2021131126-区块链214-熊灵欣-第五次实验报告

1. 完善合约 contracts/erc20-usdt.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract cUSDT is ERC20 {

    constructor()

        ERC20("fake usdt in cbi", "cUSDTnp")

    {

         \_mint(msg.sender, 1 \* 10 \*\* 8 \* 10 \*\* 18);

    }

}

1. 完善合约 contracts/erc721-nft.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";

import "@openzeppelin/contracts/access/Ownable.sol";

contract NFTM is ERC721, ERC721Enumerable, Ownable {

    uint256 private \_nextTokenId;

    constructor(address initialOwner)

        ERC721("NFTM", "NFTM")

        Ownable(initialOwner)

    {}

    function \_baseURI() internal pure override returns (string memory) {

        return "https://cbisample.com/";

    }

    function safeMint(address to) public onlyOwner {

        uint256 tokenId = \_nextTokenId++;

        \_safeMint(to, tokenId);

    }

    // The following functions are overrides required by Solidity.

    function \_update(address to, uint256 tokenId, address auth)

        internal

        override(ERC721, ERC721Enumerable)

        returns (address)

    {

        return super.\_update(to, tokenId, auth);

    }

    function \_increaseBalance(address account, uint128 value)

        internal

        override(ERC721, ERC721Enumerable)

    {

        super.\_increaseBalance(account, value);

    }

    function supportsInterface(bytes4 interfaceId)

        public

        view

        override(ERC721, ERC721Enumerable)

        returns (bool)

    {

        return super.supportsInterface(interfaceId);

    }

}

1. 完善合约nft-market

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import "@openzeppelin/contracts/interfaces/IERC721Receiver.sol";

import "@openzeppelin/contracts/interfaces/IERC721.sol";

import "@openzeppelin/contracts/interfaces/IERC20.sol";

contract Market {

    IERC20 public erc20;

    IERC721 public erc721;

    bytes4 internal constant \_ERC721\_RECEIVED = 0x150b7a02;

    struct  Order {

        address seller;

        uint256 tokenId;

        uint256 price;

    }

    mapping (uint256 => Order) public orderOfId;        // token id to order

    Order[] public orders;

    mapping(uint256 => uint256) public idToOrderIndex;  //token id to index of order

    event Deal(address seller,address buyer,uint256 tokenId,uint256 price);

    event NewOrder(address seller,uint256 tokenId,uint256 price);

    event PriceChanged (address seller,uint256 tokenId,uint256 PreviousPrice,uint256 NewPrice);

    event CancelOrder(address seller,uint256 tokenId);

    constructor (address \_erc20, address \_erc721) {

        require(\_erc20 != address(0), "ERC20 address cannot be 0");

        require(\_erc721 != address(0), "ERC721 address cannot be 0");

        erc20 = IERC20(\_erc20);

        erc721 = IERC721(\_erc721);

    }

    function buy(uint256 \_tokenId) external {

        address seller = orderOfId[\_tokenId].seller;

        address buyer = msg.sender;

        uint256 price = orderOfId[\_tokenId].price;

        require(erc20.transferFrom(buyer,seller,price), "transfer failed");

        erc721.safeTransferFrom(address(this),buyer,\_tokenId);

        removeOrder(\_tokenId);

        emit Deal(seller,buyer,\_tokenId,price);

    }

    function cancelOrder(uint256 \_tokenId) external {

        address seller = orderOfId[\_tokenId].seller;

        require(msg.sender == seller, "only seller can cancel order");

        erc721.safeTransferFrom(address(this),seller,\_tokenId);

        removeOrder(\_tokenId);

        emit CancelOrder(seller,\_tokenId);

    }

    function changePrice(uint256 \_tokenId,uint256 \_price) external {

        address seller = orderOfId[\_tokenId].seller;

        require(msg.sender == seller, "only seller can change price");

        uint256 previousPrice = orderOfId[\_tokenId].price;

        orderOfId[\_tokenId].price = \_price;

        Order storage order = orders[idToOrderIndex[\_tokenId]];

        order.price = \_price;

        emit PriceChanged(seller,\_tokenId,previousPrice,\_price);

    }

    function onERC721Received(address operator, address from, uint256 tokenId, bytes calldata data) external returns (bytes4) {

        uint256 price = toUint256(data,0);

        require(price > 0, "Market: price is zero");

        // 上架

        orders.push(Order(from,tokenId,price));

        orderOfId[tokenId] = Order(from,tokenId,price);

        idToOrderIndex[tokenId] = orders.length - 1;

        emit NewOrder(from,tokenId,price);

        return \_ERC721\_RECEIVED;

    }

    function removeOrder(uint256 \_tokenId) internal {

        uint256 orderIndex = idToOrderIndex[\_tokenId];

        uint256 lastOrderIndex = orders.length - 1;

        if (orderIndex != lastOrderIndex){

            Order storage lastOrder = orders[lastOrderIndex];

            orders[orderIndex] = lastOrder;

            idToOrderIndex[lastOrder.tokenId] = orderIndex;

        }

        orders.pop();

        delete orderOfId[\_tokenId];

        delete idToOrderIndex[\_tokenId];

    }

    function getOrderLength() external view returns(uint256){

        return orders.length;

    }

    function getAllNFTs() external view returns(Order[] memory){

        return orders;

    }

    function getMyNFTs() external view returns (Order[] memory){

        Order[] memory myOrders = new Order[](orders.length);

        uint256 count = 0;

        for(uint256 i = 0; i < orders.length; i++){

            if(orders[i].seller == msg.sender){

                myOrders[count] = orders[i];

                count++;

            }

        }

        return myOrders;

    }

     function isListed(uint256 \_tokenId) public view returns (bool){

        return orderOfId[\_tokenId].seller !=address(0);

    }

    //https://stackoverflow.com/questions/63252057/how-to-use-bytestouint-function-in-solidity-the-one-with-assembly

   function toUint256(bytes memory \_bytes, uint256 \_start) internal pure returns (uint256) {

        require(\_bytes.length >= (\_start + 32), "Read out of bounds");

        uint256 tempUint;

        assembly {

            tempUint := mload(add(add(\_bytes, 0x20), \_start))

        }

        return tempUint;

    }

}

1. 完善测试 test/market.js

const {expect} = require('chai');

const {ethers} = require('hardhat');

describe ('Market', async function(){

    let usdt, nft, market, accountA, accountB;

    beforeEach(async () =>{

        [accountA, accountB] = await ethers.getSigners();

        const USDT = await ethers.getContractFactory("cUSDT");

        usdt = await USDT.deploy();

        const MyNFT = await ethers.getContractFactory("NFTM");

        nft = await MyNFT.deploy(accountA.address);

        const Market = await ethers.getContractFactory("Market");

        market = await Market.deploy(usdt.target, nft.target);

        await nft.safeMint(accountB.address);

        await nft.safeMint(accountB.address);

        await nft.connect(accountB).setApprovalForAll(accountA.address, true);

        await usdt.approve(market.target, "1000000000000000000000000");

    });

    it('its erc20 address should be usdt', async function(){

        expect(await market.erc20()).to.equal(usdt.target);

    });

    it('its erc721 address should be nft', async function(){

        expect(await market.erc721()).to.equal(nft.target);

    });

    it('accountB should have two nfts', async () => {

        expect(await nft.balanceOf(accountB.address)).to.equal(2);

    });

    it('accountA should have usdt', async () => {

        expect(await usdt.balanceOf(accountA.address)).to.equal("100000000000000000000000000");

    });

    it('accountB can list two nfts to market', async () => {

        const price= "0x0000000000000000000000000000000000000000000000000001c6bf52634000";

        expect(await nft['safeTransferFrom(address, address, uint256, bytes)'](accountB.address, market.target, 0, price)).to.emit(market, "NewOrder");

        expect(await nft['safeTransferFrom(address, address, uint256, bytes)'](accountB.address, market.target, 1, price)).to.emit(market, "NewOrder");

        expect(await nft.balanceOf(accountB.address)).to.equal(0);

        expect(await nft.balanceOf(market.target)).to.equal(2);

        expect(await market.isListed(0)).to.equal(true);

        expect(await market.isListed(1)).to.equal(true);

        expect((await market.connect(accountB).getAllNFTs())[0][0]).to.equal(accountB.address);

        expect((await market.connect(accountB).getAllNFTs())[0][1]).to.equal(0);

        expect((await market.connect(accountB).getAllNFTs())[0][2]).to.equal(price);

        expect((await market.connect(accountB).getAllNFTs())[1][0]).to.equal(accountB.address);

        expect((await market.connect(accountB).getAllNFTs())[1][1]).to.equal(1);

        expect((await market.connect(accountB).getAllNFTs())[1][2]).to.equal(price);

        expect(await market.getOrderLength()).to.equal(2);

        expect((await market.connect(accountB).getMyNFTs())[0][0]).to.equal(accountB.address);

        expect((await market.connect(accountB).getMyNFTs())[0][1]).to.equal(0);

        expect((await market.connect(accountB).getMyNFTs())[0][2]).to.equal(price);

    });

    it('accountB can remove one nft from market', async function () {

        it('accountB can remove one nft from market', async function() {

            const price = "0x0000000000000000000000000000000000000000000000000001c6bf52634000";

             // 列出NFT到市场

             await nft.connect(accountB).safeTransferFrom(accountB.address, market.address, 0, price);

             await nft.connect(accountB).safeTransferFrom(accountB.address, market.address, 1, price);

            // 验证市场状态

            expect(await market.isListed(0)).to.equal(true);

            expect(await market.isListed(1)).to.equal(true);

            expect(await market.getOrderLength()).to.equal(2);

            // 下架NFT

            await market.removeOrder(0);

            // 验证NFT是否成功下架

            expect(await market.isListed(0)).to.equal(false);

            expect(await market.isListed(1)).to.equal(true);

            expect(await market.getOrderLength()).to.equal(1);

        });

    });

      it('accountB can change price of nft from market', async function() {

        it('accountB can change price of nft from market', async function () {

            //上架一个nft

            await nft.connect(account1).safeTransferFrom(accountB.address, market.address, 0, "0x0000000000000000000000000000000000000000000000000DE0B6B3A7640000");

            // 更改NFT在市场上的价格 1 -> 0.5

            await market.changePrice(0, 500000000000000000);

            // 检查价格是否正确更新

            expect(await market.orderOfId(0).price).to.equal(500000000000000000);

          });

      });

      it('accountB can buy nft from market', async function() {

        it('accountB can buy nft from market', async function () {

            //上架一个nft 价格为1000000000000000000

            await nft.connect(accountB).safeTransferFrom(accountB.address, market.address, 0, "0x0000000000000000000000000000000000000000000000000DE0B6B3A7640000");

            // 从市场购买NFT

            await market.connect(accountA).buy(0, { value: await market.orderOfId(0).price });

            // 检查交易是否成功，NFT是否正确转移

            expect(await nft.ownerOf(0)).to.equal(accountB.address);

          });

      });

      it('accountA should have 0 nfts', async function() {

        expect(await nft.balanceOf(accountA.address)).to.equal(0);

      });

});

1. 测试通过



