第9章 智能合约抽奖

目录

CONTENTS

CONTENTS

章节

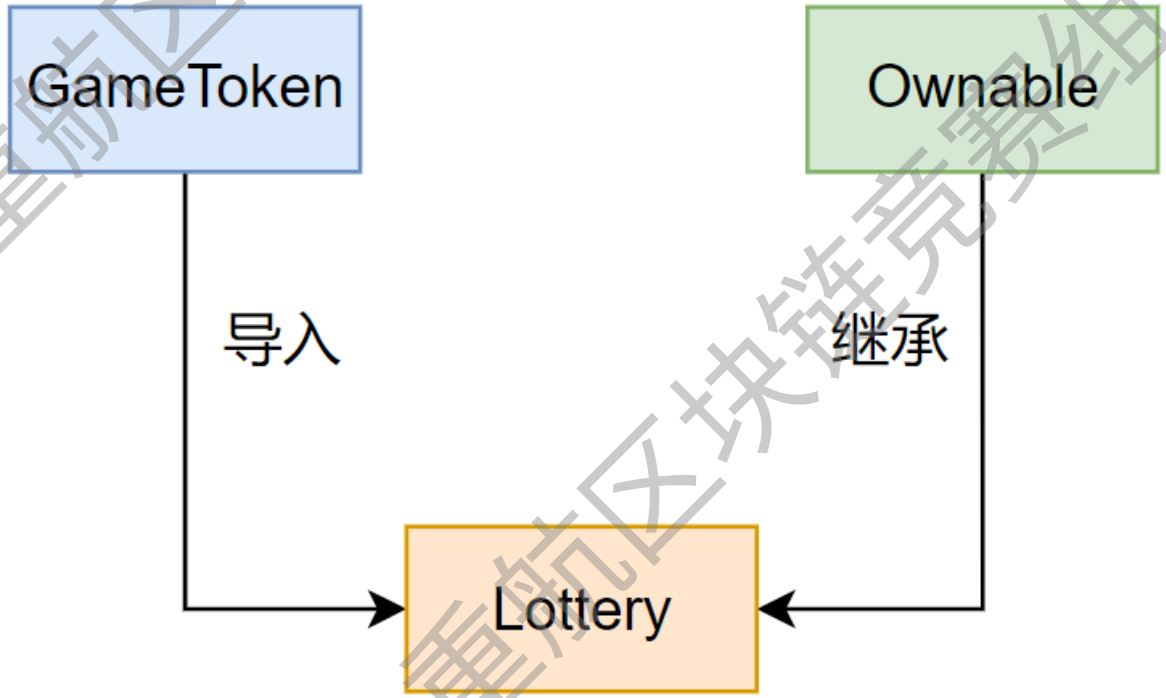
代币抽奖-合约设计

7.1 合约设计

腾讯云

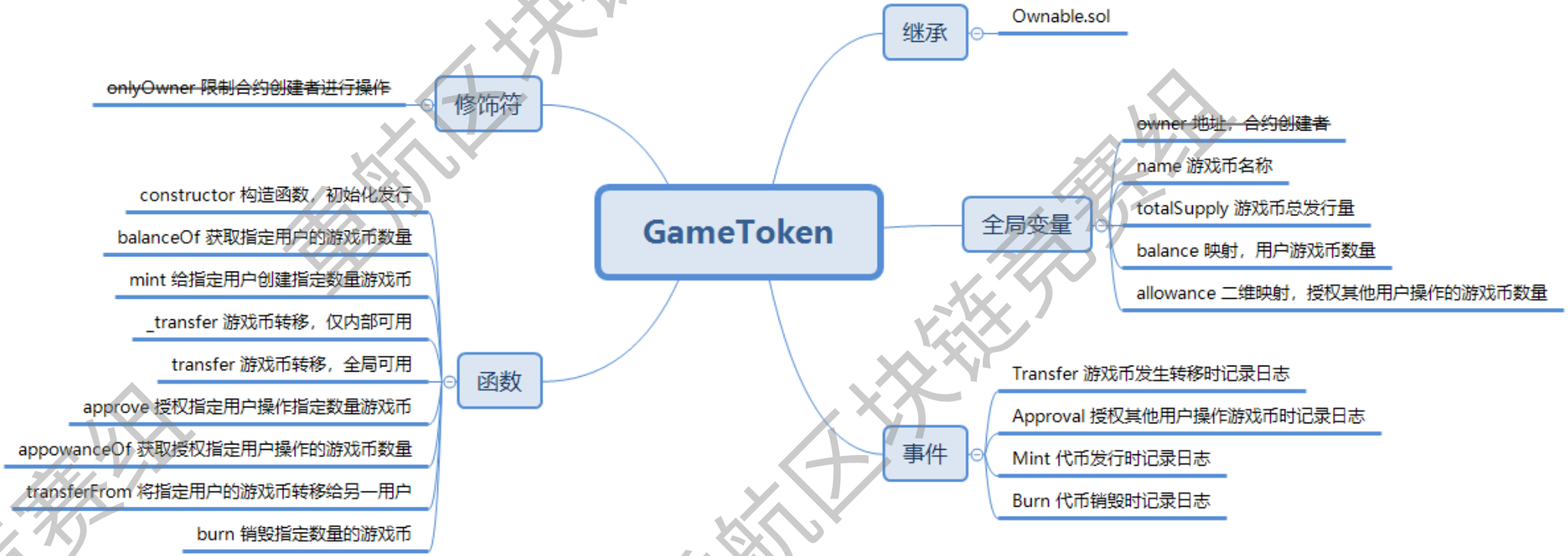
合约设计

- 本章节在“简单抽奖”合约和“游戏代币”合约的基础上，开发一个“游戏代币抽奖合约”，在抽奖合约中导入游戏币合约，使用游戏代币进行投注。并将`onlyOwner`修饰符独立成`Ownable`合约，继承该合约。



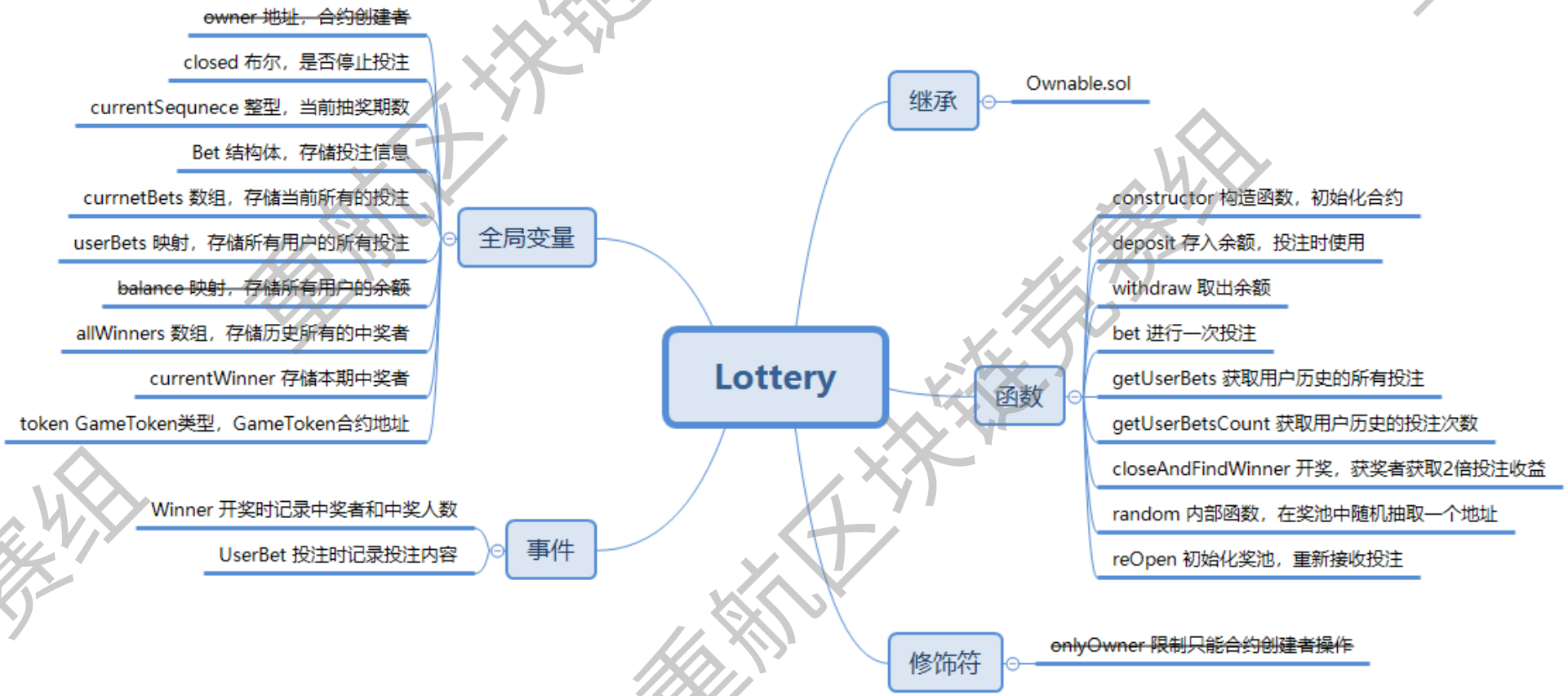
合约设计

- 代币合约中包括以下全局变量，事件和函数。在原合约的基础上移除了部分变量和修饰符，添加了继承部分



合约设计

- 代币抽奖合约中包括以下全局变量，事件和函数。在原合约的基础上移除了部分变量和修饰符，添加了继承部分



CONTENTS

章节

代币抽奖-合约编写

8.1 合约编写

腾讯云

合约编写-Ownable合约

- 编写Ownable.sol合约，该合约只具有限制合约创建者操作功能

```
// 声明solidity版本为0.6.10
pragma solidity 0.6.10;

// 定义Ownable合约
contract Ownable {
    // 合约创建者变量
    address public owner;
    // 创建合约时初始化owner变量
    constructor() public {
        owner = msg.sender;
    }
    // 修饰符，只允许合约创建者操作
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
}
```

合约编写-GameToken合约

- 初始化游戏代币合约，使用solidity 0.6.10版本，合约名为GameToken，使其继承Ownable.sol合约

```
// 声明solidity版本为0.6.10
pragma solidity 0.6.10;
import "./Ownable.sol";

// 定义GameToken合约，继承Ownbale合约
contract GameToken is Ownable {

}
```

合约编写-GameToken合约

- 定义全局变量，包括游戏币名称，总发行量，用户余额映射，授权金额二维映射

```
// 游戏币名称
string public name;

// 总发行量
uint public totalSupply;

// 用户余额
mapping (address => uint) balance;

// 授权金额
mapping (address => mapping (address => uint256)) allowance;
```

合约编写-GameToken合约

- 定义事件，在代币转移，代币授权，代币发行，代币销毁时记录日志

```
// 事件日志，代币转移
event Transfer(address from, address to, uint value);

// 事件日志，授权
event Approval(address _owner, address _spender, uint256 _value);

// 事件日志，代币发行
event Mint(address from, uint value);

// 事件日志，代币销毁
event Burn(address from, uint value);
```

合约编写-GameToken合约

- 定义构造函数，功能为初始化初始发行量、游戏币名称、合约创建者

```
// 构造函数，初始化初始发行量、游戏币名称、合约创建者
constructor(uint initialSupply, string memory tokenName) public {
    // 初始化发行量
    totalSupply = initialSupply;
    // 将初始发行的余额转移给合约创建者
    balance[msg.sender] = totalSupply;
    // 初始化合约名
    name = tokenName;
    // 初始化合约创建者
    owner = msg.sender;
}
```

合约编写-GameToken合约

- 定义mint函数，功能为给指定用户创建指定数量的游戏代币，增加指定地址余额，总发行量，并记录事件日志

```
// 给指定用户创建指定金额的游戏币
function mint(address who, uint _value) public onlyOwner returns(bool success) {
    // 给指定用户增加余额
    balance[who] += _value;
    // 增加总发行量
    totalSupply += _value;
    // 记录事件日志
    emit Mint(who, _value);
    return true;
}
```

合约编写-GameToken合约

- 定义_transfer函数，仅在内部可用，功能为游戏代币转移，减少转出地址的余额，增加转入地址的余额，并记录事件日志

```
// 游戏币转移，仅在内部使用
function _transfer(address _from, address _to, uint _value) private returns(bool
success) {
    // 余额充足才可操作
    require(balance[_from] >= _value, "余额不足");
    // 减少转出地址的余额
    balance[_from] -= _value;
    // 增加转入地址的余额
    balance[_to] += _value;
    // 记录事件日志
    emit Transfer(_from, _to, _value);
    return true;
}
```

合约编写-GameToken合约

- 定义transfer函数，全局可用，在函数中调用_transfer内部函数

```
// 将调用者的游戏币转移给_to用户
function transfer(address _to, uint _value) public returns (bool success) {
    return _transfer(msg.sender, _to, _value);
}
```

- 定义approve函数，功能为授权一个用户操作指定数量游戏币

```
// 授权_spender用户操作调用者用户_value数量余额
function approve(address _spender, uint256 _value) public returns (bool success) {
    // 授权用户操作
    allowance[msg.sender][_spender] = _value;
    // 记录事件日志
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

合约编写-GameToken合约

- 定义balanceOf函数，功能为获取指定地址的游戏币数量

```
// 获取指定用户的游戏币余额
function balanceOf(address who) public view returns(uint) {
    return balance[who];
}
```

- 定义allowanceOf函数，功能为获取授权指定地址操作的游戏币数量

```
// 获取owner用户允许spender用户操作的金额
function allowanceOf(address owner, address spender) public view
returns(uint) {
    return allowance[owner][spender];
}
```

合约编写-GameToken合约

- 定义transferFrom函数，功能将用户授权的游戏币数量转移给指定地址

```
// 调用者将_from用户的_value数量余额转移给_to用户
function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success) {
    // 授权余额必须充足
    require(_value <= allowance[_from][msg.sender], "授权金额不足");
    // 减少授权金额数量
    allowance[_from][msg.sender] -= _value;
    // 执行_transfer内部函数
    _transfer(_from, _to, _value);
    return true;
}
```

合约编写-GameToken合约

- 定义burn函数，功能为销毁指定用户指定数量的游戏币

```
// 销毁指定用户_value数量余额
function burn(address who, uint _value) public onlyOwner returns
(bool success) {
    // 限制销毁数量不能大于余额
    require(balance[who] >= _value);
    // 减少指定地址游戏币数量
    balance[who] -= _value;
    // 减少总发行量
    totalSupply -= _value;
    // 记录事件日志
    emit Burn(who, _value);
    return true;
}
```

合约编写-Lottery合约

- 初始化代币抽奖合约，使用solidity 0.6.10版本，合约名为Lottery，使其继承Ownable.sol合约

```
// 声明solidity版本为0.6.10
pragma solidity 0.6.10;
import "./Ownable.sol";

// 定义Lottery合约，继承Ownbale合约
contract Lottery is Ownable {

}
```

合约编写-Lottery合约

- 定义全局变量，包括是否停止投注，GameToken合约地址，当前期数，单次抽奖价格，抽奖结构体

```
// 是否停止投注
bool public closed;
// GameToken合约地址
GameToken public token;
// 当前抽奖期数
uint public currentSequnece = 1000;
// 单次抽奖价格
uint public price = 10;

// 抽奖结构体
struct Bet {
    address betUser; // 投注者
    bytes3  betStr; // 投注内容
    uint sequence; // 抽奖期数
}
```

合约编写-Lottery合约

- 定义全局变量，包括当前所有投注，用户的投注，历史所有中奖者，当前期中奖者

```
// 当前所有的投注
Bet[] currentBets;

// 用户的投注
mapping(address => Bet[]) userBets;

// 历史所有中奖者
address[] public allWinners;

// 当前期中奖者
address public currentWinner;
```

合约编写-Lottery合约

- 定义构造函数，在合约执行时初始化owner变量，创建一个GameToken合约，初始化1000个游戏币，游戏币名为“ZHCoin”

```
constructor() public {
    owner = msg.sender;
    // 创建GameToken合约，初始化1000个游戏币，游戏币名为ZHCoin
    token = new GameToken(1000, "ZHCoin");
}
```

合约编写-Lottery合约

- 定义deposit函数和withdraw函数，处理用户存入和取出余额逻辑

```
// 存入余额
function deposit(uint value) public {
    // 执行GameToken合约mint函数
    require(token.mint(msg.sender, value));
}
```

```
// 取出余额
function withdraw() public returns (uint) {
    // 判断余额是否足够
    uint value = token.balanceOf(msg.sender);
    // 执行GameToken合约中的burn函数
    token.burn(msg.sender, value);
    return value;
}
```

合约编写-Lottery合约

- 定义bet函数，用户进行投注，每次投注时会扣除指定数量的游戏币

```
function bet(bytes3 betStr) public {
    // 限制投注仍在进行中
    require(closed == false, "已停止投注");
    // 限制余额充足
    require(token.balanceOf(msg.sender) >= price, "余额不足");
    // 将投注者余额转移到Lottery合约中
    require(token.transferFrom(msg.sender, address(this), price));
    // 创建Bet结构体
    Bet memory item  = Bet({
        betUser: msg.sender,
        betStr: betStr,
        sequence: currentSequnece
    });
    // 将投注信息插入到currentBets数组中
    currentBets.push(item);
    userBets[msg.sender].push(item);
    // 记录事件日志
    emit UserBet(msg.sender, betStr, currentSequnece);
}
```

合约编写-Lottery合约

- 定义getUserBets函数，功能为获取用户历史的所有投注

```
// 获取用户历史所有投注
function getUserBets() public view returns (bytes3[] memory, uint[] memory){
    // 获取用户历史所有投注bets数组
    Bet[] memory bets = userBets[msg.sender];
    // 获取数组长度
    uint length = bets.length;
    // 初始化strs和seqs数组
    bytes3[] memory strs = new bytes3[](length);
    uint[] memory seqs = new uint[](length);
    // 填充strs和seqs数组
    for(uint i = 0; i <length; i++){
        Bet memory item = bets[i];
        strs[i]=(item.betStr);
        seqs[i] = (item.sequence);
    }
    return (strs, seqs);
}
```

合约编写-Lottery合约

- 定义getUserBetsCount函数，功能为获取用户历史投注次数

```
// 获取用户历史投注次数
function getUserBetsCount() public view returns (uint){
    return userBets[msg.sender].length;
}
```

- 定义random函数，功能为在当期奖池中随机抽取一个中奖地址

```
// 在当期奖池中抽取一个地址
function random() private view returns (address){
    // 随机生成一个索引值
    uint randIdx = (block.number^block.timestamp) % currentBets.length;
    // 根据索引取出中奖地址
    Bet memory item = currentBets[randIdx] ;
    return item.betUser;
}
```

合约编写-Lottery合约

- 定义closeAndFindWinner函数，功能为开奖，在函数内部执行random函数，然后将中奖收益转移给获奖者

```
// 开奖，获奖者获取2倍投注的收益
function closeAndFindWinner() public onlyOwner {
    // 限制投注已停止
    require(closed == false);
    // 限制参与投注人数大于等于2
    require(currentBets.length >= 2);
    closed = true;
    // 随机抽取中奖用户
    currentWinner = random();
    // 调用GameToken的transfer函数将奖金转给中奖用户
    require(token.transfer(currentWinner, price*2));
    // 将中奖用户地址添加到所有中奖用户数组中
    allWinners.push(currentWinner);
    // 执行事件，记录日志
    emit Winner(currentWinner, allWinners.length);
}
```

合约编写-Lottery合约

- 定义reOpen函数，功能为初始化奖池，重新接受用户投注

```
// 初始化奖池，重新接受投注
function reOpen() public onlyOwner {
    // 限制处于投注截止状态
    require(closed == true);
    closed = false;
    // 删除本期投注中的所有数据
    for (uint i = 0; i < currentBets.length; i++){
        delete userBets[currentBets[i].betUser];
    }
    // 初始化当前所有投注数组
    delete currentBets;
    // 初始化当前中奖者变量
    delete currentWinner;
}
```

CONTENTS

章节

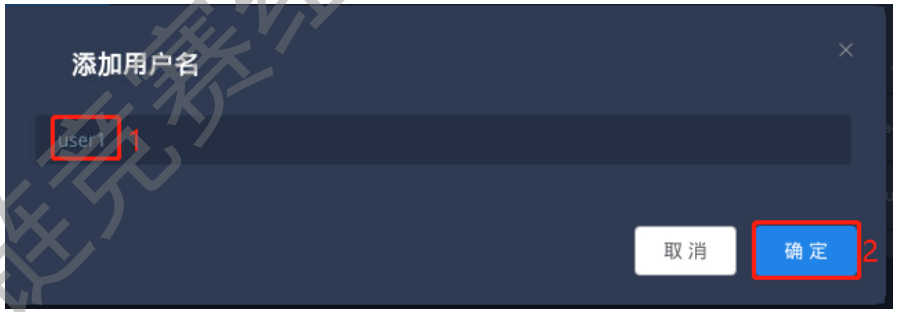
代币抽奖-合约测试

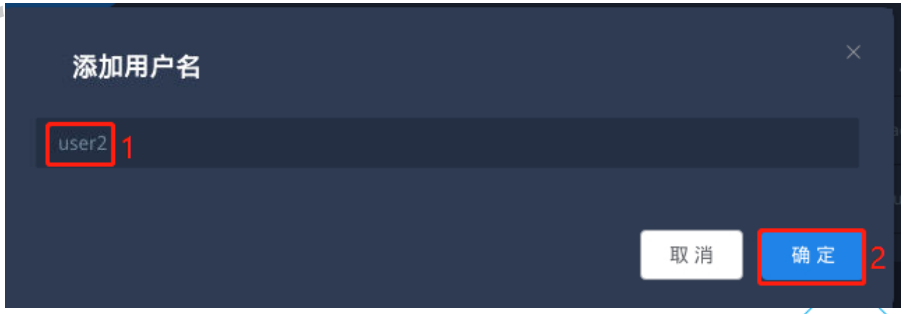
9.1 合约测试

腾讯云

合约测试

- 合约部署前，创建三个测试账户，分别为admin，user1，user2





合约测试

- user1、user2、user3三个用户对应的地址

合约测试-Lottery合约

- 合约编译完成后，使用管理员admin账户部署，部署完成后执行deposit函数，向user1地址存入50游戏代币，调用成功后Mint事件日志执行

合约测试-Lottery合约

- 执行deposit函数，向user2地址存入50游戏代币，调用成功后Mint事件日志执行

合约测试-Lottery合约

- 获取GameToken地址，在创建Lottery合约时也创建了GameToken合约，调用token变量可以查看该合约的地址

合约测试-GameToken合约

- 使用user1地址调用GameToke合约的approve函数，授权admin地址操作user1地址10游戏代币

合约测试-GameToken合约

- 使用user2地址调用GameToke合约的approve函数，授权admin地址操作user2地址10游戏代币

合约测试-Lottery合约

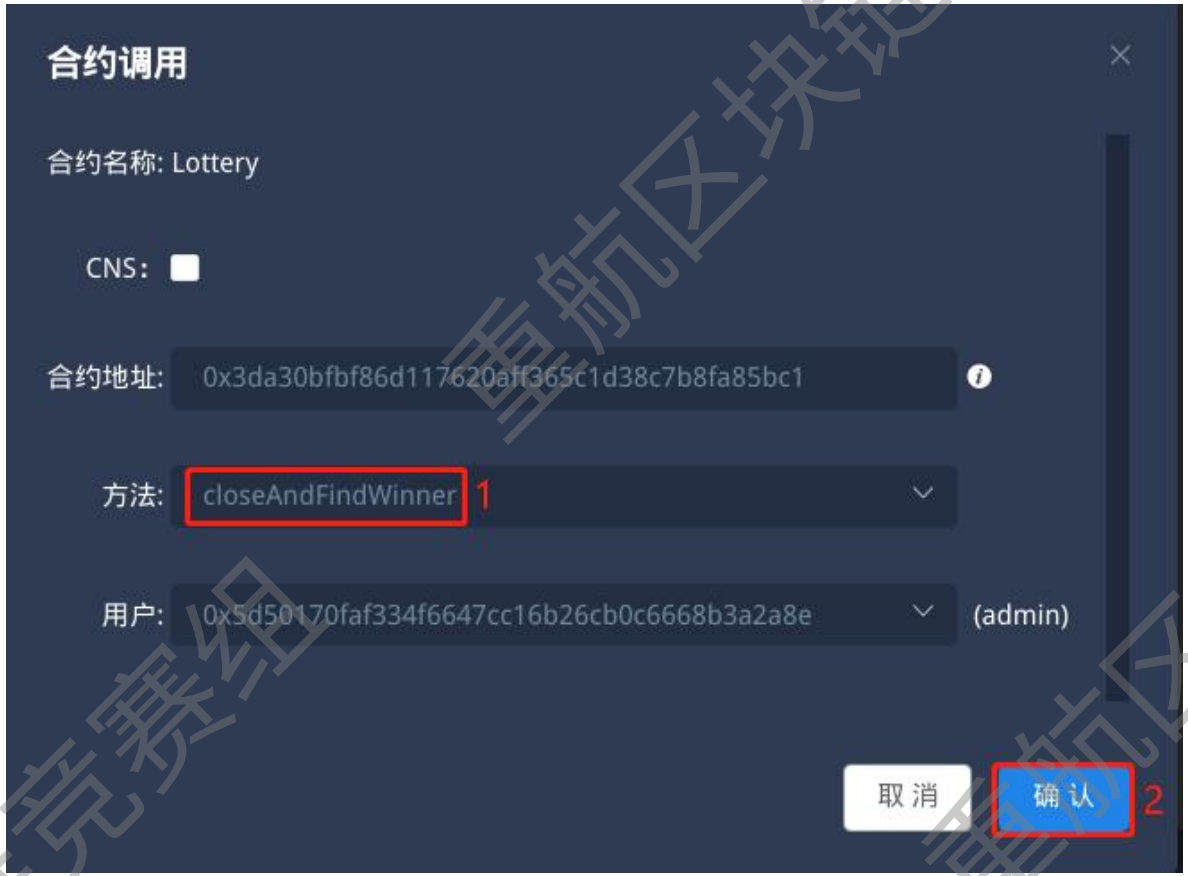
- 投注，user1地址调用Lottery合约的bet函数，投注内容为0x112233，调用成功后在交易回执中可以观察到UserBet事件日志

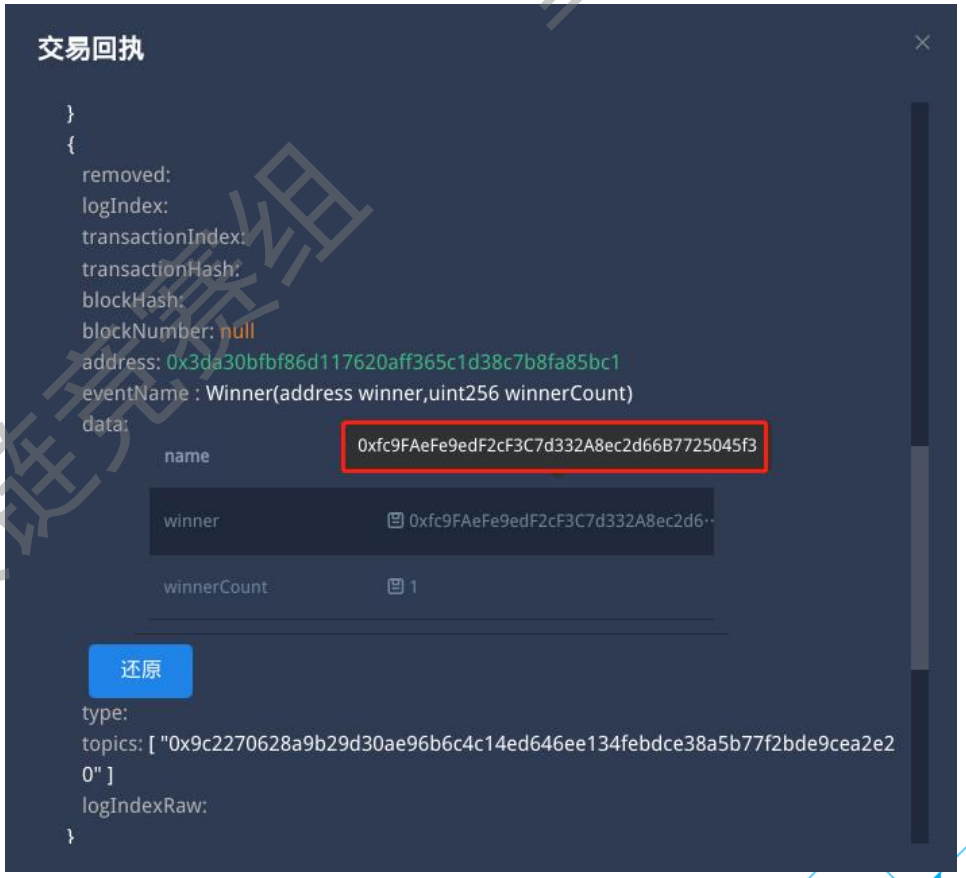
合约测试-Lottery合约

- 投注，user2地址调用Lottery合约的bet函数，投注内容为0x223344，调用成功后在交易回执中可以观察到UserBet事件日志

合约测试-Lottery合约

- 开奖，使用admin用户调用closeAndFindWinner函数，选出获奖者，执行成功后，可以在交易回执中观察到获奖者为user2地址





合约测试-Lottery合约

- user2用户取出获奖收益，执行withdraw函数，在交易回执中可以观察到共取出了60个游戏代币（50本金+10收益）

腾讯云

总结

- 本节课我们学习了
  - 代币抽奖合约的设计
  - 代币抽奖合约的编写
  - 代币抽奖合约的测试

谢谢