基于区块链的用户注册-拍卖系统

一、设计思路

使用智能合约进行拍卖活动，拍卖过程将变得简单、高效，不需要第三方的中间人进行裁决，也完全不会出现赖账问题。

在这次的作业中，我将一个注册系统与拍卖系统相结合，实现了一个简单的用户注册-拍卖的智能合约。

程序逻辑是由拍卖发起者设置活动描述和拍卖时间期限，在时间期限内，用户首先需要通过一个简单的注册系统，输入自己的昵称和年龄等基本信息与自己的地址账户进行临时绑定，随后发起竞价。如果没有进行注册将没有竞价资格；如果当前用户出价超过此前最高出价，出价金额将会打入到智能合约中进行保存，智能合约归坏出价次高者的出价；拍卖结束后，最高出价将会发送到拍卖发起者账户中。

二、设计流程与实现方法

本次合约设计的程序流程图如图 1 所示。

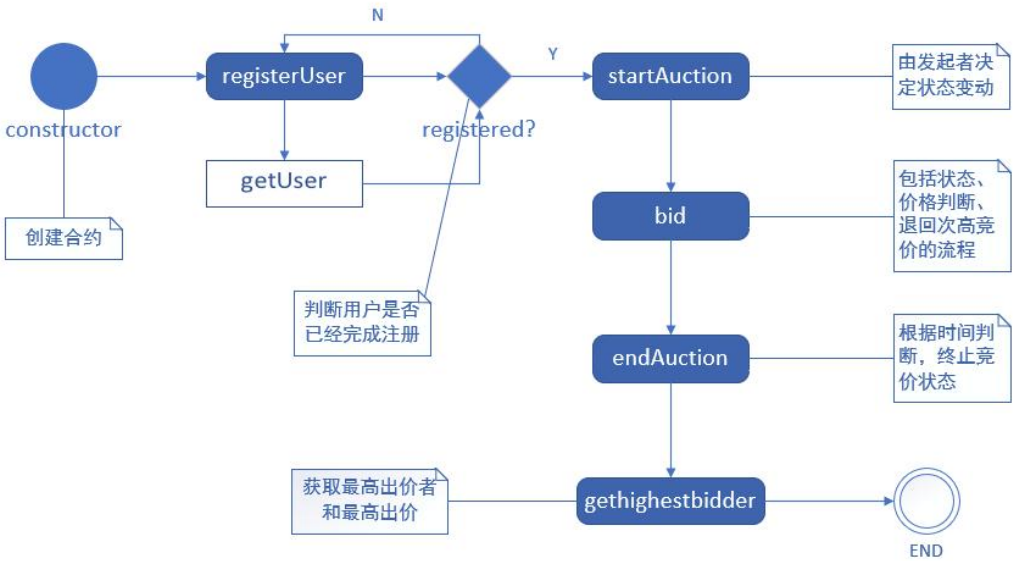

图 1 用户注册-拍卖智能合约设计流程图

部署智能合约时需要设置的变量有三个，分别为合约发起人地址、项目描述和竞拍时长。

在注册用户时，提前将用户地址使用格式转化为整型，作为 id 号映射到用户的基本信息。需要用户输入地址、昵称和年龄完成注册。注册完成后，使用一个数组存储用户 id 号。任意用户可以通过 id 号搜索其他用户的基本信息。

```
constructor (
    uint _biddingTime,
    string _projectDescription,
    address _beneficiary
)  public {
    projectDescription = _projectDescription;  //保存项目的描述
    beneficiary = _beneficiary;   //拍卖发起者账户地址
    auctionEndtime = block.timestamp + _biddingTime;    //竞拍时长
}
```

图 2 创建合约

在合约部署后，竞价状态默认为 false。竞价拍卖发起者将竞价状态设置为 true 后，直到拍卖结束都不可再修改。除此之外，用户参与竞价还需要满足时间在规定时间内，出价高于已有最高竞价，和已经完成竞价的条件。

```
    bool isinarray = false;
    require( block.timestamp <= auctionEndtime, "Auction already ended.");
    require( isAuction == true, "cannot bid now.");
    auctionStarted = true;
    // 如果出价不够高，返还你的钱
    require( msg.value > highestBid, "There already is a higher bid.");
    // 防止虚空用户竞价
    for (uint i = 0; i < addrList.length; i++) {
        if(addrList[i] == uint(msg.sender)){
            isinarray = true;
        }
    }
    require( isinarray == true,"Please sign in first next time!");
```

图 3 用户参与竞价需要满足的四个条件

一共满足这四个条件后，用户可以发起竞价。如出价为当前最高出价，该出价将发送至智能合约内；若出现更高出价，次高出价将返还至对应账户；到达预设时间后，最高出价被发送至拍卖发起人账户，拍卖结束。

```
96          // 之前的最高出价者收回出价
97          highestBidder.transfer(highestBid);
98          // 最高出价和最高出价者的转移
99          highestBidder = msg.sender;
100         highestBid = msg.value;
101         emit HighestBidIncreased(msg.sender, msg.value);
```

图 4 最高竞价和最高竞价者的转移

合约包括两个 bool 型的状态变量，分别用于判断是否是竞价状态，和竞价状态是否开始。后者用于防止开始竞价函数的重复调用。

在竞价过程中，新用户依旧可以注册并参与。用户可实时查询最高出价者和最高出价，可查询结束时间。

```
    // 1. 条件
    require(block.timestamp >= auctionEndtime, "Auction not yet ended.");
    // 2. 生效
    isAuction = false;
    emit AuctionEnded(highestBidder, highestBid);
    // 3. 交互
    beneficiary.transfer(highestBid);
```

图 5 拍卖结束的条件和支付转移

```
function gethighestbidder() external view returns (string name, uint age, uint bidnum){
    // 竞价结束后进行查询
    require(isAuction == false,"Auction not yet ended.");

    User storage user = idToUser[uint(highestBidder)];
    (name, age) = (user.name, user.age);
    bidnum = highestBid;
}
```

图 6 竞价结束后可查询最高出价者信息

至此，整个用户拍卖-竞价的系统流程结束。

三、系统仿真和运行结果

图 7 部署设置

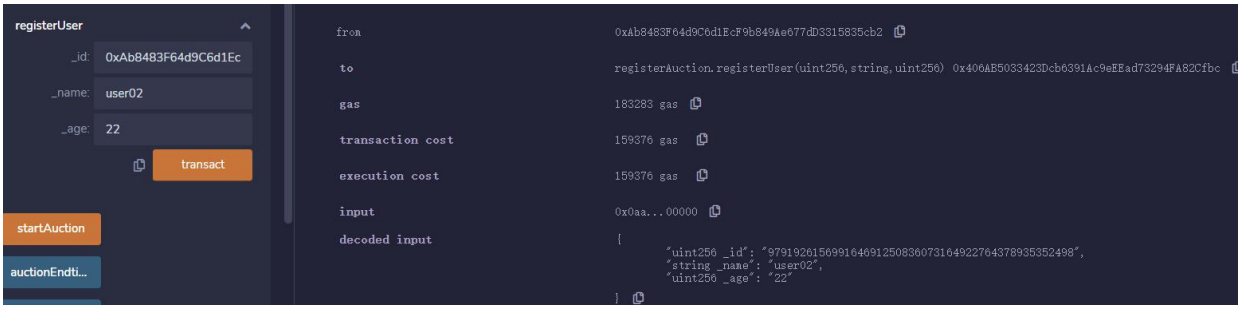

图 8 注册用户 user02

图 9 注册用户 03

图 10 注册用户 04

图 11 根据地址查询用户 04 的信息

图 12 用户 04 成功出价 1000，之前的出价被退回

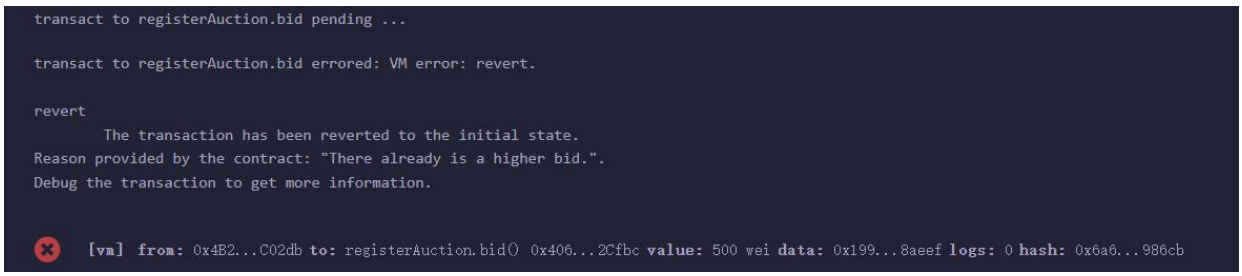

图 13 更少的出价 500wei 被回绝

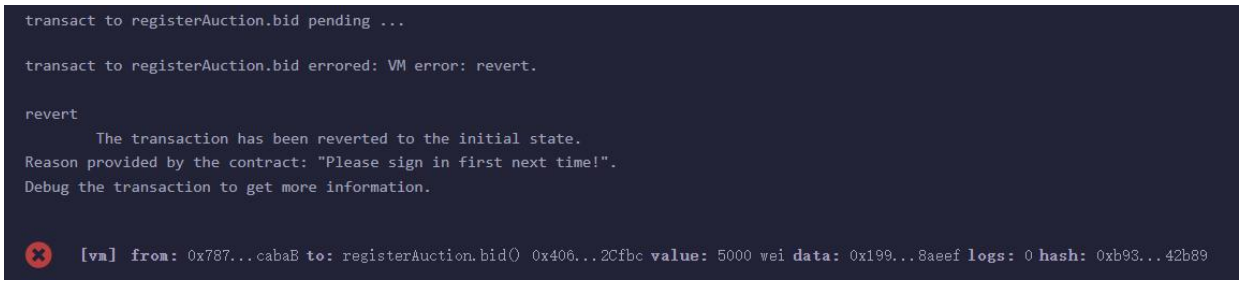

图 14 未注册的用户被回绝

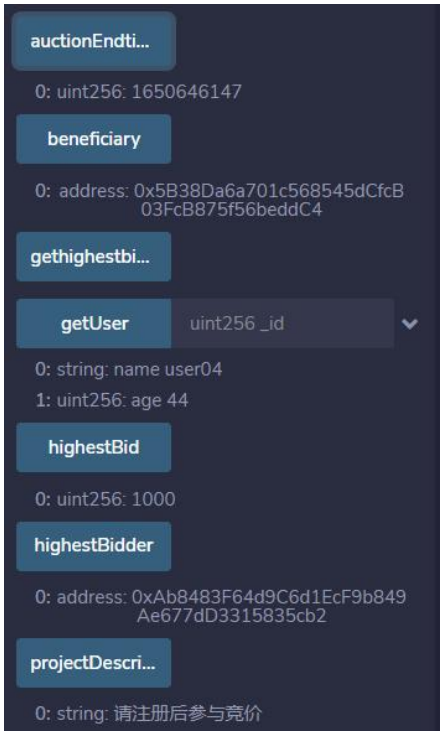

图 15 拍卖未结束时可查询的信息，包括结束时间，发起人，最高出价

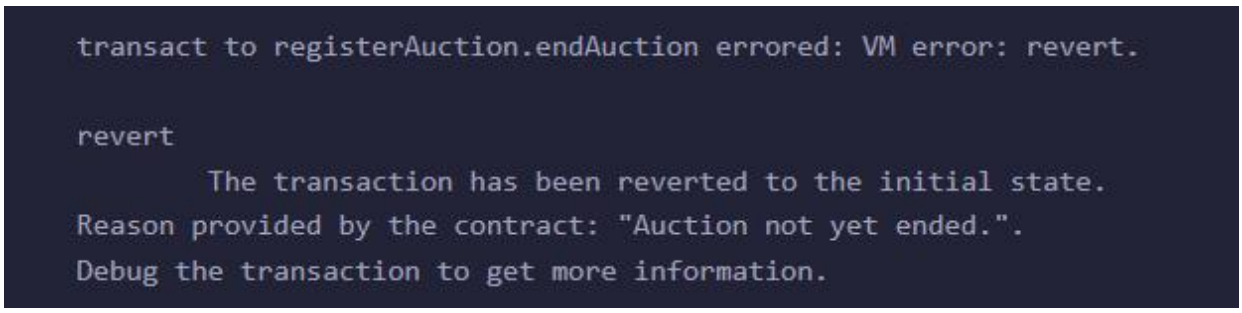

图 16 时间未到，拍卖便不可强行结束

图 17 拍卖结束，可查询最高竞价者，最高出价发送至发起人地址，一次拍卖结束

四、小结

本次作业使用 Solidity 语言实现了一个简单的智能合约的系统。面对一个之前未接触过的语言，通过研究范例代码、凭借以前的一些编程经验，最终实现了这些简单的功能。通过亲手实践智能合约设计、编程、部署的全过程，我对区块链和智能合约的原理和应用都有了进一步的体会和了解。