

区块链大作业——项目设计报告（阶段二）

18342040 景致

1. 合约设计

根据给定的需求文档，智能合约的设计需要满足以下的四个功能：

功能一：实现采购商品—签发应收账款交易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。

功能二：实现应收账款的转让上链，轮胎公司从轮毂公司购买一笔轮毂，便将于车企的应收账款单据部分转让给轮毂公司。轮毂公司可以利用这个新的单据去融资或者要求车企到期时归还钱款。

功能三：利用应收账款向银行融资上链，供应链上所有可以利用应收账款单据向银行申请融资。

功能四：应收账款支付结算上链，应收账款单据到期时核心企业向下游企业支付相应的欠款。

我是这么理解的：第一个和第四个功能是完成应收账款的签发和支付，这是比较基础的功能需求，第二和第三个功能则是要完成信用的传递，和传统的金融授信模式不同，在区块链的机制下，我们是允许授信机制中，**核心企业向下游企业的信用传递**，但我们不是直接就允许这种情况存在，而是可以将应收账款进行部分转让供应链其他部分的企业，以完成下游企业向银行融资的需求，这样才能打破传统的授信规则，允许信用传递。

那么让我们来看看我针对这四个功能需求所设计的合约：

最基本的，我们需要预先设置一些变量（结构体）、事件等，方便后续的各种接口设计（以下讨论中，我们以元为单位）：

```
//信用等级
uint[] public creditTable = [1, 50000, 500000, 2000000, 10000000,
50000000, 200000000];

//机构
struct Organization {
    bool isValid;
    address addr;
    uint creditLevel;
    uint amount; // 净资产
    uint credit_amount; // 债权资产
    uint debt_amount; // 负债额度
    mapping (address => Receipt) receipts; // 应收账款列表
}
```

```

struct Receipt {
    bool isValid;
    address debtor;    // 欠债人
    address debtee;    // 债主
    uint amount;       // 债务额
}

```

首先是一个常数数组，用来作为信用等级来衡量一个机构（Organization）的偿债能力，根据该能力来确定供应链的上端和下端，这个值加当前欠款就是机构向银行融资的资金上限；之后是两个结构体，第一个是结构体是机构，里面的成员分别用来表示该企业（机构）的有效性、哈希地址、信用等级、净资产、债权资产、负债额度、应收账款列表（用 mapping 来做一个映射来存储所有的应收账款），第二个结构体是应收账款，本质上看也就是欠条，里面的成员分别用来描述：欠债人、债主和债务额。

我们再来看看我设置的事件：

```

//设置事件，记录日志
//添加机构
event AddOrganizationEvent(int256 ret, address addr, uint256 creditLevel);
//添加资产
event AddAmountEvent(int256 ret, address addr, uint256 amount);
//获取余额
event GetAmountEvent(
    address addr,
    uint256 amount,
    uint256 bond,
    uint256 debt
);
//设置签名
event SignatureEvent(address debtor, address debtee, uint256 amount);
//交易转账
event TransferEvent(
    int256 ret,
    address sponsor,
    address src,
    address dest,
    uint256 amount
);
//向银行融资
event FinancingEvent(int256 ret, address sponsor, uint256 amount);
//支付应收账款
event PaybackEvent(
    int256 ret,
    address debtor,
    address debtee,
    uint256 amount
);

```

这些事件的设置目的在注释里有解释，这个还是挺明显的，发送事件触发参数存储到交易的日志中，在之后的函数设计里面可以很明显的看到，在调用接下来将介绍的函数时，就会有相应的事件触发。

```
uint256 public numOfOrganizations; // 机构总数
mapping(address => Organization) public organizations;
Organization bank; // 特殊机构 => 银行
```

这里需要先注意我设置的合约内部成员，有一个用来记录机构总数，一个是做地址到机构的映射，一个是默认的要有的银行。

```
// 添加机构
function addOrganization(address addr, uint256 creditLevel)
    public
    returns (bool)
{
    if (msg.sender != bank.addr) {
        emit AddOrganizationEvent(-1, addr, creditLevel);
        return false;
    }
    Organization memory org = Organization(
        true,
        addr,
        creditLevel,
        0,
        0,
        0
    );
    organizations[addr] = org;
    numOfOrganizations += 1;
    emit AddOrganizationEvent(0, addr, creditLevel);
    return true;
}

// 机构添加资产
function addAmount(address addr, uint256 amount) public returns (bool) {
    if (msg.sender != bank.addr) {
        emit AddAmountEvent(-1, addr, amount);
        return false;
    }
    organizations[addr].amount += amount;
    emit AddAmountEvent(0, addr, amount);
    return true;
}

// 获取机构资产
function getAmount() public returns (uint256) {
    emit GetAmountEvent(
        msg.sender,
        organizations[msg.sender].amount,
        organizations[msg.sender].bond_amount,
        organizations[msg.sender].debt_amount
    );
    return organizations[msg.sender].amount;
}
```

```
// 构造函数
constructor(address bankAddr) public {
    // 初始化银行资产
    bank.isValid = true;
    bank.amount = 1000000000000000;
    bank.creditLevel = 10000;
    bank.addr = bankAddr;
    organizations[bankAddr] = bank; // 添加银行至所有机构
    numOfOrganizations = 1;
}
```

各个函数的功能和目的都在注释和函数名里面表达的很清楚了，对应的参数也是比较好理解的，比如 `addOrganization(address addr, uint256 creditLevel) public returns (bool)`，这里是为了添加机构，参数是添加机构的新地址和对应的信用等级，返回是否创建成功，这里创建的是一个有效的，地址为第一个参数的，资产、债权、负债均为0的新机构。另外，这里还有一个构造函数创建合约时初始化银行机构，并给予一个较大数作为资产，因为银行是必不可少的，所以我这里作为构造函数加入，而不是在之后用其他接口加入，符合实际逻辑。

以上就是我设计的合约中的基础部分，这些都是满足最基础的编程需求所设计的，接下来我们来看为了实现所提出的4个功能，我所设计的接口：

```
/*
 * 功能：签发应收账款
 * 发起者：欠债人
 * 参数：address debtee 债主地址, uint amount 金额
 */
function signature(address debtee, uint256 amount) public {
    address debtor = msg.sender;
    if (organizations[debtee].receipts[debtor].isValid == false) {
        organizations[debtee].receipts[debtor] = Receipt(
            true,
            debtor,
            debtee,
            amount
        );
    } else {
        organizations[debtee].receipts[debtor].amount += amount;
    }
    organizations[debtee].bond_amount += amount;
    organizations[debtor].debt_amount += amount;
    emit SignatureEvent(msg.sender, debtee, amount);
}
```

这个是最基础的功能一，就是注释里有写的钱不够打欠条，由钱不够的欠债人作为调用者，参数的第一个是债主，第二个参数是欠款金额。这里，先判断这个该债主是否已经有该欠债人的应收账款（欠条），如果没有，则创建金额为第二个参数、欠债人为调用者也就是 `msg.sender`、债主为第一个参数的新应收账款（欠条），如果之前已经有欠过债了，则只需要在之前的欠款上加上对应金额即可。

```

/*
 * 功能：应收账款转让（债务转移）B 将 A -> C
 * 发起者：B
 * 参数：address A 欠债人已有债券地址，address C 转移至的机构
 * 逻辑：A 欠 B 钱，B 欠 C 钱，B 将A对B的债务转移成A欠C
 */
function transfer(address A, address C) public returns (bool) {
    address B = msg.sender;
    uint256 ret = 0;
    //要有三个人中传递性欠钱的拓扑结构
    if (organizations[B].receipts[A].isValid == false ||
        organizations[C].receipts[B].isValid == false) {
        ret = -1;
        emit TransferEvent(ret, B, A, C, 0);
        return false;
    }
    // 获取欠款金额
    uint256 ab = organizations[B].receipts[A].amount;
    uint256 bc = organizations[C].receipts[B].amount;
    uint256 ac = 0;
    Receipt memory newReceiptA2C;
    if (ab > bc) {
        // 将A对B的债务 转移到A对C
        ac = bc;
        newReceiptA2C = Receipt(true, A, C, ac);
        // 相应减少B对C的债务，A对B的债务
        organizations[B].receipts[A].amount -= newReceiptA2C.amount;
        organizations[C].receipts[B].isValid = false;
    } else {
        // 将A对B的债务 转移到A对C
        ac = ab;
        newReceiptA2C = Receipt(true, A, C, ac);
        // 相应减少B对C的债务 A对B的债务
        organizations[C].receipts[B].amount -= newReceiptA2C.amount;
        organizations[B].receipts[A].isValid = false;
        if (organizations[C].receipts[B].amount == 0) {
            organizations[C].receipts[B].isValid = false;
        }
    }
    organizations[B].bond_amount -= ac;
    organizations[B].debt_amount -= ac;
    if (organizations[C].receipts[A].isValid == false) {
        organizations[C].receipts[A] = newReceiptA2C;
    } else {
        organizations[C].receipts[A].amount += newReceiptA2C.amount;
    }
    ret = 0;
    emit TransferEvent(ret, B, A, C, ac);
    return true;
}

```

这个是第二个功能，是实现债务转移的函数。函数过程是：首先确定三方存在确定的传递欠钱的结构，即 A->B->C，函数调用者为 B（因为是 B 需要把债务进行转移，从而让 A 欠 B 的钱全部变为 A 欠 C 的钱），之后将 A 的债务转移到 C，B 的债务相应的减少，以此实现债务的转移。特别地，这里需要强调两点，一个是这里调用的先决条件是 C 没有 A 的任何欠款，因为这才是授信传递的意义所在，否则核心企业的应收账款已经足够让金融机构对其信用等级足够高了，另一个是这里没有设置转移的债务金额，因为是要全部转移，在转移之前呢就要做好相应的资产处理。

下面来看第三个功能：

```

/*
 * 功能：利用应收账款向银行融资
 * 发起者：贷款机构
 * 参数：uint amount 贷款金额
 * 逻辑：根据贷款机构总资产、贷款金额、信用额度判断是否给予贷款。
 *       贷款后，机构净资产增加
 */
function financing(uint256 amount) public returns (bool) {
    int256 ret = 0;
    address debtor = msg.sender;
    uint256 credit = organizations[debtor].creditLevel; // 贷款机构信用额度
    uint256 org_amount = organizations[debtor].amount; // 贷款机构现有净资产
    uint256 bond_amount = organizations[debtor].bond_amount; // 贷款机构已有
    债券金额
    uint256 debt_amount = organizations[debtor].debt_amount; // 贷款机构负债
    金额

    Receipt memory receipt = Receipt(true, debtor, bank.addr, amount);
    if (bank.receipts[debtor].isValid == false) {
        if (
            amount <
            creditTable[credit] + org_amount + bond_amount - debt_amount
        ) {
            bank.receipts[debtor] = receipt;
            organizations[debtor].amount += amount;
            emit FinancingEvent(ret, debtor, amount);
            return true;
        }
    } else {
        if (
            amount + bank.receipts[debtor].amount <
            creditTable[credit] + org_amount + bond_amount - debt_amount
        ) {
            bank.receipts[debtor].amount += amount;
            organizations[debtor].amount += amount;
            emit FinancingEvent(ret, debtor, amount);
            return true;
        }
    }
}

```

```

    ret = -1;
    emit FinancingEvent(ret, debtor, 0);
    return false;
}

```

这里就是向银行融资的函数了，其过程就是：首先衡量其信用额度，即净资产+债券金额（这里还要债的来源是否可信任）-负债金额，如果信用额度的审查通过，则向该申请企业借款，更改其净资产的值，否则这次融资申请就会失败。

```

/*
 * 功能：下游企业要求核心企业支付欠款（到期还钱）
 * 发起者：下游企业
 * 参数：address debtor 核心企业
 * 逻辑：欠债还钱
 */
function payback(address debtor) public returns (bool) {
    int256 ret = 0; // 0 还款成功 1 没有欠钱 2 资产不够
    address debtee = msg.sender; // 下游企业地址
    Organization storage from = organizations[debtor]; // 核心企业
    Organization storage to = organizations[debtee]; // 下游企业

    // 若核心企业没有欠钱
    if (to.receipts[debtor].isValid == false) {
        ret = 1;
        emit PaybackEvent(ret, debtor, debtee, 0);
        return false;
    }
    // 若核心企业现有资产不够
    if (from.amount < to.receipts[debtor].amount) {
        ret = 2;
        emit PaybackEvent(ret, debtor, debtee, 0);
        return false;
    }
    // 核心企业还钱
    from.amount -= to.receipts[debtor].amount;
    to.amount += to.receipts[debtor].amount;
    if (debtor != bank.addr) {
        from.debt_amount -= to.receipts[debtor].amount;
        to.bond_amount -= to.receipts[debtor].amount;
    }

    // 删除该条债务记录
    to.receipts[debtor].isValid = false;
    emit PaybackEvent(ret, debtor, debtee, to.receipts[debtor].amount);
    return true;
}

```

最后也是一个比较基础的功能，就是欠债还钱，首先检查是否欠钱以及是否有能力还钱，如果可以则进行资产转移，否则就会返回 false 宣布还钱失败。

这就是我的整个智能合约的设计，下面我将展示该合约部署后的实际效果和功能实现。在这里，我是在 Remix 上部署的智能合约，用这个平台来查看合约的实际效果。

首先我们对合约进行编译：

SOLIDITY COMPILER

COMPILER

0.4.26+commit.4563c3fc

☐ Include nightly builds

LANGUAGE

Solidity

EVM VERSION

compiler default

COMPILER CONFIGURATION

☐ Auto compile

☐ Enable optimization 200

☐ Hide warnings

Compile myChain.sol

CONTRACT

myChain (myChain.sol)

Publish on Swarm

Publish on Ipfs

Compilation Details

ABI Bytecode

```

1 pragma solidity ^0.4.21;
2
3 contract myChain {
4     //设置事件，记录日志
5     //添加机构
6     event AddOrganizationEvent(int256 ret, address addr, uint256 creditLevel);
7     //添加资产
8     event AddAmountEvent(int256 ret, address addr, uint256 amount);
9     //获取余额
10    event GetAmountEvent(
11        address addr,
12        uint256 amount,
13        uint256 bond,
14        uint256 debt
15    );
16    //设置签名
17    event SignatureEvent(address debtor, address debtee, uint256 amount);
18    //债务转移
19    event TransferEvent(
20        int256 ret,
21        address sponsor,
22        address src,
23        address dest,
24        uint256 amount
25    );
26    //向银行融资
27    event FinancingEvent(int256 ret, address sponsor, uint256 amount);
28    //支付应收账款
29    event PaybackEvent(
30        int256 ret

```

creation of myChain errored: VM error: out of gas. out of gas The transaction ran out of gas. get more information.

creation of myChain pending...

[vm] from: 0x787...cabaB to: myChain.(constructor) value: 0 wei data: 0x608...caba

call to myChain.numOfOrganizations

可以看到，这里编译是成功的。这里我用的是 0.4.26 版本的 solidity

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

JavaScript VM

ACCOUNT

0x17F...8c372 (100 ether)

GAS LIMIT

30000000000

VALUE

2000 wei

CONTRACT

myChain - browser/myChain.sol

Deploy

PUBLISH TO IPFS

```

106 organizations[msg.sender].amount,
107 organizations[msg.sender].credit_amount,
108 organizations[msg.sender].debt_amount
109 );
110 return organizations[msg.sender].amount;
111 }
112 // 构造函数
113 constructor(address bankAddr) public {
114     // 初始化银行数据
115     bank.isValid = true;
116     bank.amount = 1000000000000000;
117     bank.creditLevel = 100000;
118     bank.addr = bankAddr;
119     organizations[bankAddr] = bank; // 添加银行至所有机构
120     numOfOrganizations = 1;
121 }
122 /*
123 * 功能：签发应收账款（钱不够打欠条）
124 * 发起者：欠债人
125 * 参数：address debtor, 债主地址, uint amount, 金额
126 */
127

```

creation of myChain pending...

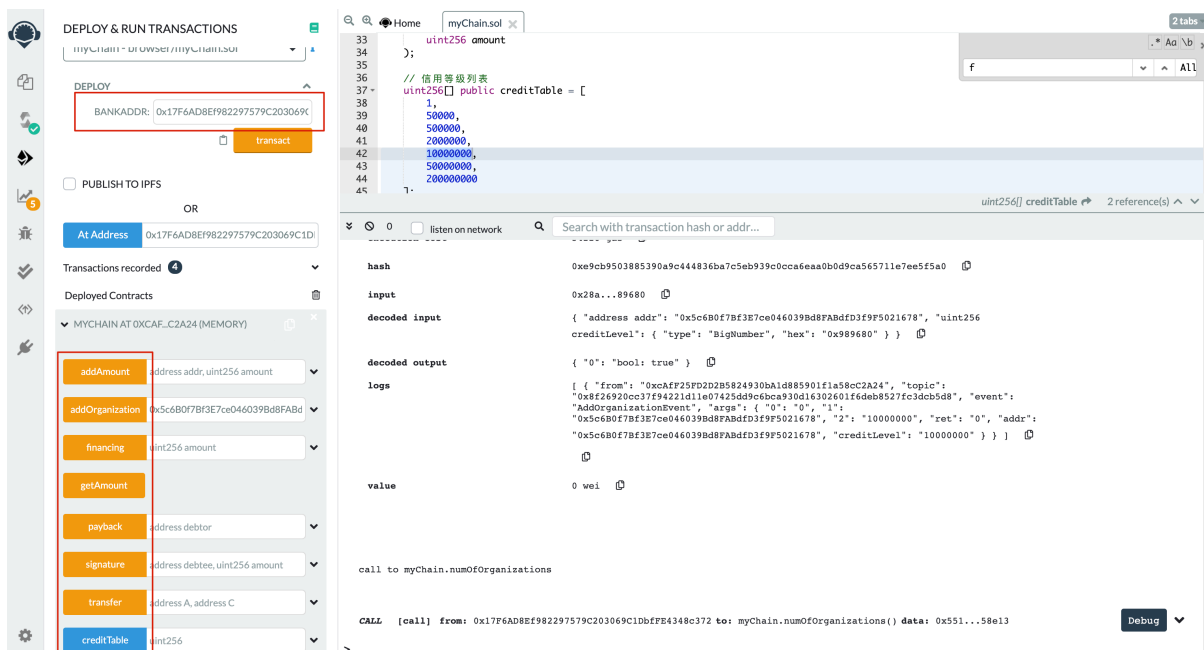
[vm] from: 0x17F...8c372 to: myChain.(constructor) value: 0 wei data: 0x608...8c372 logs: 0 hash: 0x554...5c7e4

接着，我们选择一个虚拟机提供的虚拟账号，作为构造函数的参数，也就是银行的地址，我们来部署合约：

creation of myChain pending...

[vm] from: 0x17F...8c372 to: myChain.(constructor) value: 0 wei data: 0x608...8c372 logs: 0 hash: 0x554...5c7e4

部署成功之后，左侧就出现了我们之前所定义的一系列接口：

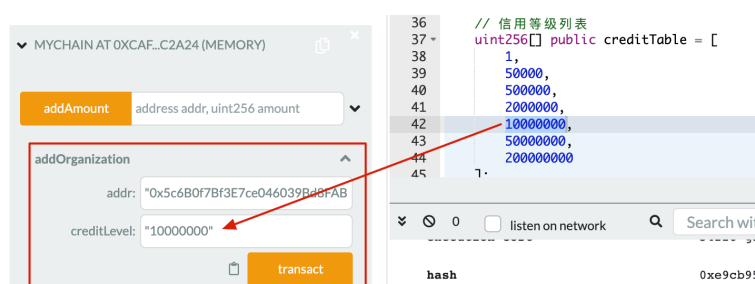


那么我们就可以开始按照文档描述的内容进行测试：

① 添加机构：调用 addOranization 接口

我们选择一个地址添加车企
(宝马)，其地址为
0x5c6B0f7Bf3E7ce046039B
d8FABdF3f9F5021678

并将其信用额度设为
10000000



右边我们看到，
我们添加成功

```
[vm] from: 0x17F...8c372 to: myChain.addOrganization(address,uint256) 0xcAf...C2A24 value: 0 wei data: 0x28a...89680 logs: 1
hash: 0xe9c...5f5a0

status true Transaction mined and execution succeed

transaction hash 0xe9cb9503885390a9c444836ba7c5eb939c0cca6eaa0b0d9ca565711e7ee5f5a0

from 0x17F6AD8EF982297579C203069C1DbfFE4348c372

to myChain.addOrganization(address,uint256) 0xcAfF25FD2D2B5824930ba1d885901f1a58c2A24

gas 3000000000000000 gas

transaction cost 77228 gas

execution cost 54228 gas

hash 0xe9cb9503885390a9c444836ba7c5eb939c0cca6eaa0b0d9ca565711e7ee5f5a0

input 0x28a...89680

decoded input { "address addr": "0x5c6B0f7Bf3E7ce046039Bd8FABdF3f9F5021678", "uint256 creditLevel": { "type": "BigNumber", "hex": "0x989680" } }

decoded output { "0": "bool: true" }

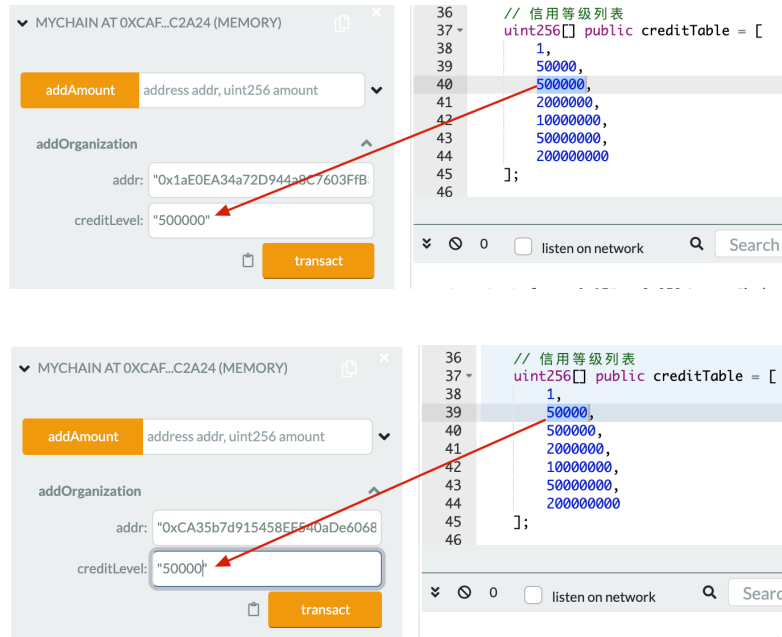
logs [ { "from": "0xcAfF25FD2D2B5824930ba1d885901f1a58c2A24", "topic": "0x8f26920cc37f94221d11e07425dd9c6bca930d16302601f6deb8527fc3dcb5d8", "event": "AddOrganizationEvent", "args": { "0": "0", "1": "0x5c6B0f7Bf3E7ce046039Bd8FABdF3f9F5021678", "2": "10000000", "ret": "0", "addr": "0x5c6B0f7Bf3E7ce046039Bd8FABdF3f9F5021678", "creditLevel": "10000000" } } ]

value 0 wei
```

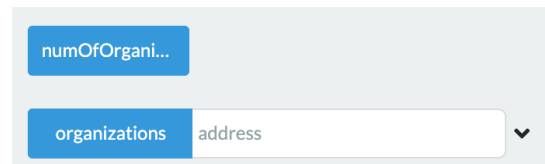
2020年12月4日 星期五

我们再添加轮胎和轮毂公司，其地址分别为
0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C
和
0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c

其信用额度分别设为
500000 和 50000



这里我们通过查看 numOfOrganization 变量和
利用 organizations 这个 mapping 接口来查看
对应的机构信息。



这里我查看一下机构数：

call to myChain.numOfOrganizations

```
CALL [call] from: 0x17F6AD8Ef982297579C203069C1DbfFE4348c372 to: myChain.numOfOrganizations() data: 0x551...58e13

transaction hash      0xed56d95f3f7e3f7f5695be9ea066bd77d977392d35832fcef4c5df488a992c4d
from                  0x17F6AD8Ef982297579C203069C1DbfFE4348c372
to                    myChain.numOfOrganizations() 0xcAfF25FD2D2B5824930bA1d885901f1a58cC2A24
transaction cost      22310 gas (Cost only applies when called by a contract)
execution cost        1038 gas (Cost only applies when called by a contract)
hash                  0xed56d95f3f7e3f7f5695be9ea066bd77d977392d35832fcef4c5df488a992c4d
input                  0x551...58e13
decoded input          {}
decoded output          { "0": "uint256: 4" }
logs                  []
```

可以看到现在一共是 4 个机构：银行、车企、轮胎和轮毂公司，符合我们的需求。

② 现在我们来往机构里添加净资产：

给车企添加 500000 资产：

addAmount

addr: "0x5c6B0f7Bf3E7ce046039Bd8FAB"

amount: "500000"

transact

decoded output

`{ "0": "bool: true" }`

logs

```
[ { "from": "0xcAfF25FD2D2B5824930bA1d885901f1a58cC2A24", "topic":
"0xa44915eae15d36433ea2ac5320a7c610f0902078ba0c28031dab2afeb7b649c", "event":
"AddAmountEvent", "args": { "0": "0", "1":
"0x5c6B0f7Bf3E7ce046039Bd8FABd3f9F5021678", "2": "500000", "ret": "0", "addr":
"0x5c6B0f7Bf3E7ce046039Bd8FABd3f9F5021678", "amount": "500000" } } ]
```

给轮胎公司添加 100000 资产：

addAmount

addr: "0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C"

amount: "100000"

transact

decoded output

`{ "0": "bool: true" }`

logs

```
[ { "from": "0xcAfF25FD2D2B5824930bA1d885901f1a58cC2A24", "topic":
"0xa44915eae15d36433ea2ac5320a7c610f0902078ba0c28031dab2afeb7b649c", "event":
"AddAmountEvent", "args": { "0": "0", "1":
"0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "2": "100000", "ret": "0", "addr":
"0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "amount": "100000" } } ]
```

给轮胎公司添加 10000 资产：

addAmount

addr: "0xCA35b7d915458EF540aDe6068"

amount: "10000"

transact

decoded output

`{ "0": "bool: true" }`

logs

```
[ { "from": "0xcAfF25FD2D2B5824930bA1d885901f1a58cC2A24", "topic":
"0xa44915eae15d36433ea2ac5320a7c610f0902078ba0c28031dab2afeb7b649c", "event":
"AddAmountEvent", "args": { "0": "0", "1":
"0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "2": "10000", "ret": "0", "addr":
"0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "amount": "10000" } } ]
```

2020年12月4日 星期五

③ 现在来进行交易，所有的交易都是先设置应付账款，再进行支付：

功能一测试：假设现在车企需要购入一批轮胎共 200000 元，我们切换到车企账号：

ACCOUNT 

调用 `signature` 函数来创造应付账款：

signature

debtee: "0x1aE0EA34a72D944a8C7603FfB"


amount: "200000"


transact


logs

```
[ { "from": "0xcAfF25FD2D2B5824930bA1d885901f1a58c2A24", "topic": "0xb5946a1a30d17baad50394a138685a5bf927c5ad3ecd9fd917fal2748b58e46", "event": "SignatureEvent", "args": { "0": "0x5c6B0f7Bf3E7ce046039Bd8FABdFD3f9F5021678", "1": "0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "2": "200000", "debtor": "0x5c6B0f7Bf3E7ce046039Bd8FABdFD3f9F5021678", "debtee": "0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "amount": "200000" } } ]
```

现在，假设轮胎公司急缺流水，而且恰好车企有五十万流水可以立刻支付全部账款，这时候，轮胎公司要求车企付款（先切换为轮胎公司）：


ACCOUNT 

0x1aE...E454C (99.999999999999) 

payback 

debtor:

0x5c6B0f7Bf3E7ce046039Bd8FAB



transact

logs

```
[ { "from": "0xcAfF25FD2D2B5824930bA1d885901f1a58cC2A24", "topic":
"0x567f3ddaf78cc76d9bf274e37837b41f83c763ef07927ff1917d93e5f6707a4c", "event":
"PaybackEvent", "args": { "0": "0", "1":
"0x5c6B0f7Bf3E7ce046039Bd8FABdfD3f9F5021678", "2":
"0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "3": "200000", "ret": "0", "debtor":
"0x5c6B0f7Bf3E7ce046039Bd8FABdfD3f9F5021678", "debtee":
"0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "amount": "200000" } } ]
```

transact to myChain.getAmount pending ...

从日志可以看出来，这里是直接根据金额，自动转账。调用轮胎公司的 `getAmount`：

```
[vsn] from: 0x1aE...E454C to: myChain.getAmount() 0xcAf...C2A24 value: 0 wei data: 0xd32...1fe29 logs: 1 hash: 0x658...f7962 [Debug] ^
```

```
status                                true Transaction mined and execution succeed
transaction hash                       0x658a1e758880617d3d4f087647fbf585ae26d2348706937d5ecae05f3a8f7962 ⓘ
from                                  0x1ae0EA34a72D944a8C7603fFB3eC30a6699E454C ⓘ
to                                    myChain.getAmount() 0xcAf25FD2D2B5824930bAld885901f1a58c2A24 ⓘ
gas                                    3000000000000000 gas ⓘ
transaction cost                       27139 gas ⓘ
execution cost                         5867 gas ⓘ
hash                                   0x658a1e758880617d3d4f087647fbf585ae26d2348706937d5ecae05f3a8f7962 ⓘ
input                                  0xd32...1fe29 ⓘ
decoded input                          {} ⓘ
decoded output                         { "0": "uint256: 3000000" } ⓘ
```

可以看到，轮胎公司现在有 300000 资产，有十万的初始资产和车企支付的二十万资产组成。功能一测试成功！

功能二测试：

现在，我们先让车企从轮胎公司购买 600000 元的轮胎，（切换账号就不演示了）

对应的，我们看到轮胎公司现在拥有车企的六十万欠款：

现在，轮胎公司需要一批轮毂，价值共 200000 元：

现在，轮胎公司希望自己欠轮毂公司的账款由车企支付一部分，也就是这二十万由车企还给轮毂公司，车企再只需要还给轮胎公司四十万，于是进行欠款转移：

✓ [vm] from: 0x1aE...E454C to: myChain.transfer(address,address) 0x62a...03549 value: 0 wei data: 0xba4...a733c logs: 1
hash: 0x5ab...5a0dd

status	true Transaction mined and execution succeed
transaction hash	0x5ab717cdd70e2f8953982c112d5c6e8d49bd501649dcf4ba3a792de3d845a0dd
from	0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C
to	myChain.transfer(address,address) 0x62ae6Ba9EDfB884625e2C23Bf8b07a0d19203549
gas	3000000000000000 gas
transaction cost	104989 gas
execution cost	95901 gas
hash	0x5ab717cdd70e2f8953982c112d5c6e8d49bd501649dcf4ba3a792de3d845a0dd
input	0xba4...a733c
decoded input	{ "address A": "0x5c6B0f7Bf3E7ce046039Bd8FABdF3f9F5021678", "address C": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c" }
decoded output	{ "0": "bool: true" }
logs	[{ "from": "0x62ae6Ba9EDfB884625e2C23Bf8b07a0d19203549", "topic": "0xf5df2a036401226dd077ea8b6a5658b51e19df596299e908976055e2fc241e93", "event": "TransferEvent", "args": { "0": "0", "1": "0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "2": "0x5c6B0f7Bf3E7ce046039Bd8FABdF3f9F5021678", "3": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "4": "200000", "ret": "0", "sponsor": "0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C", "src": "0x5c6B0f7Bf3E7ce046039Bd8FABdF3f9F5021678", "dest": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "amount": "200000" } }]
value	0 wei

可以看到，这里是成功的转移了这二十万欠款，我们来分别查看现在的欠款状态：
车企还是一样的，总共欠款 60 万；

organizations 0x5c6B0f7Bf3E7ce046039Bd8FABd

- 0: bool: isValid true
- 1: address: addr 0x5c6B0f7Bf3E7ce046039Bd8FABdF3f9F5021678
- 2: uint256: creditLevel 10000000
- 3: uint256: amount 300000
- 4: uint256: credit_amount 0
- 5: uint256: debt_amount 600000

轮胎公司现在就不欠那 20 万了可以看到，而现在也有 40 万的待收款

organizations 0x1aE0EA34a72D944a8C7603FfB3

- 0: bool: isValid true
- 1: address: addr 0x1aE0EA34a72D944a8C7603FfB3eC30a6669E454C
- 2: uint256: creditLevel 500000
- 3: uint256: amount 300000
- 4: uint256: credit_amount 400000
- 5: uint256: debt_amount 0

而轮毂公司现在就多了 20 万欠款待收。

organizations
0xCA35b7d915458EF540aDe6068c
▼

0: bool: isValid true
1: address: addr 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c
2: uint256: creditLevel 50000
3: uint256: amount 10000
4: uint256: credit_amount 200000
5: uint256: debt_amount 0

这样，我们就完成了欠款的转移，所以功能二测试成功！

功能三测试：

因为拥有了车企的欠款，那么轮毂公司信用等级，（上文有说到，其公式为，基本信用额度+当前应收账款-当前欠款）得到提升，那么就可以向银行申请更多融资，原本融资额度的只有自己信用额度，也就是上图可以看到的 5 万，但现在有了新的上限。

现在我们申请更大的额度试试：

financing
^

amount: "1000000"

transact

transact to myChain.financing pending ...

```

[vm] from: 0xCA3...a733c to: myChain.financing(uint256) 0x26f...0a4A4 value: 0 wei data: 0x89d...f4240 logs: 1 hash: 0xf33...3f52c

status true Transaction mined and execution succeed
transaction hash 0xf3340a4a5e70dddb37f0b94cb31764d672d5d69021e794cb696f0a495dd3f52c
from 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c
to myChain.financing(uint256) 0x26f15D23e7bbc62A16F014A75B75a80462F0a4A4
gas 3000000000000000 gas
transaction cost 29461 gas
execution cost 7869 gas
hash 0xf3340a4a5e70dddb37f0b94cb31764d672d5d69021e794cb696f0a495dd3f52c
input 0x89d...f4240
decoded input { "uint256 amount": { "type": "BigNumber", "hex": "0x0f4240" } }
decoded output { "0": "bool: false" }
logs [ { "from": "0x26f15D23e7bbc62A16F014A75B75a80462F0a4A4", "topic": "0x77df711616b31b37032556de5ce13156f22d12cd67f73b7504c4829dd87adfd5", "event": "FinancingEvent", "args": { "0": "-1", "1": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "2": "0", "ret": "-1", "sponsor": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "amount": "0" } } ]
value 0 wei

```


这里可以看到，这里是失败的，因为申请的额度，即使有了车企的欠款，一百万仍然是个过大的数字，所以银行不可以接受。

2020年12月4日 星期五

那么我们来试试一个略少，但是又在轮毂公司本身仅有的五万融资额度上限之上的一个金额：

financing

amount: "150000"



transact

```
transact to myChain.financing pending ...
```

```

[✓] from: 0xCA3...a733c to: myChain.financeing(uint256) 0x26f...0a4A4 value: 0 wei data: 0x89d...249f0 logs: 1 hash: 0x3bd...563be

status true Transaction mined and execution succeed

transaction hash 0x3bd888ada5b87d83c4a25e1c66978fad80c0f6178458c270e7db085d923563be ⓘ

from 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c ⓘ

to myChain.financeing(uint256) 0x26f15D23e7bbc62A16F014A75B75a80462F0a4A4 ⓘ

gas 30000000000000 gas ⓘ

transaction cost 99006 gas ⓘ

execution cost 77414 gas ⓘ

hash 0x3bd888ada5b87d83c4a25e1c66978fad80c0f6178458c270e7db085d923563be ⓘ

input 0x89d...249f0 ⓘ

decoded input { "uint256 amount": { "type": "BigNumber", "hex": "0x0249f0" } } ⓘ

decoded output { "0": "bool: true" } ⓘ

logs [ { "from": "0x26f15D23e7bbc62A16F014A75B75a80462F0a4A4", "topic": "0x77df711616b31b37032556de5ce13156f22d12cd67f73b7504c4829dd87adfd5", "event": "FinancingEvent", "args": { "0": "0", "1": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "2": "150000", "ret": "0", "sponsor": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "amount": "150000" } } ] ⓘ ⓘ

value 0 wei ⓘ

```

可以看到，这里是申请融资成功的，即使是申请大于轮毂公司原本的信用额度五万的十五万融资，这里银行也是批准的因为有了车企的应收账款。查看轮毂公司的情况：

```
organizations 0xCA35b7d915458EF540aDe6068c
0:    bool: isValid true
1:    address: addr 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c
2:    uint256: creditLevel 50000
3:    uint256: amount 160000
4:    uint256: credit_amount 200000
5:    uint256: debt_amount 0
```

这里可以看到，轮毂公司现在的净资产为十六万，是原本的一万初始资金和融资得到的十五万。功能三测试成功！

功能四测试：

这个功能就是最基本的需求，让有应收账款的下游企业要求核心企业支付该款项。这一点其实在功能一的时候有测试过，这里再单独测试一次。

这里直接让车企支付上文中欠轮毂公司的二十万欠款：

支付前，车企的情况：

```
organizations      0x5c6B0f7Bf3E7ce046039Bd8FABd
0:    bool: isValid true
1:    address: addr 0x5c6B0f7Bf3E7ce046039Bd8FABdFD3f9F5021678
2:    uint256: creditLevel 1000000
3:    uint256: amount 300000
4:    uint256: credit_amount 0
5:    uint256: debt_amount 600000
```

可以看到有六十万待支付款项，里面包含轮胎公司的四十万，和轮毂公司的二十万。

切换到轮毂公司，要求车企支付：

payback

debtor:

0x5c6B0f7Bf3E7ce046039Bd8FAB

transact

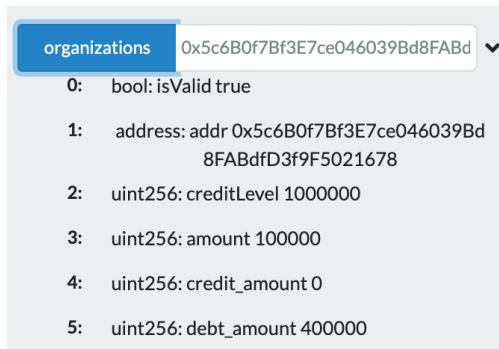
```
transact to myChain.payback pending ...
```

```
[vm] from: 0xC3A...a733c to: myChain.payback(address) 0x26f...004A4 value: 0 wei data: 0x340...21678 logs: 1 hash: 0xe3f...1ff02
```

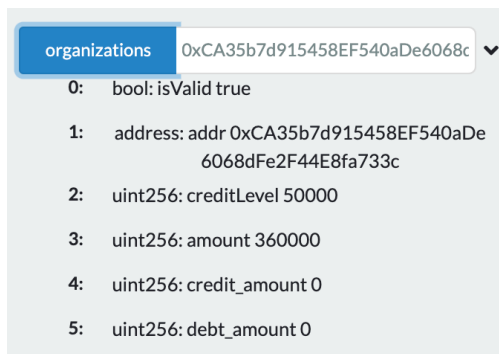
status	true Transaction mined and execution succeed
transaction hash	0xe3f2844d7b2dd088fbf4ce49e24cfef6e5c4495c45c1cbb121568cd0461ff02
from	0xC3A35b7d915458EF540aDe068dFe2F44E8fa733c
to	myChain.payback(address) 0x26f15D23e7bbC62A16F014A75B7a80462F0A4A4
gas	3000000000000000 gas
transaction cost	32615 gas
execution cost	39935 gas
hash	0xe3f2844d7b2dd088fbf4ce49e24cfef6e5c4495c45c1cbb121568cd0461ff02
input	0x340...21678
decoded input	{ "address debtor": "0x5c6B0f7Bf3E7ce046039Bd8FABdf3f9F5021678" }
decoded output	{ "0": "bool: true" }
logs	[{ "from": "0x26f15D23e7bbC62A16F014A75B7a80462F0A4A4", "topic": "0x56f73ddaf78cc76d9bf274e37837b41f83c763ef07927f1917d93e5f6707a4c", "event": "PaybackEvent", "args": { "0": "0", "1": "0x5c6B0f7Bf3E7ce046039Bd8FABdf3f9F5021678", "2": "0xC3A35b7d915458EF540aDe068dFe2F44E8fa733c", "3": "200000", "ret": "0", "debtor": "0x5c6B0f7Bf3E7ce046039Bd8FABdf3f9F5021678", "debtee": "0xC3A35b7d915458EF540aDe068dFe2F44E8fa733c", "amount": "200000" } }]
value	0 wei

从交易日志可以看到，支付成功。

分别查看各企业的状态：



可以看到车企支付了这二十万，剩下十万余额。



可以看到轮毂公司现在又获得了二十万款项，现在净资产达到了三十六万。

功能四测试成功！那么，我们就完成了对我编写的智能合约的测试。

到这里，就完成了我区块链项目的合约设计和功能测试的介绍了，也是目前我的区块链项目的进展介绍。这次我是选择单人完成，不组队，一方面是因为我认为工作量其实就是单人大作业的工作量，而且往年都是单人，没有组队的必要，另一方面我觉得这个项目会很有趣想自己尝试挑战，所以决定自己单人做了。

智能合约的代码我 push 在本次项目的 repository 上了：https://github.com/VarusJ/block_chain2020，之后会继续更新整个项目的完成。