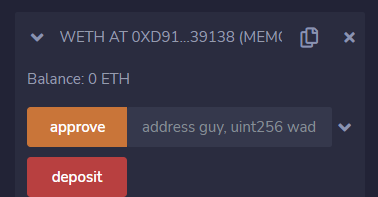
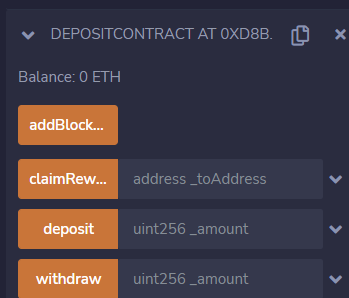
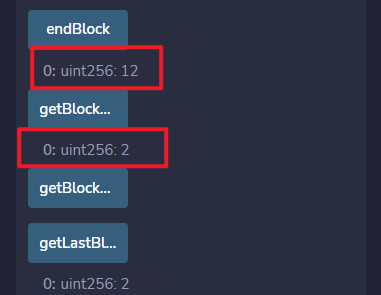
**周亚男 2020131062 202班**

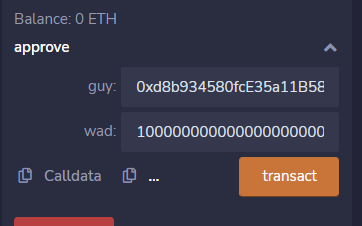
1. 部署WETH合约；  


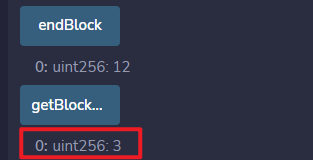
Weth合约地址是：0xd9145CCE52D386f254917e481eB44e9943F39138

2. 修改DepositContract合约里的\_weth地址为真实地址，部署 DepositContract合约，设置活动期为10个区块；  


DepositContract 合约地址为：0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8

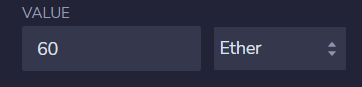
3. 查看当前区块高度和endBlock；  


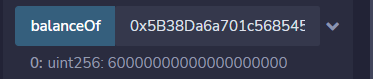
4. 在WETH合约里对DepositContract进行approve  


5. 查看此时区块高度  


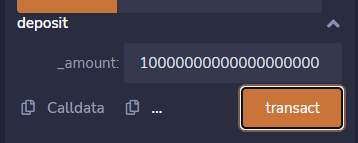
6. 充值50个WETH到WETH合约；

由于提取利息时会更新本金余额表，所所以我充值60个eth、

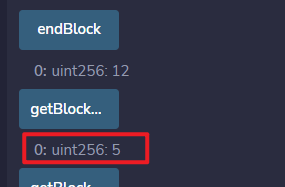




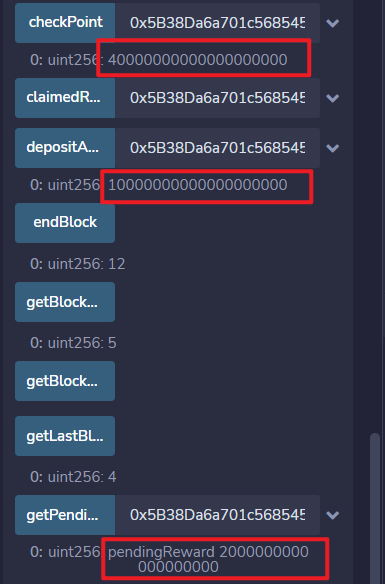
7. 充值10个WETH到DepositContract合约



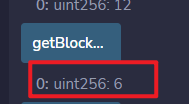
8. 查询当前区块高度



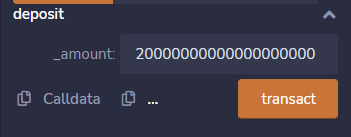
9. 调用DepositContract合约的相关方法，查看账户A的 checkPoint、depositAmount和getPendingReward



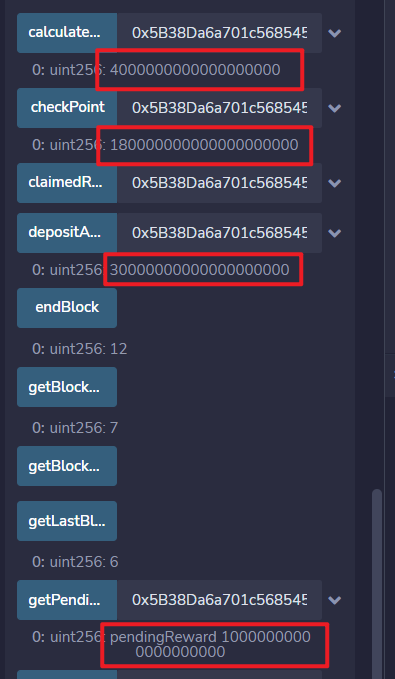
10. 调用DepositContract合约的addBlock方法，增加一个区块高 度



11. 充值20个WETH到DepositContract



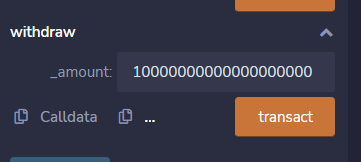
12. 调用DepositContract合约的相关方法，查看账户A的 calculatedReward、checkPoint、depositAmount和 getPendingReward



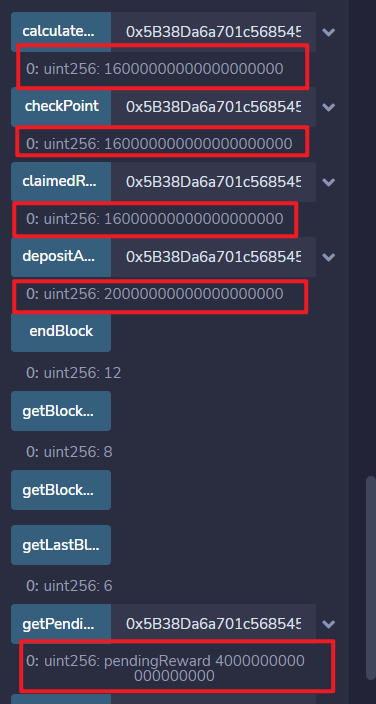
13. 调用DepositContract合约的addBlock方法，增加一个区块高 度。



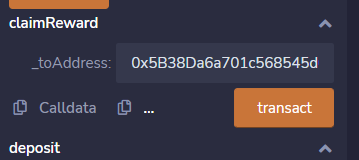
14. 调用DepositContract的withdraw方法，提取10个ETH本金



15. 查看账户A的 calculatedReward/checkPoint/depositAmount/getPendingR eward/claimedReward



16. 调用DepositContract的claimReward合约，提取利息到任意 地址；



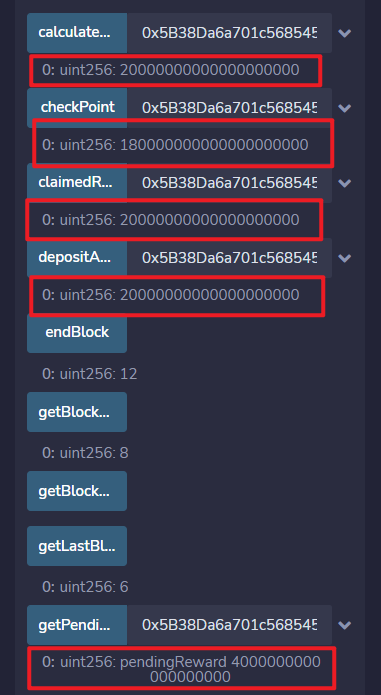
17. 查看当前链上状态

// 这一步不和梁老师一样 我不认为自己是错的，在不知道梁老师源码的情况下，

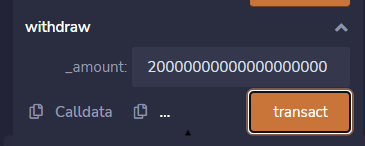
    // 我只能理解为，梁老师的这一步会重复提取同一块利息，是不符合生活场景的

    // 第二次思考：梁老师这一步是没有更新calculatedReward的，也就是没有记录提款这个点

    // 虽然不会重复提取利息，但是不符合生活使用场景



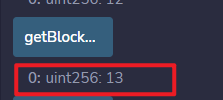
18. 调用DepositContract合约的withdraw方法，提取所有本金20 个ETH；



19. 查看此时的链上状态



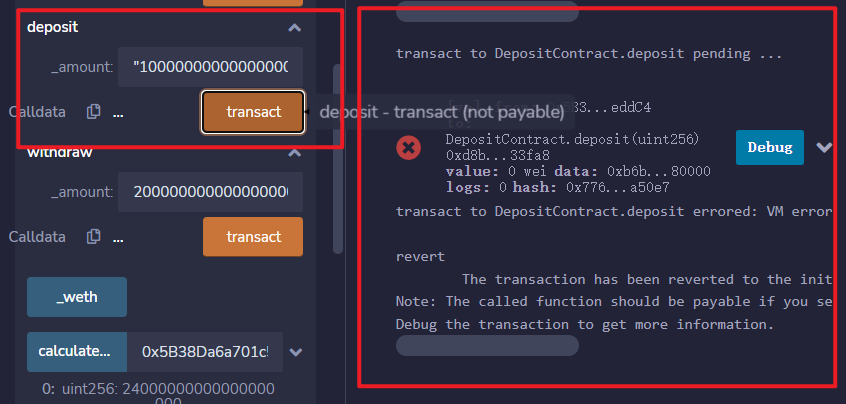
20. 调用DepositContract合约的addBlock方法，增加两个区块高 度



21. 查看此时链上状态



22. 调用DepositContract合约的deposit方法，会报错。因为已经 超过有效时间



1. 阐述constant和immutable的联系与区别

// constant常量 简单来说，常量不可改变，不能够再复值，修饰的变量需要在编译期确定值

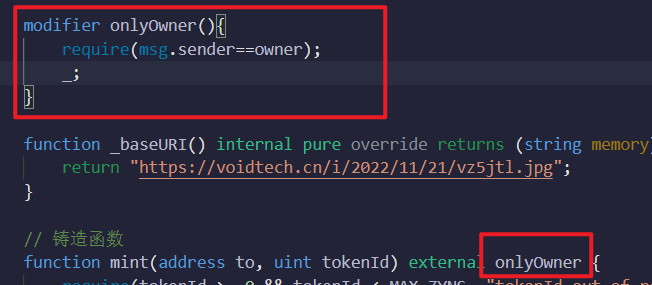
    // immutable 永恒的 不可改变的 ，部署的时候确定变量的值,

    // 它在构造函数中赋值一次之后,就不在改变, 这是一个运行时赋值, 就可以解除之前 constant 不支持使用运行时状态赋值的限制

    //immutable可以作为一次性使用变量，构造函数成立之后，他的值永不变化，升级版的constant

2. 阐述modifier的用法；

对某个函数起限制作用，用法如下：



标着mint函数的调用者必须是owner

**实验源码：**

Depositstart.sol

// SPDX-License-Identifier: MIT

pragma solidity 0.7.6;

import "./WETH.sol"; //  WETH9

import "./SafeMath.sol"; //SafeMath

contract DepositContract {

    using SafeMath for uint256;

    // 10个 10 00000000000000000

    // constant常量 简单来说，常量不可改变，不能够再复值，修饰的变量需要在编译期确定值

    // immutable 永恒的 不可改变的 ，部署的时候确定变量的值,

    // 它在构造函数中赋值一次之后,就不在改变, 这是一个运行时赋值, 就可以解除之前 constant 不支持使用运行时状态赋值的限制

    //immutable可以作为一次性使用变量，构造函数成立之后，他的值永不变化，升级版的constant

    address payable public immutable \_weth; // 替换为自己部署的 WETH 地址===》构造函数确定

    uint256 public constant rewardBase = 5; // 每5个币经过一个区块，可以领取1个ETH奖励。注意这里的奖励是ETH而不是WETH；

    uint256 public immutable startBlock; // 在构造函数中定义，起始区块

    uint256 public immutable endBlock; // 在构造函数中定义，结束区块

    mapping(address => uint256) public depositAmount; // 用户的存款总量

    // checkpoint（存取款的时刻点）：存款总量\*区块高度==》可以拿到区块高度

    mapping(address => uint256) public checkPoint; // 每次存款或提取本金时，更新这个值

    mapping(address => uint256) public lastBlockNumber;//上一次存取款高度

    // 下面两个数据相减，就是还剩多少利息

    mapping(address => uint256) public calculatedReward; // 已经计算的利息==》有多少利息，设计本金时触发

    mapping(address => uint256) public claimedReward; // 已经提取的利息==》提取了多少利息

    // 存本金

    event Deposit(address indexed sender, uint256 amount);

    // 取本金

    event Withdraw(address indexed sender, uint256 amount);

    // 取利息

    event Claim(address indexed sender, address recipient, uint256 amount);

    constructor(address payable \_wethAddress, uint256 \_period) {

        // period 为从当前开始，延续多少个区块

        startBlock = block.number;

        endBlock = block.number + \_period + 1;

        \_weth = \_wethAddress;

    }

        // 修饰符，充值时只允许在设定的区块范围内

        // 活动区块范围

    modifier onlyValidTime() {

        require(block.number<=endBlock && block.number>=startBlock);

        \_;

    }

        //上次存钱时间点（高度）

    function getLastBlockNumber() public view returns (uint256)  {

        if(depositAmount[msg.sender]==0){

            return getBlockNumber();

        }

        else{

            return SafeMath.div(checkPoint[msg.sender],depositAmount[msg.sender]);

        }

    }

        // 存钱到合约

    function deposit(uint256 \_amount) public onlyValidTime returns (bool)  {

        require(\_amount>0);

        lastBlockNumber[msg.sender]=getLastBlockNumber();

        //每次存本金时更新之前的利息==>上一次到这一次之间

        calculatedReward[msg.sender] = SafeMath.add(calculatedReward[msg.sender] , (block.number-lastBlockNumber[msg.sender])\*(depositAmount[msg.sender]/rewardBase));

        // //存钱到weth合约，前提是approve了，要不然账户a对此合约没有预存款

        WETH(\_weth).transferFrom(msg.sender,address(this),\_amount);

        // 函数外面可以查看存取款时刻

        depositAmount[msg.sender] = SafeMath.add(depositAmount[msg.sender],\_amount);

        checkPoint[msg.sender] = SafeMath.mul(depositAmount[msg.sender],block.number);

        emit Deposit(msg.sender, \_amount);

        return true;

    }

        //查看一下上次存取款到现在的块利息，不触发记账

    function getBlockReward() public onlyValidTime view returns(uint256) {

        return SafeMath.mul((block.number-getLastBlockNumber()),(depositAmount[msg.sender]/rewardBase));

    }

        // 当前位置的应有总利息

    function getPendingReward(address \_account)  public view returns (uint256 pendingReward) {

        return SafeMath.add(SafeMath.sub(calculatedReward[\_account],claimedReward[\_account]),getBlockReward());

    }

    // 这一步不和梁老师一样 我不认为自己是错的，在不知道梁老师源码的情况下，

    // 我只能理解为，梁老师的这一步会重复提取同一块利息，是不符合生活场景的

    // 第二次思考：梁老师这一步是没有更新calculatedReward的，也就是没有记录提款这个点

    // 虽然不会重复提取利息，但是不符合生活使用场景

    // 领取利息 一次性全部领取

    function claimReward(address payable \_toAddress) public returns (bool) {

        uint256  pendingReward = getPendingReward(msg.sender);

        WETH(\_weth).withdrawTo(\_toAddress,pendingReward);

        claimedReward[msg.sender] = SafeMath.add(claimedReward[msg.sender],pendingReward);

        // 不用+=，直接更新利息

        calculatedReward[msg.sender]= claimedReward[msg.sender];

        checkPoint[msg.sender] = SafeMath.mul(depositAmount[msg.sender],block.number);

        emit Claim(msg.sender, \_toAddress, pendingReward);

        return true;

    }

        // 提取一定数量的本金

    function withdraw(uint256 \_amount) public returns (bool) {

        require(\_amount>0&&depositAmount[msg.sender]>=\_amount);

        lastBlockNumber[msg.sender]=getLastBlockNumber();

        claimReward(msg.sender);

        WETH(\_weth).withdrawTo(msg.sender,\_amount);

        depositAmount[msg.sender] = SafeMath.sub(depositAmount[msg.sender],\_amount);

        checkPoint[msg.sender] = depositAmount[msg.sender]\*block.number;

        emit Withdraw(msg.sender, \_amount);

        return true;

    }

    // 以下不用改

    // 用于在Remix本地环境中增加区块高度

    uint256 counter;

    function addBlockNumber() public {

        counter++;

    }

    // 获取当前区块高度

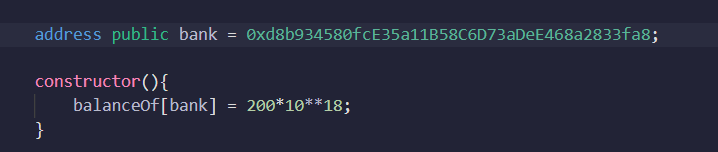
    function getBlockNumber() public view returns (uint256) {

        return block.number;

    }

}

Weth.sol合约修改部分：



这一步是防止利息+本金不够。由于系统是默认生成的地址，所以可以这样用。

本次实验主要是体验用sol制作活期存款的过程，个别细节部分请勿深究，谢谢！