



智能合约安全审计报告



慢雾安全团队于 2020-09-21 日，收到 LINA 团队对 LINA 项目智能合约安全审计申请。如下为本次智能合约安全审计细节及结果：

Token 名称：

LINA

合约地址：

0xeea988387700db4e163cd72d8d4a6994af31eeb1

0x32423a4c53cFB980264F804DF4500340Ca3318A8

0x3e9bc21c9b189c09df3ef1b824798658d5011937

0xa7e9da4851992b424bab4c8ae97689af69c654fa

0x38d47d313e70d0cbcf618adbb84b0da66d35ed5e

地址链接：

<https://etherscan.io/address/0xeea988387700db4e163cd72d8d4a6994af31eeb1#code>

<https://etherscan.io/address/0x32423a4c53cFB980264F804DF4500340Ca3318A8#code>

<https://etherscan.io/address/0x3e9bc21c9b189c09df3ef1b824798658d5011937#code>

<https://etherscan.io/address/0xa7e9da4851992b424bab4c8ae97689af69c654fa#code>

<https://etherscan.io/address/0x38d47d313e70d0cbcf618adbb84b0da66d35ed5e#code>

本次审计项及结果：

（其他未知安全漏洞不包含在本次审计责任范围）

序号	审计大类	审计子类	审计结果
1	溢出审计	-	通过
2	条件竞争审计	-	通过
3	权限控制审计	权限漏洞审计	通过
		权限过大审计	有风险
4	安全设计审计	Zeppelin 模块使用安全	通过
		编译器版本安全	通过
		硬编码地址安全	通过
		Fallback 函数使用安全	通过
		显现编码安全	通过

		函数返回值安全	通过
		call 调用安全	通过
5	拒绝服务审计	-	通过
6	Gas 优化审计	-	通过
7	设计逻辑审计	-	通过
8	“假充值”漏洞审计	-	通过
9	恶意 Event 事件日志审计	-	通过
10	变量声明及作用域审计	-	通过
11	重放攻击审计	ECDSA 签名重放审计	通过
12	未初始化的存储指针	-	通过
13	算术精度误差	-	通过

备注：审计意见及建议见代码注释 //SlowMist//……

审计结果：**有风险**

审计编号：0X002009260002

审计日期：2020 年 09 月 26 日

审计团队：慢雾安全团队

(声明：慢雾仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，慢雾无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料（简称“已提供资料”）。慢雾假设：已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，慢雾对由此而导致的损失和不利影响不承担任何责任。慢雾仅对该项目的安全情况进行约定内的安全审计并出具了本报告，慢雾不对该项目背景及其他情况进行负责。)

总结：此为代币(token)合约。合约代币总量可变，Admin 可铸币和燃烧任意账户代币，合约有限制代币总量。

当前时间大于 locktime 时，Admin 可以转走 Staking 合约中的 Token。Admin 可以通过修改 LnTokenStorage 或 LnTokenStorageLock 的 onlyOperator 权限然后调用 setBalanceOf 修改任意账户余额而总量保持不变。Admin 可以通过修改 accessCtrl 权限，任意增加和减少用户抵押资产。经与项目方沟通，将 LnSimpleStaking 合约的 admin 设置为多签，locktime 需要大于当前时间 2 天。建议监控

Translock 的事件，可以在非预期的情况发生时候及时知晓，其他合约的 admin 设置为 LnEndAdmin，且 LnEndAdmin 无法调用除 becomeAdmin 外的其他函数。使用了 SafeMath 安全模块，值得称赞的做法。

合约不存在溢出、条件竞争问题。

合约源代码如下：

IERC20.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.24;

interface IERC20 {

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view returns (uint8);

    function totalSupply() external view returns (uint);

    function balanceOf(address owner) external view returns (uint);

    function allowance(address owner, address spender) external view returns (uint);

    function transfer(address to, uint value) external returns (bool);

    function approve(address spender, uint value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint value
    ) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint value);

    event Approval(address indexed owner, address indexed spender, uint value);
}
```

LinearFinanceToken.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.12;

import "./LnAdmin.sol";

import "./LnTokenStorage.sol";

import "./LnErc20Handler.sol";

import "./LnOperatorModifier.sol";

import "@openzeppelin/contracts/math/SafeMath.sol";

import "@openzeppelin/contracts/utils/Pausable.sol";

contract LinearFinance is LnErc20Handler {

string public constant TOKEN_NAME = "Linear Token";

string public constant TOKEN_SYMBOL = "LINA";

uint8 public constant DECIMALS = 18;

uint256 public constant MAX_SUPPLY = 10000000000e18;

constructor(

address payable _proxy,

LnTokenStorage _tokenStorage,

address _admin,

uint _totalSupply

)

public

LnErc20Handler(_proxy, _tokenStorage, TOKEN_NAME, TOKEN_SYMBOL, _totalSupply, DECIMALS, _admin)

{

}

//

function _mint(address account, uint256 amount) private {

require(account != address(0), "ERC20: mint to the zero address"); //SlowMist// 这类检查很好，避免失误导致

Token 丢失

require(totalSupply.add(amount) <= MAX_SUPPLY, "Can not mint over max supply"); //SlowMist// 限制最大铸币

数量

_beforeTokenTransfer(address(0), account, amount);

```
tokenStorage.setBalanceOf(account, tokenStorage.balanceOf(account).add(amount));
totalSupply = totalSupply.add(amount);

emitTransfer(address(0), account, amount);
}

function mint(address account, uint256 amount) external onlyAdmin {
    _mint(account, amount);
}

function _burn(address account, uint256 amount) private {
    require(account != address(0), "ERC20: burn from the zero address");
    _beforeTokenTransfer(account, address(0), amount);

    tokenStorage.setBalanceOf(account, tokenStorage.balanceOf(account).sub(amount));
    totalSupply = totalSupply.sub(amount);
    emitTransfer(account, address(0), amount);
}

function burn(address account, uint256 amount) external onlyAdmin {
    _burn(account, amount);
}

function _beforeTokenTransfer(address from, address to, uint256 amount) internal override {
    super._beforeTokenTransfer(from, to, amount);

    require(!paused, "ERC20Pausable: token transfer while paused");
}

//////////////////// paused
```

//SlowMist// 在出现重大交易异常时可以暂停所有交易，值得称赞的做法

```
bool public paused = false;
modifier notPaused {
    require(!paused, "This action cannot be performed while the contract is paused");
    _;
}

function setPaused(bool _paused) external onlyAdmin {
    if (_paused == paused) {
        return;
    }
}
```

```
        paused = _paused;
        emit PauseChanged(paused);
    }

    //////////////////////////////////////

    event PauseChanged(bool isPaused);
}
```

LnAccessControl.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/utils/Address.sol";

// example:
//LnAccessControl accessCtrl = LnAccessControl(addressStorage.getAddress("LnAccessControl"));
//require(accessCtrl.hasRole(accessCtrl.DEBT_SYSTEM(), _address), "Need debt system access role");

// contract access control
contract LnAccessControl is AccessControl {
    using Address for address;

    // -----
    // role type
    bytes32 public constant ISSUE_ASSET_ROLE = ("ISSUE_ASSET"); //keccak256
    bytes32 public constant BURN_ASSET_ROLE = ("BURN_ASSET");

    bytes32 public constant DEBT_SYSTEM = ("LnDebtSystem");
    // -----

    constructor(address admin) public {
        _setupRole(DEFAULT_ADMIN_ROLE, admin);
    }

    function IsAdmin(address _address) public view returns (bool) {
        return hasRole(DEFAULT_ADMIN_ROLE, _address);
    }

    function SetAdmin(address _address) public returns (bool) {
        require(IsAdmin(msg.sender), "Only admin");
    }
}
```

```
_setupRole(DEFAULT_ADMIN_ROLE, _address);
}

// -----
// this func need admin role. grantRole and revokeRole need admin role
function SetRoles(bytes32 roleType, address[] calldata addresses, bool[] calldata setTo) external {
    require(IsAdmin(msg.sender), "Only admin");

    _setRoles(roleType, addresses, setTo);
}

function _setRoles(bytes32 roleType, address[] calldata addresses, bool[] calldata setTo) private {
    require(addresses.length == setTo.length, "parameter address length not eq");

    for (uint256 i=0; i < addresses.length; i++) {
        //require(addresses[i].isContract(), "Role address need contract only");
        if (setTo[i]) {
            grantRole(roleType, addresses[i]);
        } else {
            revokeRole(roleType, addresses[i]);
        }
    }
}

// function SetRoles(bytes32 roleType, address[] calldata addresses, bool[] calldata setTo) public {
//     _setRoles(roleType, addresses, setTo);
// }

// Issue burn
function SetIssueAssetRole(address[] calldata issuer, bool[] calldata setTo) public {
    _setRoles(ISSUE_ASSET_ROLE, issuer, setTo);
}

function SetBurnAssetRole(address[] calldata burner, bool[] calldata setTo) public {
    _setRoles(BURN_ASSET_ROLE, burner, setTo);
}

//
function SetDebtSystemRole(address[] calldata _address, bool[] calldata _setTo) public {
    _setRoles(DEBT_SYSTEM, _address, _setTo);
}
```



```
}
```

LnAdmin.sol

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.6.12;
```

```
contract LnAdmin {
```

```
    address public admin;
```

```
    address public candidate;
```

```
    constructor(address _admin) public {
```

```
        require(_admin != address(0), "admin address cannot be 0");
```

```
        admin = _admin;
```

```
        emit AdminChanged(address(0), _admin);
```

```
    }
```

```
    function setCandidate(address _candidate) external onlyAdmin {
```

```
        address old = candidate;
```

```
        candidate = _candidate;
```

```
        emit candidateChanged( old, candidate);
```

```
    }
```

```
    function becomeAdmin( ) external {
```

```
        require( msg.sender == candidate, "Only candidate can become admin");
```

```
        address old = admin;
```

```
        admin = candidate;
```

```
        emit AdminChanged( old, admin );
```

```
    }
```

```
    modifier onlyAdmin {
```

```
        require( (msg.sender == admin), "Only the contract admin can perform this action");
```

```
        _;
```

```
    }
```

```
    event candidateChanged(address oldCandidate, address newCandidate );
```

```
    event AdminChanged(address oldAdmin, address newAdmin);
```

```
}
```

LnErc20Handler.sol

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.6.12;
```

```
import "./IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "./SafeDecimalMath.sol";

import "./LnAdmin.sol";
import "./LnProxyImpl.sol";
import "./LnTokenStorage.sol";

contract LnErc20Handler is IERC20, LnAdmin, LnProxyImpl {
    using SafeMath for uint;
    using SafeDecimalMath for uint;

    LnTokenStorage public tokenStorage;

    string public override name;
    string public override symbol;
    uint public override totalSupply;
    uint8 public override decimals;

    constructor( address payable _proxy, LnTokenStorage _tokenStorage, string memory _name,
        string memory _symbol, uint _totalSupply, uint8 _decimals, address _admin )
        public LnAdmin(_admin) LnProxyImpl(_proxy) {

        tokenStorage = _tokenStorage;
        name = _name;
        symbol = _symbol;
        totalSupply = _totalSupply;
        decimals = _decimals;
    }

    function allowance(address owner, address spender) public view virtual override returns (uint) {
        return tokenStorage.allowance(owner, spender);
    }

    function balanceOf(address account) external view override returns (uint) {
        return tokenStorage.balanceOf(account);
    }

    function setTokenStorage(LnTokenStorage _tokenStorage) external optionalProxy_onlyAdmin {
        tokenStorage = _tokenStorage;
    }
}
```

```
emitTokenStorageUpdated(address(tokenStorage));
}

function _internalTransfer( address from, address to, uint value ) internal returns (bool) {

    require(to != address(0) && to != address(this) && to != address(proxy), "Cannot transfer to this address");
    _beforeTokenTransfer(from, to, value);

    tokenStorage.setBalanceOf(from, tokenStorage.balanceOf(from).sub(value));
    tokenStorage.setBalanceOf(to, tokenStorage.balanceOf(to).add(value));

    emitTransfer(from, to, value);

    return true;
}

function _transferByProxy(
    address from,
    address to,
    uint value
) internal returns (bool) {
    return _internalTransfer(from, to, value);
}

function _transferFromByProxy(
    address sender,
    address from,
    address to,
    uint value
) internal returns (bool) {

    tokenStorage.setAllowance(from, sender, tokenStorage.allowance(from, sender).sub(value));
    return _internalTransfer(from, to, value);
}

function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }

// default transfer
function transfer(address to, uint value) external virtual override optionalProxy returns (bool) {
    _transferByProxy(messageSender, to, value);
}
```

```
    return true; //SlowMist// 返回值符合 EIP20 规范
}

// default transferFrom
function transferFrom(
    address from,
    address to,
    uint value
) external virtual override optionalProxy returns (bool) {
    return _transferFromByProxy(messageSender, from, to, value);
}

function approve(address spender, uint value) public virtual override optionalProxy returns (bool) {
    address sender = messageSender;

    tokenStorage.setAllowance(sender, spender, value);
    emitApproval(sender, spender, value);

    return true; //SlowMist// 返回值符合 EIP20 规范
}

function addressToBytes32(address input) internal pure returns (bytes32) {
    return bytes32(uint256(uint160(input)));
}

event Transfer(address indexed from, address indexed to, uint value);
bytes32 internal constant TRANSFER_SIG = keccak256("Transfer(address,address,uint256)");

function emitTransfer(
    address from,
    address to,
    uint value
) internal {
    proxy.Log3( abi.encode(value), TRANSFER_SIG, addressToBytes32(from), addressToBytes32(to) );
}

event Approval(address indexed owner, address indexed spender, uint value);
bytes32 internal constant APPROVAL_SIG = keccak256("Approval(address,address,uint256)");
```

```
function emitApproval(
    address owner,
    address spender,
    uint value
) internal {
    proxy.Log3( abi.encode(value), APPROVAL_SIG, addressToBytes32(owner), addressToBytes32(spender) );
}

event TokenStorageUpdated(address newTokenStorage);
bytes32 internal constant TOKENSTORAGE_UPDATED_SIG = keccak256("TokenStorageUpdated(address)");

function emitTokenStorageUpdated(address newTokenStorage) internal {
    proxy.Log1( abi.encode(newTokenStorage), TOKENSTORAGE_UPDATED_SIG );
}
}
```

LnLinearStaking.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "./IERC20.sol";
import "./LnAdmin.sol";
import "./LnOperatorModifier.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Pausable.sol";
import "./LnAccessControl.sol";

interface ILinearStaking {
    function staking(uint256 amount) external returns (bool);
    function cancelStaking(uint256 amount) external returns (bool);
    function claim() external returns (bool);
    function stakingBalanceOf(address account) external view returns(uint256);
}

contract LnLinearStakingStorage is LnAdmin {
    using SafeMath for uint256;

    LnAccessControl public accessCtrl;

    bytes32 public constant DATA_ACCESS_ROLE = "LinearStakingStorage";
}
```

```
struct StakingData {
    uint256 amount;
    uint256 staketime;
}

mapping (address => StakingData[]) public stakesdata;
mapping (uint256 => uint256) public weeksTotal; // week staking amount

uint256 public stakingStartTime = 1600329600; // TODO: UTC or UTC+8
uint256 public stakingEndTime = 1605168000;
uint256 public totalWeekNumber = 8;
uint256 public weekRewardAmount = 18750000e18;

constructor(address _admin, address _accessCtrl) public LnAdmin(_admin) {
    accessCtrl = LnAccessControl(_accessCtrl);
}

modifier OnlyLinearStakingStorageRole(address _address) {
    require(accessCtrl.hasRole(DATA_ACCESS_ROLE, _address), "Only Linear Staking Storage Role");
    _;
}

function setAccessControl(address _accessCtrl) external onlyAdmin {
    accessCtrl = LnAccessControl(_accessCtrl);
}

function weekTotalStaking() public view returns (uint256[] memory) {
    uint256[] memory totals = new uint256[](totalWeekNumber);
    for (uint256 i=0; i< totalWeekNumber; i++) {
        uint256 delta = weeksTotal[i];
        if (i == 0) {
            totals[i] = delta;
        } else {
            totals[i] = totals[i-1].add(delta);
        }
    }
    return totals;
}

function getStakesdataLength(address account) external view returns(uint256) {
```

```
    return stakesdata[account].length;
}

function getStakesDataByIndex(address account, uint256 index) external view returns(uint256, uint256) {
    return (stakesdata[account][index].amount, stakesdata[account][index].staketime);
}

function stakingBalanceOf(address account) external view returns(uint256) {
    uint256 total = 0;
    StakingData[] memory stakes = stakesdata[account];
    for (uint256 i=0; i < stakes.length; i++) {
        total = total.add(stakes[i].amount);
    }
    return total;
}

function requireInStakingPeriod() external view {
    require(stakingStartTime < block.timestamp, "Staking not start");
    require(block.timestamp < stakingEndTime, "Staking stage has end.");
}

function requireStakingEnd() external view {
    require(block.timestamp > stakingEndTime, "Need wait to staking end");
}

function PushStakingData(address account, uint256 amount, uint256 staketime) external
OnlyLinearStakingStorageRole(msg.sender) {
    LnLinearStakingStorage.StakingData memory data = LnLinearStakingStorage.StakingData({
        amount: amount,
        staketime: staketime
    });
    stakesdata[account].push(data);
}

//SlowMist//Admin 可以通过修改 accessCtrl 权限，任意增用户抵押资产

function StakingDataAdd(address account, uint256 index, uint256 amount) external
OnlyLinearStakingStorageRole(msg.sender) {
    stakesdata[account][index].amount = stakesdata[account][index].amount.add(amount);
}

//SlowMist//Admin 可以通过修改 accessCtrl 权限，任意减少用户抵押资产
```

```
function StakingDataSub(address account, uint256 index, uint256 amount) external
OnlyLinearStakingStorageRole(msg.sender) {
    stakesdata[account][index].amount = stakesdata[account][index].amount.sub(amount, "StakingDataSub sub
overflow");
}

function DeleteStakesData(address account) external OnlyLinearStakingStorageRole(msg.sender) {
    delete stakesdata[account];
}

function PopStakesData(address account) external OnlyLinearStakingStorageRole(msg.sender) {
    stakesdata[account].pop();
}

function AddWeeksTotal(uint256 staketime, uint256 amount) external OnlyLinearStakingStorageRole(msg.sender) {
    uint256 weekNumber = staketime.sub(stakingStartTime, "AddWeeksTotal sub overflow") / 1 weeks;
    weeksTotal[weekNumber] = weeksTotal[weekNumber].add(amount);
}

function SubWeeksTotal(uint256 staketime, uint256 amount) external OnlyLinearStakingStorageRole(msg.sender) {
    uint256 weekNumber = staketime.sub(stakingStartTime, "SubWeeksTotal weekNumber sub overflow") / 1 weeks;
    weeksTotal[weekNumber] = weeksTotal[weekNumber].sub(amount, "SubWeeksTotal weeksTotal sub overflow");
}

function setWeekRewardAmount(uint256 _weekRewardAmount) external onlyAdmin {
    weekRewardAmount = _weekRewardAmount;
}

function setStakingPeriod(uint _stakingStartTime, uint _stakingEndTime) external onlyAdmin {
    require(_stakingEndTime > _stakingStartTime);

    stakingStartTime = _stakingStartTime;
    stakingEndTime = _stakingEndTime;

    totalWeekNumber = stakingEndTime.sub(stakingStartTime, "setStakingPeriod totalWeekNumber sub overflow") / 1
weeks;
    if (stakingEndTime.sub(stakingStartTime, "setStakingPeriod stakingEndTime sub overflow") % 1 weeks != 0) {
        totalWeekNumber = totalWeekNumber.add(1);
    }
}
}
```



```
contract LnLinearStaking is LnAdmin, Pausable, ILinearStaking {
    using SafeMath for uint256;

    IERC20 public linaToken; // lina token proxy address
    LnLinearStakingStorage public stakingStorage;

    constructor(
        address _admin,
        address _linaToken,
        address _storage
    ) public LnAdmin(_admin) {
        linaToken = IERC20(_linaToken);
        stakingStorage = LnLinearStakingStorage(_storage);
    }

    function setLinaToken(address _linaToken) external onlyAdmin {
        linaToken = IERC20(_linaToken);
    }

    function setPaused(bool _paused) external onlyAdmin {
        if (_paused) {
            _pause();
        } else {
            _unpause();
        }
    }

    ////////////////////////////////////

    event Staking(address indexed who, uint256 value, uint staketime);
    event CancelStaking(address indexed who, uint256 value);
    event Claim(address indexed who, uint256 rewardval, uint256 totalStaking);

    uint256 public accountStakingListLimit = 50;
    uint256 public minStakingAmount = 1e18; // 1 token
    uint256 public constant PRECISION_UINT = 1e23;

    function setLinaTokenAddress(address _token) external onlyAdmin {
        linaToken = IERC20(_token);
    }
}
```

```
function setStakingListLimit(uint256 _limit) external onlyAdmin {
    accountStakingListLimit = _limit;
}

function setMinStakingAmount(uint256 _minStakingAmount) external onlyAdmin {
    minStakingAmount = _minStakingAmount;
}

function stakingBalanceOf(address account) external override view returns(uint256) {
    return stakingStorage.stakingBalanceOf(account);
}

function getStakesdataLength(address account) external view returns(uint256) {
    return stakingStorage.getStakesdataLength(account);
}

//-----

function staking(uint256 amount) public whenNotPaused override returns (bool) {
    stakingStorage.requireInStakingPeriod();

    require(amount >= minStakingAmount, "Staking amount too small.");
    require(stakingStorage.getStakesdataLength(msg.sender) < accountStakingListLimit, "Staking list out of limit.");

    //linaToken.burn(msg.sender, amount);
    linaToken.transferFrom(msg.sender, address(this), amount);

    stakingStorage.PushStakingData(msg.sender, amount, block.timestamp);
    stakingStorage.AddWeeksTotal(block.timestamp, amount);

    emit Staking(msg.sender, amount, block.timestamp);
    return true;
}

function cancelStaking(uint256 amount) public whenNotPaused override returns (bool) {
    stakingStorage.requireInStakingPeriod();

    require(amount > 0, "Invalid amount.");

    uint256 returnToken = amount;
    for (uint256 i = stakingStorage.getStakesdataLength(msg.sender); i >= 1 ; i--) {
        (uint256 stakingAmount, uint256 staketime) = stakingStorage.getStakesDataByIndex(msg.sender, i-1);
```

```
if (amount >= stakingAmount) {
    amount = amount.sub(stakingAmount, "cancelStaking sub overflow");

    stakingStorage.PopStakesData(msg.sender);
    stakingStorage.SubWeeksTotal(staketime, stakingAmount);
} else {
    stakingStorage.StakingDataSub(msg.sender, i-1, amount);
    stakingStorage.SubWeeksTotal(staketime, amount);

    amount = 0;
}
if (amount == 0) break;
}
require(amount == 0, "Cancel amount too big then staked.");

//linaToken.mint(msg.sender, returnToken);
linaToken.transfer(msg.sender, returnToken);

emit CancelStaking(msg.sender, returnToken);

return true;
}

// claim reward
// Note: 需要提前把奖励 token 转进来
function claim() public whenNotPaused override returns (bool) {
    stakingStorage.requireStakingEnd();

    require(stakingStorage.getStakesdataLength(msg.sender) > 0, "Nothing to claim");

    uint256 totalWeekNumber = stakingStorage.totalWeekNumber();

    uint256 totalStaking = 0;
    uint256 totalReward = 0;

    uint256[] memory finalTotals = stakingStorage.weekTotalStaking();
    for (uint256 i=0; i < stakingStorage.getStakesdataLength(msg.sender); i++) {
        (uint256 stakingAmount, uint256 staketime) = stakingStorage.getStakesDataByIndex(msg.sender, i);
        uint256 stakedWeedNumber = staketime.sub(stakingStorage.stakingStartTime(), "claim sub overflow") / 1 weeks;

        totalStaking = totalStaking.add(stakingAmount);
```

```
uint256 reward = 0;
for (uint256 j=stakedWeedNumber; j < totalWeekNumber; j++) {
    reward = reward.add( stakingAmount.mul(PRECISION_UINT).div(finalTotals[j]) );
    //move .mul(weekRewardAmount) to next line.
}
reward = reward.mul(stakingStorage.weekRewardAmount()).div(PRECISION_UINT);

totalReward = totalReward.add( reward );
}

stakingStorage.DeleteStakesData(msg.sender);

//linaToken.mint(msg.sender, totalStaking.add(totalReward) );
linaToken.transfer(msg.sender, totalStaking.add(totalReward) );

emit Claim(msg.sender, totalReward, totalStaking);
return true;
}
}
```

LnOperatorModifier.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "./LnAdmin.sol";

abstract contract LnOperatorModifier is LnAdmin {

    address public operator;

    constructor(address _operator) internal {
        require(admin != address(0), "admin must be set");

        operator = _operator;
        emit OperatorUpdated(_operator);
    }

    function setOperator(address _operator) external onlyAdmin {
        operator = _operator;
        emit OperatorUpdated(_operator);
    }
}
```

```
}

modifier onlyOperator() {
    require(msg.sender == operator, "Only operator can perform this action");
    _;
}

event OperatorUpdated(address operator);
}
```

LnProxyERC20.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "./LnProxyImpl.sol";
import "./IERC20.sol";

contract LnProxyERC20 is LnProxyBase, IERC20 {
    constructor(address _admin) public LnProxyBase(_admin) {}

    function name() public view override returns (string memory) {

        return IERC20(address(target)).name();
    }

    function symbol() public view override returns (string memory) {

        return IERC20(address(target)).symbol();
    }

    function decimals() public view override returns (uint8) {

        return IERC20(address(target)).decimals();
    }

    function totalSupply() public view override returns (uint256) {

        return IERC20(address(target)).totalSupply();
    }

    function balanceOf(address account) public view override returns (uint256) {
```

```
    return IERC20(address(target)).balanceOf(account);
}

function allowance(address owner, address spender) public view override returns (uint256) {

    return IERC20(address(target)).allowance(owner, spender);
}

function transfer(address to, uint256 value) public override returns (bool) {

    target.setMessageSender(msg.sender);

    IERC20(address(target)).transfer(to, value);

    return true;
}

function approve(address spender, uint256 value) public override returns (bool) {

    target.setMessageSender(msg.sender);

    IERC20(address(target)).approve(spender, value);

    return true;
}

function transferFrom(
    address from,
    address to,
    uint256 value
) public override returns (bool) {

    target.setMessageSender(msg.sender);

    IERC20(address(target)).transferFrom(from, to, value);

    return true;
}
}
```

LnProxyImpl.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "./LnAdmin.sol";

contract LnProxyBase is LnAdmin {
    LnProxyImpl public target;

    constructor(address _admin) public LnAdmin(_admin) {}

    function setTarget(LnProxyImpl _target) external onlyAdmin {
        target = _target;
        emit TargetUpdated(_target);
    }

    function Log0( bytes calldata callData ) external onlyTarget {
        uint size = callData.length;
        bytes memory _callData = callData;
        assembly {
            log0(add(_callData, 32), size)
        }
    }

    function Log1( bytes calldata callData, bytes32 topic1 ) external onlyTarget {
        uint size = callData.length;
        bytes memory _callData = callData;
        assembly {
            log1(add(_callData, 32), size, topic1 )
        }
    }

    function Log2( bytes calldata callData, bytes32 topic1, bytes32 topic2 ) external onlyTarget {
        uint size = callData.length;
        bytes memory _callData = callData;
        assembly {
            log2(add(_callData, 32), size, topic1, topic2 )
        }
    }

    function Log3( bytes calldata callData, bytes32 topic1, bytes32 topic2, bytes32 topic3 ) external onlyTarget {
```

```
uint size = callData.length;
bytes memory _callData = callData;
assembly {
    log3(add(_callData, 32), size, topic1, topic2, topic3 )
}
}
```

```
function Log4( bytes calldata callData, bytes32 topic1, bytes32 topic2, bytes32 topic3, bytes32 topic4 ) external
onlyTarget {
    uint size = callData.length;
    bytes memory _callData = callData;
    assembly {
        log4(add(_callData, 32), size, topic1, topic2, topic3, topic4 )
    }
}
```

//receive: It is executed on a call to the contract with empty calldata. This is the function that is executed on plain Ether transfers (e.g. via .send() or .transfer()).

//fallback: can only rely on 2300 gas being available,

```
receive() external payable {
    target.setMessageSender(msg.sender);

    assembly {
        let free_ptr := mload(0x40)
        calldatacopy(free_ptr, 0, calldatasize())

        let result := call(gas(), sload(target_slot), callvalue(), free_ptr, calldatasize(), 0, 0)
        returndatacopy(free_ptr, 0, returndatasize())

        if iszero(result) {
            revert(free_ptr, returndatasize())
        }
        return(free_ptr, returndatasize())
    }
}

modifier onlyTarget {
    require(LnProxyImpl(msg.sender) == target, "Must be proxy target");
    _;
}
```



```
event TargetUpdated(LnProxyImpl newTarget);
}

abstract contract LnProxyImpl is LnAdmin {

    LnProxyBase public proxy;
    LnProxyBase public integrationProxy;

    address public messageSender;

    constructor(address payable _proxy) internal {

        require(admin != address(0), "Admin must be set");

        proxy = LnProxyBase(_proxy);
        emit ProxyUpdated(_proxy);
    }

    function setProxy(address payable _proxy) external onlyAdmin {
        proxy = LnProxyBase(_proxy);
        emit ProxyUpdated(_proxy);
    }

    function setIntegrationProxy(address payable _integrationProxy) external onlyAdmin {
        integrationProxy = LnProxyBase(_integrationProxy);
    }

    function setMessageSender(address sender) external onlyProxy {
        messageSender = sender;
    }

    modifier onlyProxy {
        require(LnProxyBase(msg.sender) == proxy || LnProxyBase(msg.sender) == integrationProxy, "Only the proxy can call");
        _;
    }

    modifier optionalProxy {
        if (LnProxyBase(msg.sender) != proxy && LnProxyBase(msg.sender) != integrationProxy && messageSender != msg.sender) {
```

```
        messageSender = msg.sender;
    }
    _;
}

modifier optionalProxy_onlyAdmin {
    if (LnProxyBase(msg.sender) != proxy && LnProxyBase(msg.sender) != integrationProxy && messageSender !=
msg.sender) {
        messageSender = msg.sender;
    }
    require(messageSender == admin, "only for admin");
    _;
}

event ProxyUpdated(address proxyAddress);
}
```

LnSimpleStaking.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "./IERC20.sol";
import "./LnAdmin.sol";
import "./LnOperatorModifier.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Pausable.sol";
import "./LnAccessControl.sol";
import "./LnLinearStaking.sol";
import "./SafeDecimalMath.sol";

contract LnRewardCalculator {
    using SafeMath for uint256;

    struct UserInfo {
        uint256 reward;
        uint256 amount;
        uint256 rewardDebt;
    }

    struct PoolInfo {
```

```
uint256 amount;
uint256 lastRewardBlock;
uint256 accRewardPerShare;
}

uint256 public rewardPerBlock;

PoolInfo public mPoolInfo;
mapping (address => UserInfo) public userInfo;

uint256 public startBlock;
uint256 public remainReward;
uint256 public accReward;

constructor( uint256 _rewardPerBlock, uint256 _startBlock ) public {
    rewardPerBlock = _rewardPerBlock;
    startBlock = _startBlock;
    mPoolInfo.lastRewardBlock = startBlock;
}

function _calcReward( uint256 curBlock, address _user) internal view returns (uint256) {
    PoolInfo storage pool = mPoolInfo;
    UserInfo storage user = userInfo[_user];
    uint256 accRewardPerShare = pool.accRewardPerShare;
    uint256 lpSupply = pool.amount;
    if (curBlock > pool.lastRewardBlock && lpSupply != 0) {
        uint256 multiplier = curBlock.sub( pool.lastRewardBlock, "cr curBlock sub overflow" );
        uint256 curReward = multiplier.mul(rewardPerBlock);
        accRewardPerShare = accRewardPerShare.add(curReward.mul(1e20).div(lpSupply));
    }
    uint newReward = user.amount.mul(accRewardPerShare).div(1e20).sub(user.rewardDebt, "cr newReward sub overflow");
    return newReward.add( user.reward );
}

function rewardOf( address _user ) public view returns( uint256 ){
    return userInfo[_user].reward;
}
```

```
function amount( ) public view returns( uint256 ){
    return mPoolInfo.amount;
}

function amountOf( address _user ) public view returns( uint256 ){
    return userInfo[_user].amount;
}

function getUserInfo(address _user) public view returns(uint256,uint256,uint256) {
    return (userInfo[_user].reward, userInfo[_user].amount, userInfo[_user].rewardDebt);
}

function getPoolInfo() public view returns(uint256,uint256,uint256) {
    return (mPoolInfo.amount, mPoolInfo.lastRewardBlock, mPoolInfo.accRewardPerShare);
}

function _update( uint256 curBlock ) internal {
    PoolInfo storage pool = mPoolInfo;
    if (curBlock <= pool.lastRewardBlock) {
        return;
    }
    uint256 lpSupply = pool.amount;
    if (lpSupply == 0) {
        pool.lastRewardBlock = curBlock;
        return;
    }
    uint256 multiplier = curBlock.sub( pool.lastRewardBlock, "_update curBlock sub overflow" );
    uint256 curReward = multiplier.mul(rewardPerBlock);

    remainReward = remainReward.add( curReward );
    accReward = accReward.add( curReward );

    pool.accRewardPerShare = pool.accRewardPerShare.add(curReward.mul(1e20).div(lpSupply));
    pool.lastRewardBlock = curBlock;
}

function _deposit( uint256 curBlock, address _addr, uint256 _amount) internal {
    PoolInfo storage pool = mPoolInfo;
    UserInfo storage user = userInfo[ _addr];
    _update( curBlock );
```

```
if (user.amount > 0) {
    uint256 pending = user.amount.mul(pool.accRewardPerShare).div(1e20).sub(user.rewardDebt, "_deposit pending
sub overflow");
    if(pending > 0) {
        reward( user, pending );
    }
}
if(_amount > 0) {
    user.amount = user.amount.add(_amount);
    pool.amount = pool.amount.add(_amount);
}
user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e20);
}

function _withdraw( uint256 curBlock, address _addr, uint256 _amount) internal {
    PoolInfo storage pool = mPoolInfo;
    UserInfo storage user = userInfo[_addr];
    require(user.amount >= _amount, "_withdraw: not good");
    _update( curBlock );
    uint256 pending = user.amount.mul(pool.accRewardPerShare).div(1e20).sub(user.rewardDebt, "_withdraw pending
sub overflow");
    if(pending > 0) {
        reward( user, pending );
    }
    if(_amount > 0) {
        user.amount = user.amount.sub(_amount, "_withdraw user.amount sub overflow");
        pool.amount = pool.amount.sub(_amount, "_withdraw pool.amount sub overflow");
    }
    user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e20);
}

function reward( UserInfo storage user, uint256 _amount) internal {
    if (_amount > remainReward) {
        _amount = remainReward;
    }
    remainReward = remainReward.sub( _amount, "reward remainReward sub overflow");
    user.reward = user.reward.add( _amount );
}

function _claim( address _addr ) internal {
    UserInfo storage user = userInfo[_addr];
```

```
        if( user.reward > 0 )
        {
            user.reward = 0;
        }
    }
}

contract LnRewardCalculatorTest is LnRewardCalculator{
    constructor( uint256 _rewardPerBlock, uint256 _startBlock ) public
        LnRewardCalculator( _rewardPerBlock, _startBlock ) {
    }

    function deposit( uint256 curBlock, address _addr, uint256 _amount) public {
        _deposit( curBlock, _addr, _amount );
    }

    function withdraw( uint256 curBlock, address _addr, uint256 _amount) public {
        _withdraw( curBlock, _addr, _amount );
    }

    function calcReward( uint256 curBlock, address _user) public view returns (uint256) {
        return _calcReward( curBlock, _user);
    }
}

contract LnSimpleStaking is LnAdmin, Pausable, ILinearStaking, LnRewardCalculator {
    using SafeMath for uint256;
    using SafeDecimalMath for uint256;

    IERC20 public linaToken; // lina token proxy address
    LnLinearStakingStorage public stakingStorage;
    uint256 public mEndBlock;
    address public mOldStaking;
    uint256 public mOldAmount;
    uint256 public mWithdrawRewardFromOldStaking;

    uint256 public claimRewardLockTime = 1620806400; // 2021-5-12

    address public mTargetAddress;
```

```
uint256 public mTransLockTime;

mapping (address => uint ) public mOldReward;

constructor(
    address _admin,
    address _linaToken,
    address _storage, uint256 _rewardPerBlock, uint256 _startBlock, uint256 _endBlock )
    public LnAdmin(_admin) LnRewardCalculator(_rewardPerBlock, _startBlock ){
    linaToken = IERC20(_linaToken);
    stakingStorage = LnLinearStakingStorage(_storage);
    mEndBlock = _endBlock;
}

function setLinaToken(address _linaToken) external onlyAdmin {
    linaToken = IERC20(_linaToken);
}
```

//SlowMist// 在出现重大交易异常时可以暂停所有操作，值得称赞的做法

```
function setPaused(bool _paused) external onlyAdmin {
    if (_paused) {
        _pause();
    } else {
        _unpause();
    }
}

////////////////////////////////////

event Staking(address indexed who, uint256 value, uint staketime);
event CancelStaking(address indexed who, uint256 value);
event Claim(address indexed who, uint256 rewardval, uint256 totalStaking);

uint256 public accountStakingListLimit = 50;
uint256 public minStakingAmount = 1e18; // 1 token
uint256 public constant PRECISION_UINT = 1e23;

function setLinaTokenAddress(address _token) external onlyAdmin {
    linaToken = IERC20(_token);
}

function setStakingListLimit(uint256 _limit) external onlyAdmin {
```

```
accountStakingListLimit = _limit;
}

function setMinStakingAmount(uint256 _minStakingAmount) external onlyAdmin {
    minStakingAmount = _minStakingAmount;
}

function stakingBalanceOf(address account) external override view returns(uint256) {
    uint256 stakingBalance = super.amountOf(account).add( stakingStorage.stakingBalanceOf(account) );
    return stakingBalance;
}

function getStakesdataLength(address account) external view returns(uint256) {
    return stakingStorage.getStakesdataLength(account);
}
//-----

function migrationsOldStaking( address contractAddr, uint amount, uint blockNb ) public onlyAdmin {
    super._deposit( blockNb, contractAddr, amount );
    mOldStaking = contractAddr;
    mOldAmount = amount;
}

function staking(uint256 amount) public whenNotPaused override returns (bool) {
    stakingStorage.requireInStakingPeriod();

    require(amount >= minStakingAmount, "Staking amount too small.");
    //require(stakingStorage.getStakesdataLength(msg.sender) < accountStakingListLimit, "Staking list out of limit.");

    linaToken.transferFrom(msg.sender, address(this), amount);

    uint256 blockNb = block.number;
    if (blockNb > mEndBlock) {
        blockNb = mEndBlock;
    }
    super._deposit( blockNb, msg.sender, amount );

    emit Staking(msg.sender, amount, block.timestamp);

    return true;
}
```



```
}

function _withdrawFromOldStaking( address _addr, uint amount ) internal {
    uint256 blockNb = block.number;
    if (blockNb > mEndBlock) {
        blockNb = mEndBlock;
    }

    uint oldStakingAmount = super.amountOf( mOldStaking );
    super._withdraw( blockNb, mOldStaking, amount );
    // sub already withdraw reward, then cal portion
    uint reward = super.rewardOf( mOldStaking).sub( mWithdrawRewardFromOldStaking, "_withdrawFromOldStaking
reward sub overflow" )
        .mul( amount ).mul(1e20).div( oldStakingAmount ).div(1e20);
    mWithdrawRewardFromOldStaking = mWithdrawRewardFromOldStaking.add( reward );
    mOldReward[ _addr ] = mOldReward[_addr].add( reward );
}

function _cancelStaking(address user, uint256 amount) internal {
    uint256 blockNb = block.number;
    if (blockNb > mEndBlock) {
        blockNb = mEndBlock;
    }

    uint256 returnAmount = amount;
    uint256 newAmount = super.amountOf(user);
    if (newAmount >= amount) {
        super._withdraw( blockNb, user, amount );
        amount = 0;
    } else {
        if (newAmount > 0) {
            super._withdraw( blockNb, user, newAmount );
            amount = amount.sub(newAmount, "_cancelStaking amount sub overflow");
        }

        for (uint256 i = stakingStorage.getStakesdataLength(user); i >= 1 ; i--) {
            (uint256 stakingAmount, uint256 staketime) = stakingStorage.getStakesDataByIndex(user, i-1);
            if (amount >= stakingAmount) {
                amount = amount.sub(stakingAmount, "_cancelStaking amount sub overflow");

                stakingStorage.PopStakesData(user);
            }
        }
    }
}
```

```
stakingStorage.SubWeeksTotal(staketime, stakingAmount);
_widthdrawFromOldStaking( user, stakingAmount );

} else {
    stakingStorage.StakingDataSub(user, i-1, amount);
    stakingStorage.SubWeeksTotal(staketime, amount);
    _widthdrawFromOldStaking( user, amount );

    amount = 0;
}
if (amount == 0) break;
}
}

// cancel as many as possible, not fail, that waste gas
//require(amount == 0, "Cancel amount too big then staked.");

linaToken.transfer(msg.sender, returnAmount.sub(amount));
}

function cancelStaking(uint256 amount) public whenNotPaused override returns (bool) {
    //stakingStorage.requireInStakingPeriod();

    require(amount > 0, "Invalid amount.");

    _cancelStaking(msg.sender, amount);

    emit CancelStaking(msg.sender, amount);

    return true;
}

function getTotalReward( uint blockNb, address _user ) public view returns ( uint256 total ){
    if( blockNb > mEndBlock ){
        blockNb = mEndBlock;
    }

    // 这里奖励分成了三部分
    // 1,已经从旧奖池中 cancel了的
    // 2,还在旧奖池中的
    // 3, 在新奖池中的
```

```
total = mOldReward[ _user ];
uint iMyOldStaking = 0;
for (uint256 i=0; i < stakingStorage.getStakesdataLength( _user ); i++) {
    (uint256 stakingAmount, ) = stakingStorage.getStakesDataByIndex( _user, i);
    iMyOldStaking = iMyOldStaking.add( stakingAmount );
}
if( iMyOldStaking > 0 ){
    uint oldStakingAmount = super.amountOf( mOldStaking );
    uint iReward2 = super._calcReward( blockNb, mOldStaking).sub( mWithdrawRewardFromOldStaking,
"getTotalReward iReward2 sub overflow" )
        .mul( iMyOldStaking ).div( oldStakingAmount );
    total = total.add( iReward2 );
}

uint256 reward3 = super._calcReward( blockNb, _user );
total = total.add( reward3 );
}

// claim reward
// Note: 需要提前把奖励 token 转进来
function claim() public whenNotPaused override returns (bool) {
    //stakingStorage.requireStakingEnd();
    require(block.timestamp > claimRewardLockTime, "Not time to claim reward");

    uint iMyOldStaking = stakingStorage.stakingBalanceOf( msg.sender );
    uint iAmount = super.amountOf( msg.sender );
    _cancelStaking( msg.sender, iMyOldStaking.add( iAmount ));

    uint iReward = getTotalReward( mEndBlock, msg.sender );

    _claim( msg.sender );
    mOldReward[ msg.sender ] = 0;
    linaToken.transfer(msg.sender, iReward );

    emit Claim(msg.sender, iReward, iMyOldStaking.add( iAmount ));
    return true;
}

function setRewardLockTime(uint256 newtime) public onlyAdmin {
    claimRewardLockTime = newtime;
}
```

```
}

function calcReward( uint256 curBlock, address _user) public view returns (uint256) {
    return _calcReward( curBlock, _user);
}

function setTransLock(address target, uint256 locktime) public onlyAdmin {
    mTargetAddress = target;
    mTransLockTime = locktime;
}

//SlowMist// 当前时间大于 locktime 时, Admin 可以转走 Staking 合约中的 Token

function transTokens(uint256 amount) public onlyAdmin {
    require(mTransLockTime > 0, "mTransLockTime not set");
    require(now > mTransLockTime, "Pls wait to unlock time");
    linaToken.transfer(mTargetAddress, amount);
}

}

contract HelperPushStakingData is LnAdmin {

    constructor(address _admin) public LnAdmin(_admin) {

    }

    function pushStakingData(address _storage, address[] calldata account, uint256[] calldata amount, uint256[] calldata
    staketime) external {
        require(account.length > 0, "array length zero");
        require(account.length == amount.length, "array length not eq");
        require(account.length == staketime.length, "array length not eq");

        LnLinearStakingStorage stakingStorage = LnLinearStakingStorage(_storage);
        for (uint256 i=0; i<account.length; i++) {
            stakingStorage.PushStakingData(account[i], amount[i], staketime[i]);
            stakingStorage.AddWeeksTotal(staketime[i], amount[i]);
        }
    }

    //unstaking.
}
```

```
contract MultiSigForTransferFunds {
    mapping (address => uint ) public mAdmins;
    uint public mConfirmNumb;
    uint public mProposalNumb;
    uint public mAmount;
    LnSimpleStaking public mStaking;
    address[] public mAdminArr;
    uint public mTransLockTime;

    constructor(address[] memory _addr, uint iConfirmNumb, LnSimpleStaking _staking ) public {
        for( uint i = 0; i < _addr.length; ++i ){
            mAdmins[ _addr[i] ] = 1;
        }
        mConfirmNumb = iConfirmNumb;
        mProposalNumb = 0;
        mStaking = _staking;
        mAdminArr = _addr;
    }

    function becomeAdmin(address target) external {
        LnAdmin(target).becomeAdmin();
    }

    function setTransLock(address target, uint256 locktime, uint amount ) public {
        require( mAdmins[ msg.sender] == 1, "not in admin list or set state" );
        _reset();
        mStaking.setTransLock( target, locktime );
        mAmount = amount;
        mProposalNumb = 1;
        mAdmins[ msg.sender] = 2; //

        mTransLockTime = locktime;
    }

    // call this when the locktime expired
    function confirmTransfer() public {
        require( mAdmins[ msg.sender] == 1, "not in admin list or set state" );
        mProposalNumb = mProposalNumb + 1;
        mAdmins[ msg.sender ] = 2;
    }
}
```

```
function doTransfer() public {
    require(mTransLockTime > 0, "mTransLockTime not set");
    require(now > mTransLockTime, "Pls wait to unlock time");
    require(mProposalNumb >= mConfirmNumb, "need more confirm");

    _reset();
    mStaking.transTokens( mAmount );
}
```

```
function _reset() internal {
    mProposalNumb = 0;
    mTransLockTime = 0;
    // reset
    for( uint i = 0; i < mAdminArr.length; ++i ){
        mAdmins[ mAdminArr[i] ] = 1;
    }
}
}
```

LnTokenStorage.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "./LnAdmin.sol";
import "./LnOperatorModifier.sol";

contract LnTokenStorage is LnAdmin, LnOperatorModifier {

    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    constructor(address _admin, address _operator) public LnAdmin(_admin) LnOperatorModifier(_operator) {}

    function setAllowance(address tokenOwner, address spender, uint value) external onlyOperator {
        allowance[tokenOwner][spender] = value;
    }
}
```

//SlowMist// 项目方可以通过修改 onlyOperator 权限然后调用 setBalanceOf 修改任意账户余额而 总量保持不变

```
function setBalanceOf(address account, uint value) external onlyOperator {  
    balanceOf[account] = value;  
}  
}
```

LnTokenStorageLock.sol

```
// SPDX-License-Identifier: MIT  
pragma solidity ^0.6.12;  
  
import "./LnAdmin.sol";  
import "./LnOperatorModifier.sol";  
import "./LnTokenStorage.sol";  
  
contract LnTokenStorageLock is LnAdmin {  
    address public operator;  
    address public mOpNew;  
    uint public mLock;  
    LnTokenStorage public mStorage;  
  
    constructor(address _admin, address _operator, LnTokenStorage store) public LnAdmin(_admin) {  
        mStorage = store;  
        operator = _operator;  
    }  
  
    modifier onlyOperator() {  
        require(msg.sender == operator, "Only operator can perform this action 2");  
        _;  
    }  
  
    function setOperator( address op, uint time ) public onlyAdmin {  
        mLock = time;  
        mOpNew = op;  
    }  
  
    function _updateOperator() internal {  
        if( mOpNew != address(0) ) {  
            if( now > mLock ){
```

```
        operator = mOpNew;
        mOpNew = address(0);
    }
}

function setAllowance(address tokenOwner, address spender, uint value) public onlyOperator {
    _updateOperator();
    mStorage.setAllowance( tokenOwner, spender, value );
}
```

//SlowMist// 项目方可以通过修改 onlyOperator 权限然后调用 setBalanceOf 修改任意账户余额而 总量保持不变

```
function setBalanceOf(address account, uint value) public onlyOperator {
    _updateOperator();
    mStorage.setBalanceOf( account, value );
}

function allowance(address owner, address spender) public view returns(uint) {
    return mStorage.allowance(owner, spender);
}

function balanceOf(address account) public view returns(uint) {
    return mStorage.balanceOf(account);
}
}
```

SafeDecimalMath.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";

library SafeDecimalMath {
    using SafeMath for uint;

    uint8 public constant decimals = 18;
    uint8 public constant highPrecisionDecimals = 27;

    uint public constant UNIT = 10**uint(decimals);
}
```



```
uint public constant PRECISE_UNIT = 10**uint(highPrecisionDecimals);
uint private constant UNIT_TO_HIGH_PRECISION_CONVERSION_FACTOR = 10**uint(highPrecisionDecimals - decimals);

function unit() external pure returns (uint) {
    return UNIT;
}

function preciseUnit() external pure returns (uint) {
    return PRECISE_UNIT;
}

function multiplyDecimal(uint x, uint y) internal pure returns (uint) {

    return x.mul(y) / UNIT;
}

function _multiplyDecimalRound(
    uint x,
    uint y,
    uint precisionUnit
) private pure returns (uint) {

    uint quotientTimesTen = x.mul(y) / (precisionUnit / 10);

    if (quotientTimesTen % 10 >= 5) {
        quotientTimesTen += 10;
    }

    return quotientTimesTen / 10;
}

function multiplyDecimalRoundPrecise(uint x, uint y) internal pure returns (uint) {
    return _multiplyDecimalRound(x, y, PRECISE_UNIT);
}

function multiplyDecimalRound(uint x, uint y) internal pure returns (uint) {
    return _multiplyDecimalRound(x, y, UNIT);
}

function divideDecimal(uint x, uint y) internal pure returns (uint) {
    return x.mul(UNIT).div(y);
}
```

```
}  
  
function _divideDecimalRound(  
    uint x,  
    uint y,  
    uint precisionUnit  
) private pure returns (uint) {  
    uint resultTimesTen = x.mul(precisionUnit * 10).div(y);  
  
    if (resultTimesTen % 10 >= 5) {  
        resultTimesTen += 10;  
    }  
  
    return resultTimesTen / 10;  
}  
  
function divideDecimalRound(uint x, uint y) internal pure returns (uint) {  
    return _divideDecimalRound(x, y, UNIT);  
}  
  
function divideDecimalRoundPrecise(uint x, uint y) internal pure returns (uint) {  
    return _divideDecimalRound(x, y, PRECISE_UNIT);  
}  
  
function decimalToPreciseDecimal(uint i) internal pure returns (uint) {  
    return i.mul(UNIT_TO_HIGH_PRECISION_CONVERSION_FACTOR);  
}  
  
function preciseDecimalToDecimal(uint i) internal pure returns (uint) {  
    uint quotientTimesTen = i / (UNIT_TO_HIGH_PRECISION_CONVERSION_FACTOR / 10);  
  
    if (quotientTimesTen % 10 >= 5) {  
        quotientTimesTen += 10;  
    }  
    return quotientTimesTen / 10;  
}  
}
```



官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号

