单信息的模块
        ii. OrderValidator: 用于处理**订单逻辑验证**的模块
        iii. ProtocolManager: 用于**管理协议费**的模块
        iv. 。。。可以自己补充其他模块，如NFT版税管理
    b. **Vault合约：**实现了一个资金和 NFT 资产管理库，用于去中心化订单系统的**资产管理**。

2 **链下（Go 后端服务）：**

- **链上事件监听模块**：实时监听合约事件（如 MatchOrder，CancelOrder），并同步到数据库。
- **API 服务模块**：为前端提供核心的API接口，如 NFT数据，NFT交易（挂单， 购买，转移）等操作。
- **定时任务模块：**定期执行平台内的各类数据更新任务，包括计算和更新Collection Rank（NFT合集排名）、NFT集合地板价，历史交易量统计等析等
- 。。。

3.你在这个项目中主要负责哪些模块?

我负责了部分后端架构还有核心合约的设计和实现，具体包括：

- **链上数据同步服务**：包括合约事件监听、区块确认处理、重试机制等。
- **API 服务**：采用 RESTful 架构，支持分页、过滤、排序功能，满足 NFT 展示与交易查询需求。
- **性能优化与缓存设计**：使用 Redis 缓存热门 NFT 数据，提高高并发下的响应速度。
- **核心合约逻辑设计和实现**：包括 Trade 合约、NFT 合约的接口设计、事件定义、交易函数编写。
- 。。。

4.这个项目用了哪些技术栈？为什么选择它们?

Golang：性能优秀，适合处理链上数据同步和高并发 API 请求。

- **Gin + GORM**：快速开发 Web 接口和数据库 ORM 映射。
- **Redis**：用于缓存热点数据，减少数据库压力。
- **MySQL**：持久化存储链上同步数据、交易记录、用户挂单等。

Solidity：编写 NFT 和交易智能合约。

- **Ethers.js / Web3.js**：用于监听事件、调用合约。**
- **Hardhat / Foundry**：用于编写和测试合约。

5.⭐NFT在NFT市场中，NFT的全生命周期

> Mint（铸造）-> Approve（授权） -> List (上架) -> Buy(购买) /Withdraw（撤单）

一、NFT导入&展示

- 通过后端的NFT导入服务，将指定NFT集合的所有NFT相关的链下/链上数据存储至数据库；
  - RPC（Alchemy，Infura。。。），NFT第三方服务商API（Alchemy API，Moralis API, NFTSCAN...）交互
  - 直接调用链上合约方法`tokenURI(tokenId)`获取NFT Metadata数据；
- 前端请求后端API服务接口，获取NFT相关信息展示。

二、 **NFT上架**

- 用户（卖方）发起上架请求：前端基于NFT市场合约的挂单函数MakeOrder构造参数，并**前端直接发起交易**（这里发交易是不涉及后端）；
  - 挂单价格、有效期。。。

- **后端同步服务**监听挂单事件，并将链上订单信息写入数据库，前端请求API展示订单信息；

三、NFT被购买

- 其他用户（买方）浏览NFT， 发现可购买的NFT，并点击“购买”；
- 前端基于NFT市场合约的撮合函数MatchOrder构造参数，并直接发起交易；
- 合约：执行撮合逻辑，验证订单，资产转移。。
- **后端同步服务**监听撮合事件，并将链上订单信息写入数据库，前端请求API展示订单信息；

三、NFT挂单被撤销

- 用户并点击“取消订单”；
- 前端基于NFT市场合约的撮合函数CancelOrder构造参数，并直接发起交易；
- 合约：执行取消逻辑，验证订单，资产转移。。
- **后端同步服务**监听取消事件，并将链上订单信息写入数据库，前端请求API更新订单信息；

6. **项目数据量级**

1个Collection：>10000Item；

NFT Collection数量级：上千个集合。。。

NFT Item数量级： 千万/上亿

成交量：>1万ETH (2%手续费)

二、NFT 相关知识

1.什么是 NFT？它和 ERC20 有什么不同？

2. ERC721 和 ERC1155 的区别？

3.⭐NFT Market如何实现 NFT 的授权与交易？approve 和 safeTransferFrom 是怎么用的？

NFT 的授权与交易一般通过如下流程实现：

1. 授权 Approve：

- approve(to, tokenId)：授权某个地址可以转移你指定的某个 tokenId
- setApprovalForAll(operator, true)：授权某个地址可以操作你所有 NFT

2. 安全转账 safeTransferFrom：

- safeTransferFrom(from, to, tokenId)：推荐使用的 NFT 转移方式，会检查接收者是否实现了 onERC721Received 接口
- 可以防止 NFT 被误转到不支持接收的地址（如合约地址），避免资产丢失

在NFT Market项目中，采用的是**链上订单簿 + 撮合合约 + 资产托管 Vault**的架构，交易流程中并**不直接调用 approve 或 safeTransferFrom 来完成 NFT 的授权和转移**，而是引入了 **Vault 合约托管资产**来实现去中心化的交易机制。

Step 1 用户授权 Vault 合约: 当用户创建出售订单（List 订单）时，他们必须将 NFT 转移到平台的资产托管合约（Vault）。这就要求用户在**首次交易**前，通过前端或链下交互完成标准的授权流程。这一步是典型的 approve 用法，授权 Vault 代用户转移 NFT。

Step 2：用户创建订单（挂单）：

- 卖家发起 makeOrders()，合约会将 NFT 从卖家地址转移（transfer）到 Vault 合约。
- 买家发起 makeOrders()，会将 ETH 存入（transfer） Vault合约中。

Step 3：撮合订单（match）：当任意一方想要撮合时，调用 matchOrder(sellOrder, buyOrder)：

- 合约内部会校验两笔订单是否合法、价格匹配、未过期、未撮合等
- 然后从 Vault 中完成资产转移
  - ETH 从买家 -> 卖家（减去协议手续费）
  - NFT 从 Vault -> 买家

交易撮合gas谁出：谁结算谁出；买家撮合卖单-> 买家出gas；卖家撮合买单-> 卖家出gas；

三、智能合约实现与安全

1.你写合约项目时遵循哪些安全规范?

1. **使用 OpenZeppelin 审计过的合约模块：**使用 Ownable, Pausable, ReentrancyGuard, Initializable 等模块，避免重复造轮子，降低安全风险。
2. **避免重入攻击：**所有涉及 ETH 或 NFT 转移的逻辑都使用 nonReentrant 修饰，并在逻辑完成前**先修改状态再转账**。
3. **精确控制合约权限：**所有关键操作函数都添加 onlyOwner 或自定义权限控制。合约初始化逻辑使用 initializer 限制，防止被二次初始化。

4. **使用 Checks-Effects-Interactions 模式：**函数中先做校验（Checks），再更新状态（Effects），最后交互（Interactions），防止逻辑穿插导致异常行为。
5. **防止整数溢出/下溢：**Solidity ^0.8.x 默认启用了 SafeMath 内置检查，但仍遵守加减乘除时注意边界条件。
6. **明确函数入口与调用关系：**对 delegatecall 做保护限制，设置 onlyDelegateCall 修饰符防止直接调用内部逻辑函数
7. 。。。

2.⭐NFT Market合约中有哪些潜在的安全风险？你是怎么防范的？

风险点	防范策略
重入攻击	nonReentrant + 安全转账库 safeTransferETH
签名伪造/重放攻击	EIP-712 结构签名 + salt + orderHash 唯一标识
订单提前执行 / 前置攻击（frontrun）	订单必须签名 + 撮合前验证链上状态是否变更
授权滥用 / NFT 被盗转	使用 Vault 托管资产 + 用户主动授权，减少 approve
撮合不匹配 / 错配资产转账	matchOrder 做资产验证 + 订单匹配逻辑严格匹配（collection, tokenId）
委托调用被直接执行	使用 onlyDelegateCall 限制 matchOrderWithoutPayback 仅由 delegatecall 调用
退款逻辑疏漏 / ETH 被锁定	所有 ETH 相关操作有退款逻辑，例如 msg.value > costValue 时自动退款

3.你有做合约审计吗？如何测试合约逻辑是否安全？

主要有三步：

1. **内部代码审查与安全测试：**使用**Hardhat** 编写全覆盖的单元测试，包含：撮合正确性测试、异常路径测试（过期、无授权、余额不足等）、安全测试（重入攻击模拟）。
2. **使用工具进行静态分析：**如**Slither**工具，用于检测重入、死代码、不安全调用、权限缺失等问题。
3. **外部审计（如上线主网前）：**如果合约部署在正式环境，我们会交给第三方审计团队（如 Certik）进行专业审计，尤其是交易、资金流动、权限部分。

（补充）合约安全：

1. 代码是否有漏洞：重入、溢出。。
2. 中心化风险（高风险）：项目本身有额外权限处理用户资产。

4.有没有遇到过⭐合约升级⭐的问题？用的是哪种升级方式？

在项目中使用了**可升级合约（Upgradeable Contracts）**，具体采用的是：

- **UUPS 升级模式（通过 @openzeppelin/contracts-upgradeable）：**使用 Initializable, OwnableUpgradeable, UUPSUpgradeable 实现合约的可升级结构。核心合约如 NFT Market OrderBook 和 Vault 都支持 UUPS 升级方式。
- **遇到的问题：**必须注意 **状态变量布局不能变更顺序**，否则会造成数据错乱；升级后合约必须验证逻辑和 ABI 一致，避免代理调用失败；多个模块组合时（如 OrderValidator / OrderStorage）要保持继承结构一致；
- **解决方式：**规范使用 storage gap；在升级测试中使用 **Hardhat Upgrades 插件**进行升级；编写专门的升级脚本与测试脚本，确保状态保持一致。

代码块

```
export interface HardhatUpgrades {
  deployProxy: DeployFunction; // 部署可升级的代理合约
  upgradeProxy: UpgradeFunction; // 升级已有的代理合约到新实现
  validateImplementation: ValidateImplementationFunction; // 验证合约是否符合可升级标准
  validateUpgrade: ValidateUpgradeFunction; // 验证新实现合约是否可安全升级
  deployImplementation: DeployImplementationFunction; // 仅部署实现合约，不创建代理
  prepareUpgrade: PrepareUpgradeFunction; // 预部署新实现合约，为升级做准备

  deployBeacon: DeployBeaconFunction;
  deployBeaconProxy: DeployBeaconProxyFunction;
  upgradeBeacon: UpgradeBeaconFunction;

  deployProxyAdmin: DeployAdminFunction;
  forceImport: ForceImportFunction;
  silenceWarnings: typeof silenceWarnings;

  admin: {
    getInstance: GetInstanceFunction; // 获取 `ProxyAdmin` 实例
    changeProxyAdmin: ChangeAdminFunction; // 更换代理合约的管理员
    transferProxyAdminOwnership: TransferProxyAdminOwnershipFunction; // 转移 `ProxyAdmin` 所有权
  };

  erc1967: {
    getAdminAddress: (proxyAddress: string) => Promise<string>; // 获取代理合约的管理员地址
    getImplementationAddress: (proxyAddress: string) => Promise<string>; // 获取代理合约当前的实现合约地址
    getBeaconAddress: (proxyAddress: string) => Promise<string>;
  };

  beacon: {
```

```
30       getImplementationAddress: (beaconAddress: string) => Promise<string>;
31     };
32   }
```

5.你用过哪些合约开发工具？比如 Hardhat / Remix？为什么用这个？

在开发中主要使用：

- **Hardhat（主力工具）：**

插件丰富，支持 Ethers.js、Waffle、Hardhat Network、本地调试非常方便；

支持合约升级（@openzeppelin/hardhat-upgrades）；

编写测试逻辑简洁，配合 Ethers.js 调试链下逻辑很灵活；

内置本地链可以方便调试交易、Gas 费用等问题。

- **Remix：**

用于快速验证一些小合约逻辑；

部署测试时可视化效果好，新人友好，但不适合大型合约项目。

工具选择理由：

Hardhat 用于完整项目开发流程；Remix 用于交互测试、调 ABI/接口用例。

四、链上链下数据同步

1. 链上事件是如何同步到后端的？你用的是什么机制？

通过一个 Sync 服务模块，专门用于链上事件同步。通过**配置文件**中定义的同步的链，监听的链上合约合约地址等。整体区块交易流程为：采用**区块轮询**方式，定期调用 eth_getLogs **获取指定合约、指定事件的日志**；针对订单簿，我们同步以下事件：LogMake → 创建订单；LogCancel → 取消订单；LogMatch → 撮合订单。同步过程采用可恢复 goroutine + context.Context 管理生命周期；同步事件解析后落库，使用 GORM 管理 MySQL 数据库。

2. ⭐为什么选择“轮询+日志查询”的方式进行事件同步，而不是 WebSocket？它有什么优劣？

选择 **轮询+事件日志过滤（eth_getLogs）** 的方式，是出于以下考虑：

原因：

- WebSocket 通常适用于实时性要求高、事件量小的场景，但容易断链、重连、漏事件；

- eth_getLogs 支持 **事件批量拉取 + 精确过滤（按合约地址、topics、区块范围）**，适合大批量数据同步；
- 轮询方式更稳定，易于做确认机制、回滚处理。

方式	优点	缺点
WebSocket	实时性高	不稳定、易漏事件、维护成本高
eth_getLogs	稳定、可追溯、支持大数据同步	实时性稍差，需确认延迟

3. ⭐你是如何解决链重组和数据不一致问题的?

1. 延迟同步策略（初级）

- **核心思想**：避免同步那些还可能被链重组覆盖的“浅区块”。
- **实现方式**：仅同步“当前链头 - N”高度之前的区块（例如延迟 6 个区块）。
- **合理性说明**：多数主流区块链（如 Ethereum、BSC）中，6 个区块后发生重组的概率极低，属于默认的“最终确认”标准。
- **可配置项**：延迟区块数，可根据链稳定性动态调整。

比如： 最新区块号：100； 那么只同步100-N前的区块数据；

2. **幂等数据结构设计（中级）**

- **核心思想**：无论链上是否重复推送事件、或者链下重复处理，同一事件只能被处理一次。
- **实现方式**：
  - 所有事件数据使用 (txHash, logIndex) 作为唯一索引；
  - 数据入库或缓存前检查是否存在重复；

3. **区块哈希对比机制（中级）**

- **核心思想**：通过记录并周期性比对链上已同步区块的哈希，发现是否发生了重组。
- **实现方式**：
  - 维护一个最近 M 个区块（如 12 个区块）的blockNumber -> blockHash映射；
  - 每轮同步时，重新获取当前链上的这 M 个高度区块的哈希（eth_getBlockByNumber），与本地缓存做比对；
  - 如发现不一致，则说明发生了 Reorg，进入回滚流程。
- **可配置项**：重组区块数，考虑链上传播延迟与极端分叉情况。

4. **回滚机制与临时缓存（高级）**

- **核心思想**：对“尚未确认”的最新区块的数据不立即持久化，而是临时缓存，便于快速回滚。
- **实现方式**：
  - 将最新 finalityConfirmations（重组区块数） 个区块范围内的数据缓存在 Redis 等临时存储；
  - 若检测到 Reorg，只需回滚该部分缓存数据；
  - 超过确认范围的数据再正式写入数据库，视为“最终确定”。

4. 同步时如何防止“写一半失败”问题？如何确保事务原子性？

1. **同步进度延后更新，确保完整处理单区块数据**

- 这样做可以有效防止因异常中断而导致某个区块数据只写了一半、却被错误地标记为“已同步”。
- 同步进度的更新应作为数据库事务的一部分，与区块内事件写入保持一致性。

2. 通过唯一索引保障幂等性，避免重复写入

- 遇到网络抖动、重试机制触发等情况时，同一事件可能被重复处理；
- 唯一约束可防止重复写入，避免脏数据、脏业务行为（如重复积分、重复入账等）；
- 实现幂等性是分布式系统容错重试机制的关键。

3. **单笔交易的多表写入通过事务保障一致性**

- 例如：Activity 表记录事件数据 + Item 表更新 NFT 拥有者；
- 事务失败则全部回滚，确保不会出现一部分数据写入、另一部分缺失的“脏写”问题。

5. ....

五、后端服务与性能优化

1. 项目中 API 服务是如何设计的？是 RESTful 还是 GraphQL？

Backend服务采用的是 **RESTful API** 架构，理由如下：

- 使用 Gin 框架构建服务，API 按资源设计路由，支持版本化（如 /api/v1/...）；
- 所有资源（Collection、Item、Activity、Order）都按照 REST 规范提供 GET/POST/PUT/DELETE；
- API 分模块管理，具体逻辑放在 src/api/router/v1 和 src/service/v1 目录中；
- 路由通过 loadV1() 按功能组装，每个 handler 都注入系统上下文 svcCtx 实现解耦。

。。。

REST 的优势：

- 与前端集成简单清晰；
- 搭配 API 缓存中间件（CacheApi）可灵活优化性能；
- API 层配合 Auth 中间件支持鉴权、Token 验证。

2. 你用了哪些缓存机制？缓存了哪些数据？

我们在多个场景中使用 Redis 缓存，提升响应速度，保护后端：

缓存方式：

- 使用封装的 xkv.Store 接口操作 Redis；
- 封装中间件 CacheApi，自动缓存接口响应数据；
- 响应体使用 JSON 序列化+哈希路径键，保证缓存唯一性。

缓存内容包括：

- NFT 项目的地板价、成交价；
- Collection 排名结果（/ranking 接口）；
- 用户挂单信息（如 Bids、Listings）；
- 高频 Item 图片接口（如 /items/:token_id/image）；
- 等等

3. ⭐个人贡献/项目难点/项目优化。。。相关问题

API 服务的高并发与性能优化

背景

NFT Market 项目上线后，由于用户数激增和交易频次增加，API 层面承受了巨大的并发压力。用户不仅需要查询海量的 NFT 信息（千万-亿级数据），还可能在高峰时段（NFT项目开盲盒 or 举办售卖活动时）同时发起交易、数据更新等操作，任何响应延迟或数据瓶颈都可能影响用户体验和系统稳定性。此外，随着 NFT 数据量和历史记录不断增加，数据库查询和写入的性能问题也逐渐显现。

目标

- **高并发响应：** 确保 API 能在高并发情况下保持低延迟，快速响应用户请求。
- **高可用性与扩展性：** 构建支持水平扩展的架构，保证系统能够动态应对流量高峰。
- **数据一致性与高性能查询：** 优化数据库设计，确保查询效率同时保证数据准确性。
- **弹性与容错：** 在部分组件或服务出现故障时，系统能快速恢复并保持整体服务可用性。

方案

1. **缓存设计**

- **热点数据缓存：**

利用 Redis 等内存数据库缓存热点数据，减少对后端数据库的频繁查询。

举例：热门 NFT 信息、排行榜、实时交易数据等。

- **合理设置 TTL：**

对缓存数据设置合适的失效时间，防止缓存脏数据，并根据数据访问频率调整 TTL 策略。

- **分布式缓存集群：**

当单点 Redis 无法承载高并发时，采用 Redis 集群方案，实现数据分片和负载均衡。

2. **数据库优化**

- **SQL 优化与索引：**

针对高频查询的表建立适当的索引(索引区分度, 最左匹配原则和聚簇索引)，使用查询分析工具（如 EXPLAIN）查找慢查询，调整 SQL 语句以提高查询效率。

举例：比如activity表，经常需要查询指定collection，指定tokenid，指定type类型的activity，可以添加（collection_address, token_id, activity_type）联合索引，并在使用过程中，依据最左匹配原则；

- **读写分离**

将数据库读写操作分离，部署主从复制，利用从库处理大部分查询请求，从而减轻主库压力。

- **数据库分库分表**

对于数据量极大的表，采取分库分表策略，按**业务逻辑**或**数据特性**拆分数据，提高查询并发性和写入效率。

举例： **不同链**的数据分表存储（**垂直分表**），**NFT Item**冷热数据分表存储（**水平分表**）

eth：eth_activity, eth_collection...

Bsc: bsc_activity, bsc_collection...

NFT: **collection_address, token_id,,,,**image_url, metadata...

3. **异步处理与消息队列**

- **任务异步化：**

对于不需要即时返回结果的操作，采用异步任务来分担 API 服务的实时压力。

举例：**刷新NFT集合的Metadata、更新NFT集合地板价**

- **消息队列引入：**

使用消息队列对高并发下的写入操作进行流量削峰，保证数据处理的平滑性和稳定性。

举例：**更新NFT集合地板价、更新订单状态**（统计NFT上架数量、检查订单过期状态）等

4. **负载均衡与水平扩展**

- **负载均衡器部署：**

通过 Nginx或云服务自带的负载均衡器（如 AWS ELB）实现流量分发，将请求均匀分配到后端多台服务器。

- **服务无状态化设计：**

设计 API 服务时尽量保证无状态，使得每个服务实例可以独立处理请求，便于水平扩展。

- **自动伸缩：**

配置自动伸缩策略，根据流量实时增加或减少实例数量，确保在高峰期资源充足，同时降低低流量时的成本。

5. **压力测试与监控调优**

- **模拟高并发环境：**

测试同事利用压测工具进行压力测试，提前模拟高流量情况，找出系统瓶颈, 针对性的进行优化。

- **实时监控与报警：**

部署 Prometheus、Grafana 等监控工具，实时跟踪 API 响应时间、错误率、数据库负载等关键指标，建立报警机制，及时发现和处理异常情况。

结果：

- **响应速度提升：**

通过缓存、负载均衡和数据库优化，API 的响应时间大幅缩短，在高并发情况下仍能保持毫秒级响应。

- **系统稳定性增强：**

自动伸缩和异步任务机制确保在流量高峰时服务稳定，系统在故障发生时也能迅速恢复，用户体验显著提升。

- **整体业务承载力增强：**

数据库分库分表和读写分离策略有效降低了单点压力，系统整体吞吐量显著提高。

链上链下同步服务的准确性与高可用性优化

。。。

六、交易撮合与资产安全

1. ⭐你实现的是链上交易撮合，还是链下撮合链上结算？为什么？

实现的是**全链上订单簿 + 链上撮合结算机制**。

实现方式：

- **挂单行为**（Make）直接通过**合约函数**发起，链上创建/存储订单并将资产转入托管 Vault合约；
- **撮合行为**（Match）由任意用户通过合约**查询订单信息**，并调用合约**发起撮合交易**，完成资产划转和订单状态更新；
- 合约本身维护**订单状态**、**成交历史**，无需任何中心化服务器。

为什么选择链上撮合？

- **透明性强**：订单和成交记录全链可查；
- **避免前置交易（Front-running）**：合约撮合逻辑严格按规则执行；
- **资金托管安全**：资产托管在 Vault，避免链下订单伪造或钓鱼链接；
- **DeFi 场景友好**：天然支持**去中心化**、**无信任交易**，适用于高价值 NFT 和协议交互。

2. 你是如何处理挂单和成交逻辑的？是否涉及订单簿？

是的，我们实现了一个完整的**链上订单簿（On-chain OrderBook）系统**。

订单簿实现要点：

1. **订单结构**定义在链上，挂单数据包括 maker、价格、资产、过期时间等；
2. **订单存储**采用**红黑树 + 链式队列结构**，按照价格 + 时间排序，实现“价格优先、时间优先”；
3. 每个 NFT 集合单独维护**价格树和订单队列**，实现高效的订单匹配逻辑；
4. 成交时，合约会**校验订单状**态、资产余额、价格一致性，并调用 Vault 进行**资产划转**；
5. 支持单笔或批量撮合，合约通过 matchOrder / **matchOrders** 执行撮合逻辑。

卖方卖nft（list）：卖方调用makeorder创建卖单（Order结构信息）-> 买方看到了这笔卖单，（前端）构造相匹配（价格匹配，collection adress, token id信息）的买单，调用matchorder撮合这两笔订单，实现购买nft（buy）

买方想要买nft（offer）：买方调用makeorder创建买单（期望价格、collection adress, token id...） -> 买方看到这笔买单，（前端）构造相匹配的卖单，调用matchorder撮合这两笔订单，实现售卖nft（sell）

通过前端交互合约方法：合约方法的参数都是前端构造的，用户只在签名签名时参数确认；

通过合约交互合约方法： 用户自己构造合约方法的参数，用户自己发起交易确认；

3. 你是如何管理用户资产托管与划转的？

设计了一个独立的 **Vault 合约** 实现资产托管逻辑，核心要点如下：

- **NFT 托管**：用户挂单卖出时，NFT 被转入 Vault；
- **ETH 托管**：挂买单时，ETH 同样会进入 Vault；
- **资产提取**：
  - 成交后，Vault 根据订单 ID 向买家转 NFT，向卖家转 ETH；
  - 若订单被取消，Vault 会原路退还资产；
- **接口限制**：Vault 合约仅接受 OrderBook 合约调用，防止被外部恶意调用；
- **支持编辑订单**：挂单价格调整时，Vault 会自动处理多退少补逻辑。

七、行业背景与趋势理解

1. 你怎么看待 NFT 市场的未来发展？

NFT 的未来将走向**资产化、应用化与协议化**三个方向：

- **资产化**：NFT 不再仅是“图片”，而是数字商品、门票、身份、金融衍生品等通用数字资产形态；
- **应用化**：NFT 会和真实业务更紧密结合，例如门票 NFT 与线下活动绑定、链游中的装备道具、音乐版权的确权与分发；
- **协议化**：从平台主导走向底层协议建设，例如 OpenSea 的 Seaport 协议、Blur 的 Blend，把 NFT 交易行为协议化、模块化，便于集成和组合创新。

思考：NFT + RWA, NFT + AI

2. ⭐NFT 项目如何与 DeFi、GameFi 等结合？你有哪些思考？

NFT 与 DeFi/GameFi 的结合，是 Web3 应用爆发的重要通路，主要思考如下：

与 DeFi 结合：

- **NFT 抵押借贷（NFT-Fi）**：将 NFT 抵押换取流动性资产，核心挑战是估值体系与清算机制；
- **AMM + NFT 流动池（如 Sudoswap）**：通过 bonding curve 提供流动性，交易更高效；
- **收益凭证/资产包裹**：NFT 表征收益凭证、保险产品、期权等组合金融工具。

与 GameFi 结合：

- **资产拥有权 + 道具可组合（ERC-6551）；**
- **游戏内经济体系透明**：NFT 持有即参与经济生态，推动“Play to Own”；
- **链上升级与技能绑定机制**：NFT 本身通过合约实现升级、技能拓展，替代中心化数据库。

3. 你认为 NFT 项目中最大的技术挑战是什么？

当前 NFT 项目面临的**三大技术挑战**是：

1.**链上链下数据一致性与高效同步**：链上事件繁杂、链下数据需实时构建，特别是活动、订单等频繁变化信息，挑战事件处理与回滚机制；

2.**性能与可扩展性**：热门项目中订单簿、活动记录、属性索引等会遇到查询瓶颈，如何构建高效缓存与分页查询结构是难点；

3.**安全性与资产托管机制**：尤其在自建 Vault 或订单簿撮合系统中，要处理重入、授权、资产转移等多个攻击面，考验合约设计细节。