**周亚男 2020131062**

## Vault的工作原理

将存入这个合约的币按比例分给不同用户，并提供兑换代币和赎回代币的方法

## 完善合约代码

// SPDX-License-Identifier: SEE LICENSE IN LICENSE

pragma solidity ^0.8.12;

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

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

import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";

contract ZYN is ERC20, Ownable, ERC20Burnable {

    constructor() ERC20("ZYN", "ZYN") {}

    function mint(address reciever, uint256 amount) public onlyOwner {

        \_mint(reciever, amount);

    }

    function \_burn(uint256 amount) public onlyOwner {

        burn(amount);

    }

}

// SPDX-License-Identifier: SEE LICENSE IN LICENSE

pragma solidity ^0.8.12;

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

contract Vault {

    // 用于与已部署的 ERC20 token 代币进行交互

    IERC20 public immutable token;

    uint public totalSupply;

    mapping(address => uint) public balanceOf;

    constructor(address \_token) {

        token = IERC20(\_token);

    }

    //合约内部函数 内部调用

    function \_mint(address \_to, uint \_amount) private {

        require(\_amount > 0);

        totalSupply += \_amount;

        balanceOf[\_to] += \_amount;

    }

    function \_burn(address \_from, uint \_amount) private {

        require(\_amount > 0 && \_amount <= totalSupply);

        require(balanceOf[\_from] >= \_amount);

        totalSupply -= \_amount;

        balanceOf[\_from] -= \_amount;

    }

    // 在 Deposit 函数中，通过计算当前用户所要存入的代币数量与合约总代币量的比例，

    // 得到当前用户应该增加多少份额，并将其相应地增加至 balanceOf 字典中；

    // 而在 withdraw 函数中，则需要计算出对应份额下所能取得的代币数量，然后将相应份额和代币转移给用户

    function deposit(uint \_amount) external {

        require(\_amount > 0);

        require(token.allowance(msg.sender, address(this)) >= \_amount);

        token.transferFrom(msg.sender, address(this), \_amount); // 将代币转移到合约地址

        \_mint(msg.sender, \_amount); // 给用户增加份额

    }

    function withdraw(uint \_shares) external {

        require(\_shares > 0 && \_shares <= balanceOf[msg.sender]);

        uint amount = ((\_shares \* token.balanceOf(address(this))) /

            totalSupply); // 计算出对应份额下的代币数量

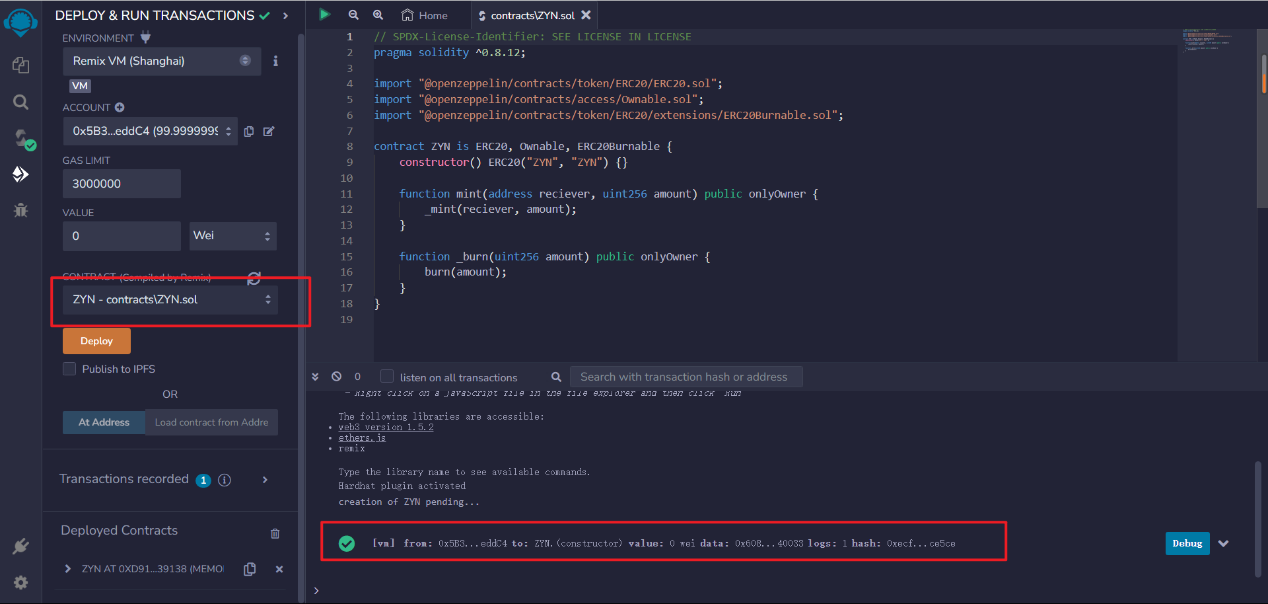
        \_burn(msg.sender, \_shares); // 将份额从用户账户中扣除

        token.transfer(msg.sender, amount); // 将对应数量的代币转移到用户账户

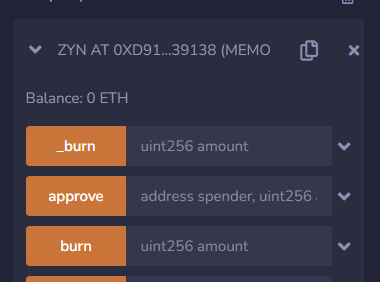
    }

}

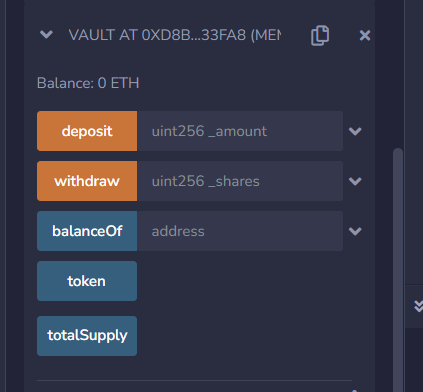
## 部署测试过程

1. 部署erc20  
   

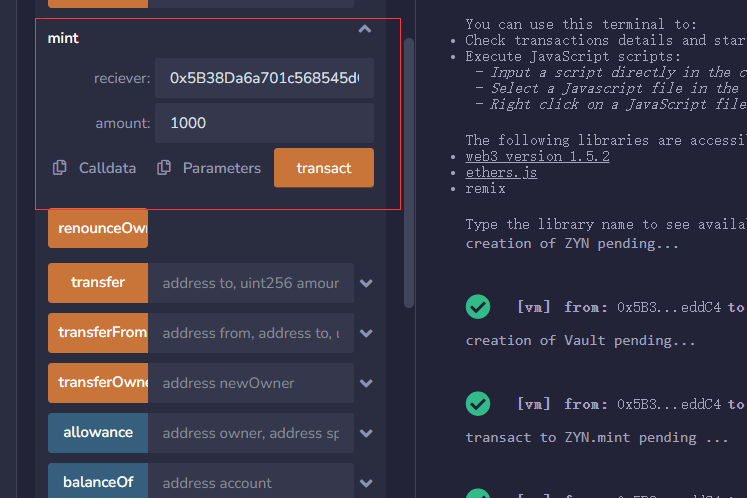
2. 部署以⾃⼰名字命名的 ERC20 合约；



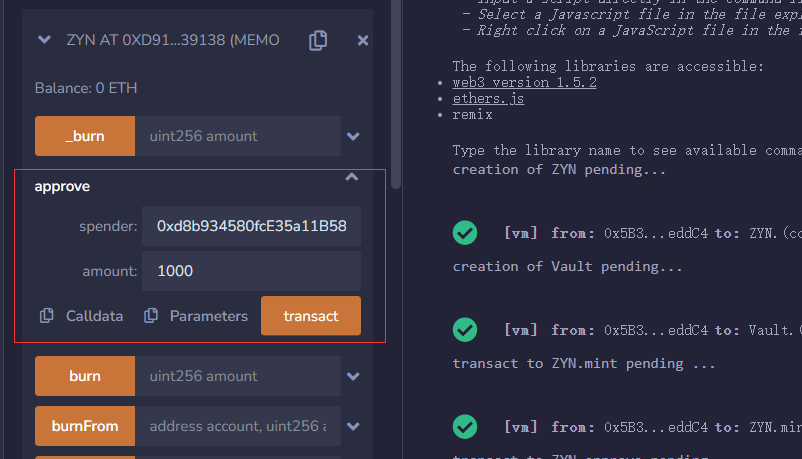
3. 部署 Vault 合约



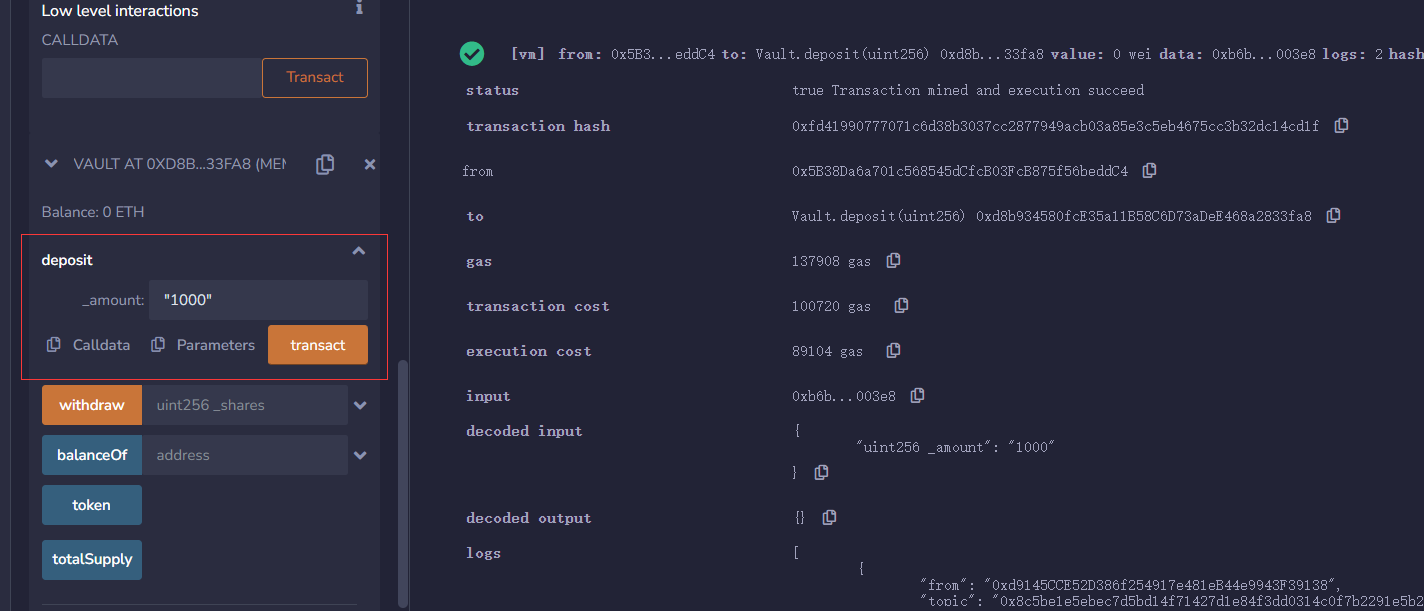
3. 给⾃⼰的地址增发 1000 个币



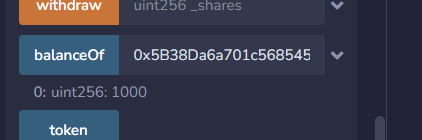
4. 在 ERC20 合约对 Vault 合约进⾏ approve



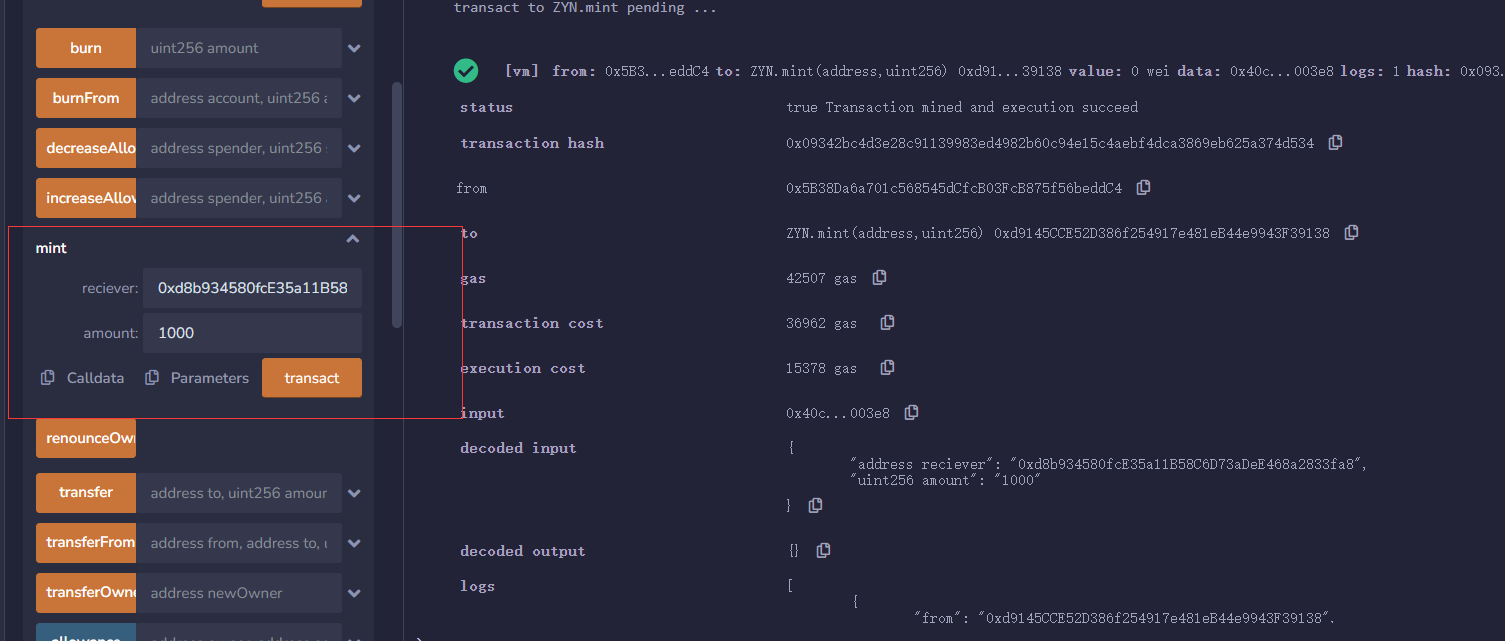
5. 在 Vault 合约充值



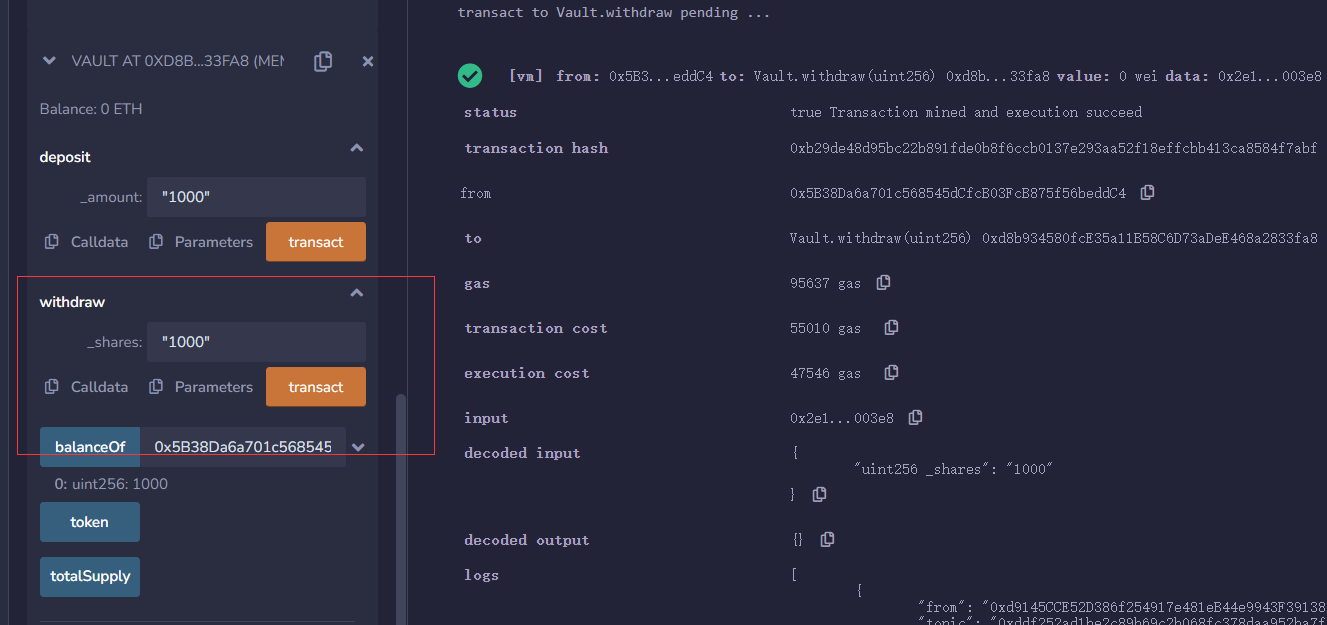
6. 查看⾃⼰的 share 持有量



7. 模拟盈利场景，对 Vault 合约增发 1000 个 ERC20



8. 提取全部 share



9. 查看⾃⼰地址的 ERC20 余额

