**周亚男 2020131062 202班**

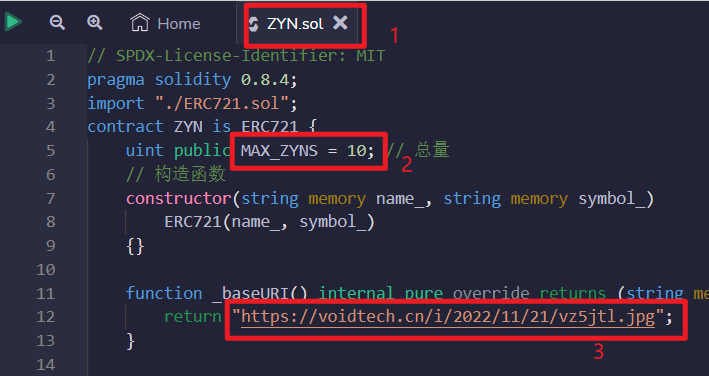
1. 准备三个账户

0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

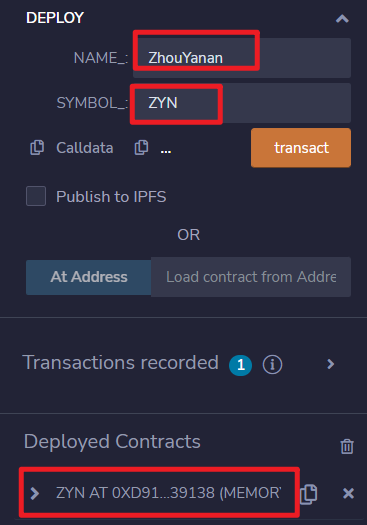
0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db  
  
二、头像图片地址

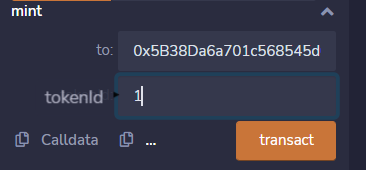
<https://voidtech.cn/i/2022/11/21/vz5jtl.jpg>

三、修改合约  


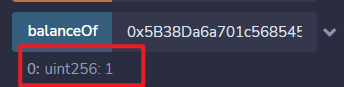
四、部署合约到本地测试网



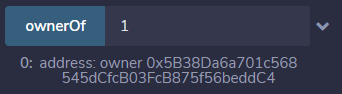
五、mint一个tokenid为1的nft到账户A



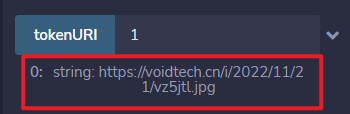
六、查看账户A的余额



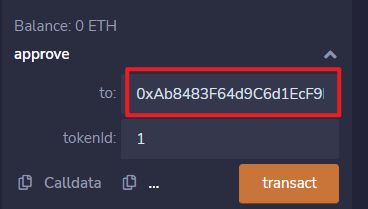
七、查看tokenid为1的owner



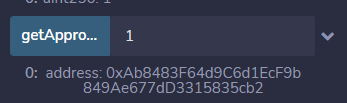
八、查看tokenid为1的tokenURI



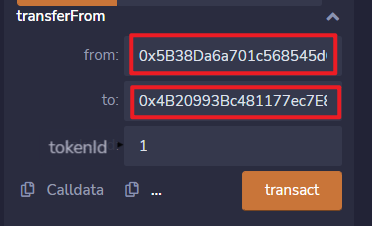
九、使用账户A对账户B进行approve



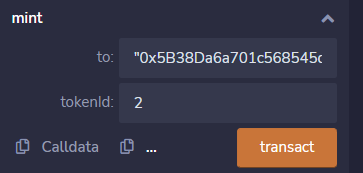
十、调用getApproved，查看tokenid为1的返回值



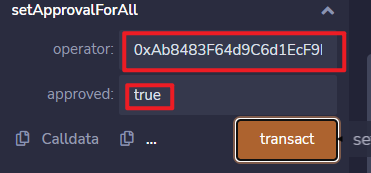
十一、调用transferFrom，把tokenid为1的NFT从账户B转到账户C



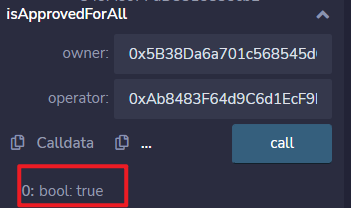
十二、mint tokenid为2 的nft到账户A；



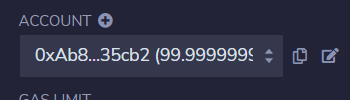
13. 使用账户A调用setApprovalForAll，对账户B进行approve

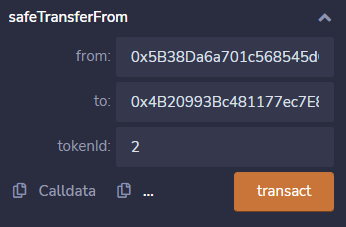


14. 调用isApprovedForAll，查看授权状态

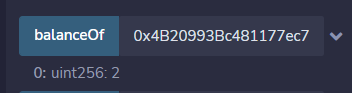


15. 使用账户B，转移tokenid为2的NFT到账户C；

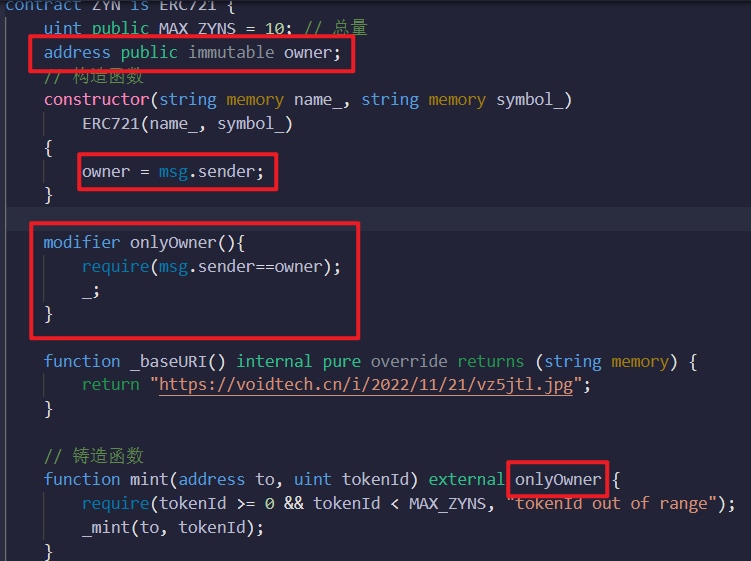




16. 查看账户C的余额



17. 修改合约代码，使得只有owner才可以mint； 截图自己修改后的代码放到这里；



18. 对比erc20和erc721 transfer事件的异同，并阐述原因；

erc20：是拥有者对某人转一定数量的币

erc721: 是拥有者对某人转tokenid代表的这一个“nft“

原因：nft不能按照数量来分发，要不然也不会有setApprovalForAll()

19. 对比erc20和ERC721 approval 事件的异同，并阐述原因；

原理同18

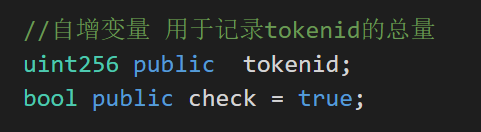
20. 在这个实验里，TokenURI的对应的图片是存储在链上还是链 下？

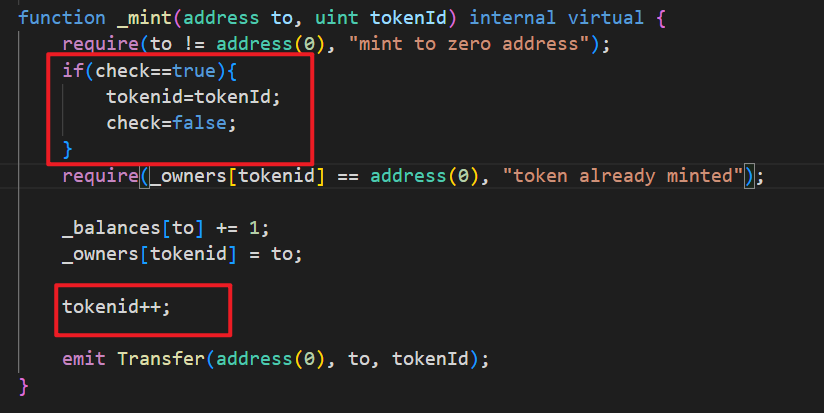
链下，图片与链上有个锚定关系，可通过节点查询到链下存储图片的地方

21. 为什么ERC721合约的接口里没有transfer方法？

被**transferFrom（非安全）和safetransferFrom（安全）** 替代。（由于ERC721代币具有唯一性，不可分割）

加分项： 1. 修改合约，使得在mint时，tokenid自增，而不是用户指定 tokenid；





源码：

ZYN.sol

// SPDX-License-Identifier: MIT

pragma solidity 0.8.4;

import "./ERC721.sol";

contract ZYN is ERC721 {

    uint public MAX\_ZYNS = 10; // 总量

    address public immutable owner;

    // 构造函数

    constructor(string memory name\_, string memory symbol\_)

        ERC721(name\_, symbol\_)

    {

        owner = msg.sender;

    }

    modifier onlyOwner(){

        require(msg.sender==owner);

        \_;

    }

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

        return "https://voidtech.cn/i/2022/11/21/vz5jtl.jpg";

    }

    // 铸造函数

    function mint(address to, uint tokenId) external onlyOwner {

        require(tokenId >= 0 && tokenId < MAX\_ZYNS, "tokenId out of range");

        \_mint(to, tokenId);

    }

}

ERC721

// SPDX-License-Identifier: MIT

pragma solidity 0.8.4;

import "./IERC165.sol";

import "./IERC721.sol";

import "./IERC721Receiver.sol";

import "./IERC721Metadata.sol";

import "./Address.sol";

import "./String.sol";

// 对IERC721的实现（方法的重写）

contract ERC721 is IERC721, IERC721Metadata {

    using Address for address; // 使用Address库，用isContract来判断地址是否为合约

    using Strings for uint256; // 使用String库，

    //自增变量 用于记录tokenid的总量

    uint256 public  tokenid;

    bool public check = true;

    // Token名称

    string public override name;

    // Token代号

    string public override symbol;

    // tokenId 到 owner address 的持有人映射

    mapping(uint => address) private \_owners;

    // address 到 持仓数量 的持仓量映射

    mapping(address => uint) private \_balances;

    // tokenID 到 授权地址 的授权映射

    // 也就是某人有这个tokenid的使用权

    mapping(uint => address) private \_tokenApprovals;

    //  owner地址。到operator地址 的批量授权映射

    mapping(address => mapping(address => bool)) private \_operatorApprovals;

    /\*\*

     \* 构造函数，初始化`name` 和`symbol` .

     \*/

    constructor(string memory name\_, string memory symbol\_) {

        name = name\_;

        symbol = symbol\_;

    }

    // 实现IERC165接口supportsInterface

    function supportsInterface(bytes4 interfaceId)

        external

        pure

        override

        returns (bool)

    {

        return

            interfaceId == type(IERC721).interfaceId ||

            interfaceId == type(IERC165).interfaceId ||

            interfaceId == type(IERC721Metadata).interfaceId;

    }

    // 实现IERC721的balanceOf，利用\_balances变量查询owner地址的balance。

    function balanceOf(address owner) external view override returns (uint) {

        require(owner != address(0), "owner = zero address");

        return \_balances[owner];

    }

    // 实现IERC721的ownerOf，利用\_owners变量查询tokenId的owner。

    function ownerOf(uint tokenId)

        public

        view

        override

        returns (address owner)

    {

        owner = \_owners[tokenId];

        require(owner != address(0), "token doesn't exist");

    }

    // 实现IERC721的isApprovedForAll，利用\_operatorApprovals变量查询owner地址是否将所持NFT批量授权给了operator地址。

    function isApprovedForAll(address owner, address operator)

        external

        view

        override

        returns (bool)

    {

        return \_operatorApprovals[owner][operator];

    }

    // 实现IERC721的setApprovalForAll，将持有代币全部授权给operator地址。调用\_setApprovalForAll函数。

    function setApprovalForAll(address operator, bool approved)

        external

        override

    {

        \_operatorApprovals[msg.sender][operator] = approved;

        emit ApprovalForAll(msg.sender, operator, approved);

    }

    // 实现IERC721的getApproved，利用\_tokenApprovals变量查询tokenId的授权地址。

    function getApproved(uint tokenId)

        external

        view

        override

        returns (address)

    {

        require(\_owners[tokenId] != address(0), "token doesn't exist");

        return \_tokenApprovals[tokenId];

    }

    // 授权函数。通过调整\_tokenApprovals来，授权 to 地址操作 tokenId，同时释放Approval事件。

    function \_approve(

        address owner,

        address to,

        uint tokenId

    ) private {

        \_tokenApprovals[tokenId] = to;

        emit Approval(owner, to, tokenId);

    }

    // 实现IERC721的approve，将tokenId授权给 to 地址。条件：to不是owner，且msg.sender是owner或授权地址。调用\_approve函数。

    function approve(address to, uint tokenId) external override {

        address owner = \_owners[tokenId];

        require(

            msg.sender == owner || \_operatorApprovals[owner][msg.sender],

            "not owner nor approved for all"

        );

        \_approve(owner, to, tokenId);

    }

    // 查询 spender地址是否可以使用tokenId（他是owner或被授权地址）。

    function \_isApprovedOrOwner(

        address owner,

        address spender,

        uint tokenId

    ) private view returns (bool) {

        return (spender == owner ||

            \_tokenApprovals[tokenId] == spender ||

            \_operatorApprovals[owner][spender]);

    }

    /\*

     \* 转账函数。通过调整\_balances和\_owner变量将 tokenId 从 from 转账给 to，同时释放Transfer事件。

     \* 条件:

     \* 1. tokenId 被 from 拥有

     \* 2. to 不是0地址

     \*/

    function \_transfer(

        address owner,

        address from,

        address to,

        uint tokenId

    ) private {

        require(from == owner, "not owner");

        require(to != address(0), "transfer to the zero address");

        \_approve(owner, address(0), tokenId);

        \_balances[from] -= 1;

        \_balances[to] += 1;

        \_owners[tokenId] = to;

        emit Transfer(from, to, tokenId);

    }

    // 实现IERC721的transferFrom，非安全转账，不建议使用。调用\_transfer函数

    function transferFrom(

        address from,

        address to,

        uint tokenId

    ) external override {

        address owner = ownerOf(tokenId);

        require(

            \_isApprovedOrOwner(owner, msg.sender, tokenId),

            "not owner nor approved"

        );

        \_transfer(owner, from, to, tokenId);

    }

    /\*\*

     \* 安全转账，安全地将 tokenId 代币从 from 转移到 to，会检查合约接收者是否了解 ERC721 协议，以防止代币被永久锁定。调用了\_transfer函数和\_checkOnERC721Received函数。条件：

     \* from 不能是0地址.

     \* to 不能是0地址.

     \* tokenId 代币必须存在，并且被 from拥有.

     \* 如果 to 是智能合约, 他必须支持 IERC721Receiver-onERC721Received.

     \*/

    function \_safeTransfer(

        address owner,

        address from,

        address to,

        uint tokenId,

        bytes memory \_data

    ) private {

        \_transfer(owner, from, to, tokenId);

        require(

            \_checkOnERC721Received(from, to, tokenId, \_data),

            "not ERC721Receiver"

        );

    }

    /\*\*

     \* 实现IERC721的safeTransferFrom，安全转账，调用了\_safeTransfer函数。

     \*/

    function safeTransferFrom(

        address from,

        address to,

        uint tokenId,

        bytes memory \_data

    ) public override {

        address owner = ownerOf(tokenId);

        require(

            \_isApprovedOrOwner(owner, msg.sender, tokenId),

            "not owner nor approved"

        );

        \_safeTransfer(owner, from, to, tokenId, \_data);

    }

    // safeTransferFrom重载函数

    function safeTransferFrom(

        address from,

        address to,

        uint tokenId

    ) external override {

        safeTransferFrom(from, to, tokenId, "");

    }

    /\*\*

     \* 铸造函数。通过调整\_balances和\_owners变量来铸造tokenId并转账给 to，同时释放Transfer事件。铸造函数。通过调整\_balances和\_owners变量来铸造tokenId并转账给 to，同时释放Transfer事件。

     \* 这个mint函数所有人都能调用，实际使用需要开发人员重写，加上一些条件。

     \* 条件:

     \* 1. tokenId尚不存在。

     \* 2. to不是0地址.

     \*/

    function \_mint(address to, uint tokenId) internal virtual {

        require(to != address(0), "mint to zero address");

        if(check==true){

            tokenid=tokenId;

            check=false;

        }

        require(\_owners[tokenid] == address(0), "token already minted");

        \_balances[to] += 1;

        \_owners[tokenid] = to;

        tokenid++;

        emit Transfer(address(0), to, tokenId);

    }

    // function \_mint(address to, uint tokenId) internal virtual {

    //     require(to != address(0), "mint to zero address");

    //     require(\_owners[tokenId] == address(0), "token already minted");

    //     \_balances[to] += 1;

    //     \_owners[tokenId] = to;

    //     emit Transfer(address(0), to, tokenId);

    // }

    // 销毁函数，通过调整\_balances和\_owners变量来销毁tokenId，同时释放Transfer事件。条件：tokenId存在。

    function \_burn(uint tokenId) internal virtual {

        address owner = ownerOf(tokenId);

        require(msg.sender == owner, "not owner of token");

        \_approve(owner, address(0), tokenId);

        \_balances[owner] -= 1;

        delete \_owners[tokenId];

        emit Transfer(owner, address(0), tokenId);

    }

    // \_checkOnERC721Received：函数，用于在 to 为合约的时候调用IERC721Receiver-onERC721Received, 以防 tokenId 被不小心转入黑洞。

    function \_checkOnERC721Received(

        address from,

        address to,

        uint tokenId,

        bytes memory \_data

    ) private returns (bool) {

        if (to.isContract()) {

            return

                IERC721Receiver(to).onERC721Received(

                    msg.sender,

                    from,

                    tokenId,

                    \_data

                ) == IERC721Receiver.onERC721Received.selector;

        } else {

            return true;

        }

    }

    /\*\*

     \* 实现IERC721Metadata的tokenURI函数，查询metadata。

     \*/

    function tokenURI(uint256 tokenId)

        public

        view

        virtual

        override

        returns (string memory)

    {

        require(\_owners[tokenId] != address(0), "Token Not Exist");

        string memory baseURI = \_baseURI();

        // return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";

        return

            bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI)) : "";

    }

    /\*\*

     \* 计算{tokenURI}的BaseURI，tokenURI就是把baseURI和tokenId拼接在一起，需要开发重写。

     \* BAYC的baseURI为ipfs://QmeSjSinHpPnmXmspMjwiXyN6zS4E9zccariGR3jxcaWtq/

     \*/

    function \_baseURI() internal view virtual returns (string memory) {

        return "";

    }

}