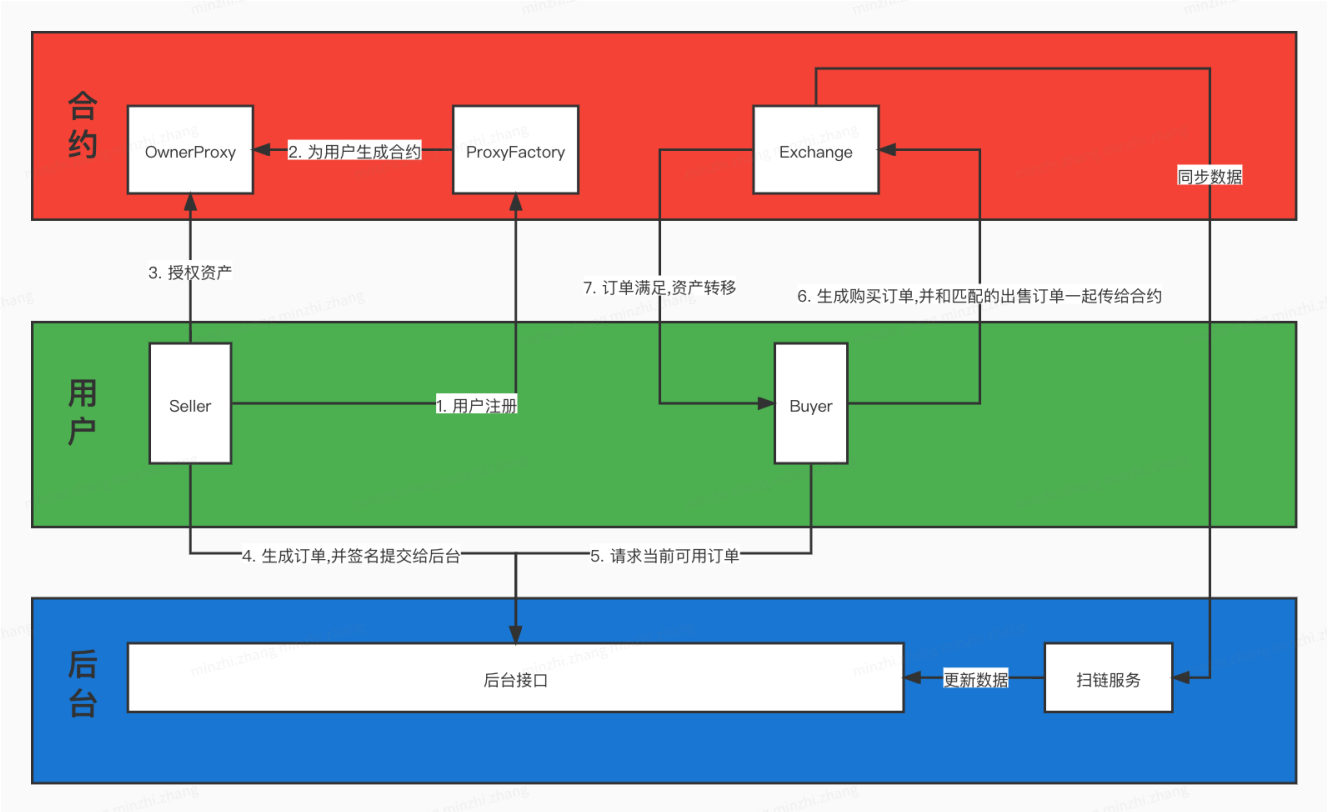
Market设计文档

整体结构



详细流程

用户注册

用户第一次加入需要进行一次注册流程。前端调用GameRegistry合约的 registerProxy 方法进行注册，方法调用完成后会给用为用户创建一个合约 OwnerProxy。通过调用registerProxy合约的proxies(user)方法能够查询为用户生成的OwnerProxy。如果该方法返回的是全0地址，则需要进行注册，否则进行下一步流程。只有卖方需要进行注册，买方不需要注册。

授权资产

卖方调用合约的授权方法，将资产授权给上一步为用户生成的OwnerProxy合约。授权后合约可以调用transferFrom等方法操作用户的资产。而该合约只有Exchange合约有权限调用触发。

生成订单

卖方生成一个具有如下字段的订单：

JavaScript

```
    let sellOrder = {
        exchange: this.exchange.address,
        maker: this.seller.address,
        taker: ADDRESS_ZERO,
        makerRelayerFee: RELAYER_FEE,
        takerRelayerFee: 0,
        makerProtocolFee: 0,
        takerProtocolFee: 0,
        feeRecipient: FEE_ADDRESS,
        feeMethod: 1,
        side: 1,
        saleKind: 0,
        target: this.erc721.address,
        howToCall: 0,
        calldata: IFACE.encodeFunctionData("transferFrom",
[this.seller.address, ADDRESS_ZERO, 1]),
        replacementPattern: ERC721_REPLACEMENT_SELL,
        staticTarget: ADDRESS_ZERO,
        staticExtradata: "0x",
        paymentToken: ADDRESS_ZERO,
        basePrice: '30000000000000000',
        extra: 0,
        listingTime: 1640785686,
        expirationTime: 0,
        salt: 1640785686000,
    };
```

各个字段的解释如下：

1. exchange: Exchange合约的地址
2. maker: 这个订单的创建者，也即是用户的地址
3. taker: 这个订单的接受者。每笔交易都是由两个订单组成的，一个购买订单一个售卖订单。我们可以指定一个订单他的接受者是谁。如果是全0地址则是任意人。
4. makerRelayerFee: 创建者需要支付的手续费比例
5. takerRelayerFee: 接收者需要支付的手续费比例
6. makerProtocolFee: 创建者需要支付的协议手续费比例
7. takerProtocolFee: 接收者需要支付的协议手续费比例
8. feeRecipient: 手续费接收地址
9. feeMethod: 手续费支付方式
10. side: 买方还是卖方
11. saleKind: 交易方式：固定价格或者荷兰拍

12. target：当订单通过后会执行这个合约上的某个方法，对于ERC721的订单，这个就是ERC721的合约地址
13. howToCall: 调用方式：staticCall或delegateCall
14. calldata: 调用的方法
15. replacementPattern: 调用方法的模式匹配
16. staticTarget: 提供一种通知模式，当一个订单执行之后会调用这个方法做一个通知
17. staticExtradata: 通知调用的函数
18. paymentToken: 需要支付的币种
19. basePrice: 支付的价格
20. extra: 如果是荷兰拍表示最低价格
21. listingTime: 订单的开始时间
22. expirationTime: 订单的超时时间，如果设置为0表示永远不超时
23. salt：订单的盐值，防止两个订单hash相同，一般取当前时间的毫秒数

生成订单数据后调用合约的 hashOrder_方法，即可得到一个订单的hash值：

JavaScript

```
 let hash = await this.exchange.hashOrder_(
            [sellOrder.exchange, sellOrder.maker, sellOrder.taker, sellOrder.feeRecipient, sellOrder.target, sellOrder.staticTarget, sellOrder.paymentToken],
            [sellOrder.makerRelayerFee, sellOrder.takerRelayerFee, sellOrder.makerProtocolFee, sellOrder.takerProtocolFee, sellOrder.basePrice, sellOrder.extra, sellOrder.listingTime, sellOrder.expirationTime, sellOrder.salt],
            sellOrder.feeMethod,
            sellOrder.side,
            sellOrder.saleKind,
            sellOrder.howToCall,
            sellOrder.calldata,
            sellOrder.replacementPattern,
            sellOrder.staticExtradata
        );
```

之后调用签名方法对订单进行签名

JavaScript

```
    let hashBytes = ethers.utils.arrayify(hash)
    let flatSig = await this.seller.signMessage(hashBytes);
    let sig = ethers.utils.splitSignature(flatSig);
```

生成签名之后可以调用validateOrder_方法进行验证，看签名是否合法

JavaScript

```
    expect(await this.exchange.validateOrder_(
        [sellOrder.exchange, sellOrder.maker, sellOrder.taker,
sellOrder.feeRecipient, sellOrder.target, sellOrder.staticTarget,
sellOrder.paymentToken],
        [sellOrder.makerRelayerFee, sellOrder.takerRelayerFee,
sellOrder.makerProtocolFee, sellOrder.takerProtocolFee, sellOrder.basePrice,
sellOrder.extra, sellOrder.listingTime, sellOrder.expirationTime,
sellOrder.salt],
        sellOrder.feeMethod,
        sellOrder.side,
        sellOrder.saleKind,
        sellOrder.howToCall,
        sellOrder.calldata,
        sellOrder.replacementPattern,
        sellOrder.staticExtradata,
        sig.v, sig.r, sig.s
    )).to.be.equal(true);
```

生成订单后，将相关信息传给后台，后台验证通过后，即表示挂单成功。

请求订单

经过前面的步骤，后台会收到所有的订单以及相关信息。这些信息保存在数据库中，之后卖方查询接口，接口根据业务需求返回给用户相关的挂单数据，供用户选择。

生成购买订单

买方选定了想要的商品后会生成一个购买订单，这样他就有一个一个卖单和一个买单两个订单。订单的结构是完全一样的，只是数据内容不一样，下面是一个具体的买单数据：

JavaScript

```
    let buyOrder = {
        exchange: this.exchange.address,
        maker: this.buyer.address,
        taker: this.seller.address,
        makerRelayerFee: RELAYER_FEE,
        takerRelayerFee: 0,
        makerProtocolFee: 0,
        takerProtocolFee: 0,
        feeRecipient: ADDRESS_ZERO,
        feeMethod: 1,
        side: 0,
        saleKind: 0,
        target: this.erc721.address,
        howToCall: 0,
        calldata: IFACE.encodeFunctionData("transferFrom", [ADDRESS_ZERO, this.buyer.address, 1]),
        replacementPattern: ERC721_REPLACEMENT_BUY,
        staticTarget: ADDRESS_ZERO,
        staticExtradata: "0x",
        paymentToken: ADDRESS_ZERO,
        basePrice: '30000000000000000',
        extra: 0,
        listingTime: 1640785686,
        expirationTime: 0,
        salt: 1640785686000,
    };
```

之后同样的方法求hash，签名，然后调用合约的atomicMatch_方法，传入两个订单数据和签名数据进行购买：

JavaScript

```
    await this.exchange.connect(this.buyer).atomicMatch_(
        [
            this.buyOrder.exchange, this.buyOrder.maker, this.buyOrder.taker, this.buyOrder.feeRecipient, this.buyOrder.target, this.buyOrder.staticTarget, this.buyOrder.paymentToken,
            this.sellOrder.exchange, this.sellOrder.maker, this.sellOrder.taker, this.sellOrder.feeRecipient, this.sellOrder.target, this.sellOrder.staticTarget, this.sellOrder.paymentToken,
        ],
        [
            this.buyOrder.makerRelayerFee, this.buyOrder.takerRelayerFee, this.buyOrder.makerProtocolFee, this.buyOrder.takerProtocolFee, this.buyOrder.basePrice, this.buyOrder.extra, this.buyOrder.listingTime, this.buyOrder.expirationTime, this.buyOrder.salt,
            this.sellOrder.makerRelayerFee, this.sellOrder.takerRelayerFee, this.sellOrder.makerProtocolFee, this.sellOrder.takerProtocolFee, this.sellOrder.basePrice, this.sellOrder.extra, this.sellOrder.listingTime, this.sellOrder.expirationTime, this.sellOrder.salt,
        ],
        [
            this.buyOrder.feeMethod, this.buyOrder.side, this.buyOrder.saleKind, this.buyOrder.howToCall,
            this.sellOrder.feeMethod, this.sellOrder.side, this.sellOrder.saleKind, this.sellOrder.howToCall,
        ],
        this.buyOrder.calldata, this.sellOrder.calldata, this.buyOrder.replacementPattern, this.sellOrder.replacementPattern, this.buyOrder.staticExtradata, this.sellOrder.staticExtradata,
        [
            this.buySig.v, this.sellSig.v
        ],
        [
            this.buySig.r, this.buySig.s, this.sellSig.r, this.sellSig.s, "0x0000000000000000000000000000000000000000000000000000000000000000"
        ],
        {value: "30000000000000000"}
    );
```

执行成功后，资产转移给相应用户

其他说明

手续费问题

Exchange中收两种手续费，一种是relayerFee一种是protocolFee，这两种手续费的区别是一种收的币种是平台币，另一种收的币种是当前的交易币种。同时两种手续费都可以选择由买方支付或者卖方支付。只要挂单者和成交者两个订单就手续费的问题达成一致即可。

如果是relayerFee 这个时候支付的是挂单的币种，此时可以在订单中指明要将手续费交给哪个地址。理论上这里可以是任意地址，即不一定是项目方的地址，只要双方达成一致即可。Exchange的思想是，只是调用合约是免费的，因为这是一个去中心化的服务，只有将订单提交给中心化服务存储，这时候第三方才应该收手续费。因此在后台接口收到订单的时候需要判断这个手续费地址是不是平台的地址，如果不是则不提供中心化服务。

如果是protocolFee这个时候支付的手续费是平台币，因此手续费会转到一个特定的地址。之所以有这两种区分，是因为Exchange希望达到不同币种的手续费由不同的地址接收，最终一个地址下只有特定种类的资产。

calldata数据

资产的交易主要由order里的两个字段决定，一个是合约里的target表示订单成交后调用哪个合约，另一个是calldata表示订单成交后调用这个合约的哪个方法，以及相关参数是什么。

但这样存在一个问题就是，第一个用户挂单时，此时还不知道交易的参数具体是什么，比如卖出NFT，由于还不知道最后会和谁成交，因此无法确定这个NFT要转给谁。因此需要引入一个额外的参数replacePattern。这样在两个成交订单中一个包含四个参数

1. 卖方的calldata
2. 卖方的replacePattern
3. 买房的calldata
4. 买方的replacePattern

以一个ERC721的买单卖单为例：当卖方挂单时不确定买方是谁因此生成的transferFrom参数的to参数时全0地址，同时在pattern中标志出calldata中to的部分并用全 f 标志。同理买房将transferFrom的from参数设置为全0地址，并在pattern中标志处，最终对这四个数据做模运算，只要两次模运算算出来的数据是一样的，就通过。并最终调用合约的这个数据方法。

上述例子中假如卖方地址为：0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC 买方地址为：0x70997970C51812dc3A010C7d01b50e0d17dc79C8 要交易的nftId为1。因此

1. 卖方的calldata为 transferFrom(0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC,0x0000000000000000000000000000000000000000,1) 最终得到的数据为：0x23b872dd0000000000000000000000003c44cdddb6a900fa2b585dd299e03d12fa4293bc0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
2. 卖方的replacementPattern为 0x000000000000000000000000000000000000000000000000000000000000000000000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00000000000000000000000000000000000000000

00000000000000000000000

3. 买方的calldata为
transferFrom(0x0000000000000000000000000000000000000000,0x70997970C51812dc3A01
0C7d01b50e0d17dc79C8,1) 最终得到的数据为：
0x23b872dd000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000070997970c51812dc3a010c7d01b50e0d17dc79c800000000000000
00000000000000000000000000000000000000000000000001

4. 买方的calldata为
0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000

最终结果为：

(!buy.replacementPattern && buy.calldata) || (buy.replacementPatther && sell.calldata)

==

(!sell.replacementPattern && sell.calldata) || (sell.replacementPattern && buy.calldata)

目前如果只需要支持ERC721 ERC1155 那么对于replacementPattern 只有四种固定的值。

1. 买方ERC721：

   00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff0000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000000000000000
   000000000000000000000

2. 卖方ERC721：

   000000000000000000000000000000000000000000000000000000000000000000000000ffffffff
   ffffffffffffffffffffffffffffffffffffffffffffffffffffff00000000000000000000000000000000000000000000
   00000000000000000000

3. 买方ERC1155：

   00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff0000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000

4. 卖方ERC1155：

   000000000000000000000000000000000000000000000000000000000000000000000000ffffffff
   ffffffffffffffffffffffffffffffffffffffffffffffffffffff00000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000000000000000
   0000000000000000000000000000000000000000000000000000000000000000000000000000
   000000000000000000000000000000000000000000000000000000000000

其他难点

1. NFT的展示问题，如何展示出一个用户所有用的所有NFT，是否需要离线跑出数据
2. 后台数据和链上数据的同步问题，当一个订单被购买了如何快速的更新后台数据