XJTLU

DataToken Developer Diary

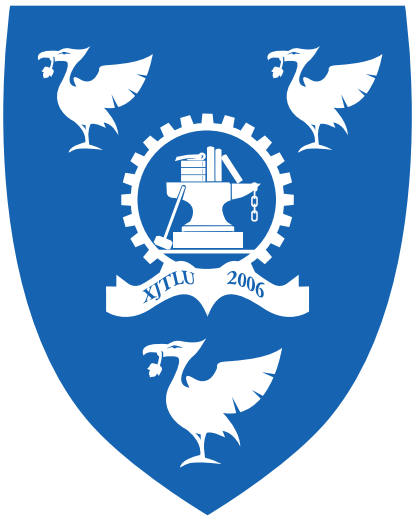
Solidity Smart Contract

P2P Cellular Data Sharing

Author:
Zhuoqun Liu

Supervisor:
Dr. Siyi Wang

November 14, 2017



Contents

Abstract

This project aims to implement a blockchain oriented cellular data sharing software on Ethereum platform. The first part of this Final Year Project focuses on software layer implementation, and the second half will consider some potential implementations in real life.

1 Introduction

DataToken is the name of the smart contract to be developed as my Final Year Project (FYP) in XJTLU. All DataToken projects will be named with postfix"alpha" since the propotype development is far from finished yet. Versions are depicted in *.* format e.g. 0.0; increment on the number before the dot means new functions or features are added, increment on the number after the dot means minor changes are added to source code.

2 DataToken Alpha 0.0

version alpha 0.0 initially designed functions createAccount, askforSharing

3 DataToken Alpha 0.1 20171014 Fri

The contract should be an ether pool (etherbase) thus it should allow user charge their account by putting ether in to the contranct. the first method to put ether in is the provide value when creating a user account. A second way is to call topUp function to send ether to this contract and get some token. By design, the token distributed by the contract should be at a constant exchange rate to ether. currently, the rate is set to be 1 token = 1 wei which is the smallest ether unit.

A withdraw function is created corresponding to topUp function. This function should allow user account to exchange their token back to ether in their ether address. But the

chellange is, when sending ether from the contract, the gas fee is to be paid by the sender, if the sender here is the message sender i.e. the user, the user will pay for the gas, however, if the sender is considered as the contract itself, there will be a problem. If the contract is charged gas for each transaction (the contract will loss at least 0.001 ether for each transaction according to current gas price), the contract etherbase will fail due to too many withdraw transactions. The contract itself is not making any profit but will have to pay some fee due to users' transaction, that's not fair. Then an experiment should be held to examine how the ethereum network perform such a transaction from the contract with msg.sender a user to be the caller of the function. 0.001 ether = 10^15 wei which is a large amount of loss in terms of wei. experiment design: create the contract with external address A; create user account with external address B, charge 0.01 ether for token; expecting: the contract has 0.01 ether and address B pay 0.01 ether plus gas fee; address B call withdraw function to withdraw 0.01 ether; expecting: the contract send 0.01 ether to B and B pay the gas, resulting B receive 0.01 minus gas fee. An easy way to examine the behavior of the function (actually the contract convention) is to set a getter for the contract. top-up the contract first, suppose the contract possesses _amountA wei check the contract balance; withdraw _amountB wei by calling function withdraw as external address (user) check the contract balance; if the balance == _amountA - _amountB the transaction fee was paid by the msg.sender i.e. the user account. else the fee was paid by the contract

If by convention the user will pay for calling a function that send ether from contract etherbase, the withdraw function need not to have a mechanism to make the ether in contract intact

The contract shouldn't be paying that fee for transfering back.

3.1 Diary 20171016 Mon

Mapping a host address to a guest address will make the link unique, thereby the sevice can only be recorded one on one which is no good for practical use. I'm now searching for a datastructurea like array which can be marked as related to one host address so that all guests of the same host address can be stored in it. In the documentation of solidity $\hat{0}.4.18$, mapping types expression

$$mapping(KeyType => ValueType)$$

allows KeyType to be almost any type except for mapping type; ValueType to any type including mapping type. This mapping feature can merge mappings from guest to payment status. And here is a solution for linking host to guest.The following code is a good demonstration from stackexchange.

```
pragma solidity ^0.4.11;

contract AuthorizationManager{
    struct User{
      string userId;
```

```
        uint roleId;
    }

    mapping (string => User[]) companyUserMap;

    function addUser(string _key,string _userId, uint _roleId){
        companyUserMap[_key].push(User(_userId,_roleId));
    }

    function removeSingleUser(string _key){
        companyUserMap[_key].length--;
    }
}
```

3.2 Diary 20171017 Tue

How should a ledger be like? An individual ledger or a public ledger to record everyone?

4 DataToken Alpha 0.2 20171110

Contract vatiable list:

- uint256 public userIndex //index of userInfo array Info
- mapping (address=>bool) public isNotNew //used or not mark
- mapping (address=>uint256) public index //to query in array Info

With the structure described in version 0.1, the dataToken v0.2 is designed as objective oriented and mapping feature of solidity language is used to distinguish different state of users. This time, each private ethereum address is treated as a usable user account, user name and user Id. The contract will create an array of users but rather a map array of users, although a ethereum address is marked as used by a map.
The first feature of this contract is to hold user information. Then the contract need to have functions to add new user and delete user as a ethereum user's wish.

4.1 Diary 20171111 Sat

Behavior test of version 0.2: addUser() expecting when calling this function with an ethereum address, the function:

1. initializes a userInfo element in a userInfo array.

2. mark the message sender as used address by mapping the address to bool value true (used), which by default is false (unused).

3. index increment is needed since the function add userInfo to the same userInfo array each time and the index should be different.

4. store the index of current userInfo by mapping the address to a uint256 value.

Method addUser() is described by the following code:

```
function addUser() isNew public {

    Info.push(userInfo(msg.sender,false,0));
    isNotNew[msg.sender]=true;
    index[msg.sender]=userIndx;
    userIndx+=1;
}
```

Bug:

1. **Default target value (uint256) in a map is 0**
   An used address will have the userIndex mapping value 0, which will be interpreted as position [0] if Info[0] does hold information of a user.
   Trail Fix 1: Initialize userIndex with none zero value, say 1. Position [0] in Info array is initialized with the contract address and other default values.

Limitations:

1. Information of users is stored by an array but the array may be size limited.
   Answer: array is capable to store a very big number of indices like 2^{256}; it maybe not a problem for the scope of a small application on Ethereum. This user information system cound be altered by some more flexible methods in solidity language.

2. A method inside dataToken contract named removeUser() will remove message sender from user list and redeem Ethers according to token balance. Once a user (address) is removed from Info[] array, the position (index) of it will be empty, however, current addUser function won't be able to reuse the released indices.
   Answer: For this version of implementation (alpha V0.2), the removed positions will be left empty and not be used by the contract.

4.2 Diary 20171114 Tue

Design of removeUser() method. The very first question is what have been created by a call of addUser(). As the definition of addUser() in Diary 20171111 Sat, once addUser() is called:

- Info[] will have a new userInfo cascated at the end of it.
- A corresponding position in map **index** is created to hold the index of that userInfo.
- A nother map **isNotNew** will hold the state of whether an addresss is used or not. Being True means this address is not a new user.

Once call of removeUser() will remove the created information above.

- Mark the isNotNew mapping of the address as false. The information of this address won't be removed from **Info**. The corresponding index will be reserved.
- Once this address is marking itself as a user again, addUser() could check whether it's **index** mapping is non-zero, then the old position of this address can be marked as user by just changing the value of **isNotNew** mapping.

Limitations

1. This will also cause another problem that a user must be aware that the token won't be redeemed if withdraw function is not called.

Behavior of this function is as expected so far.