中山大学数据科学与计算机学院本科生实验报告

（2019 年秋季学期）

课程名称：区块链原理与技术　　任课教师： 郑子彬

年级	16	专业（方向）	电子政务
学号	16340216	姓名	王健豪
电话		Email	1412106667@qq.com
开始日期		完成日期	

一、 项目背景

基于已有的开源区块链系统 FISCO-BCOS 以联盟链为主，开发基于区块链或区块链智能合约的供应链金融平台，实现供应链应收账款资产的溯源、流转。

在某个交易中，轮胎公司从轮毂公司购买了一批轮毂，但由于租金暂时短缺向轮胎公司签订了 500 万的应收账款单据，承诺 1 年后归还轮胎公司 500 万。当轮毂公司想利用这个应收账款单据向金融机构借款融资的时候，金融机构因为不认可轮胎公司的还款能力，需要对轮胎公司进行详细的信用分析以评估其还款能力同时验证应收账款单据的真实性，才能决定是否借款给轮毂公司。这个过程将增加很多经济成本，而这个问题主要是由于该车企的信用无法在整个供应链中传递以及交易信息不透明化所导致的。

将供应链上的每一笔交易和应收账款单据上链，同时引入第三方可信机构来确认这些信息的交易，例如银行，物流公司等，确保交易和单据的真实性。同时，支持应收账款的转让，融资，清算等，让核心企业的信用可以传递到供应链的下游企业，减小中小企业的融资难度。

二、 方案设 计

Part1 存储设计

两个存储对象 Bill 和 Organization，用 solidity 语言结构体表示

```
struct Bill {
  address owner;       // 账单欠款方
  address borrower;    // 账单借款方
  address verifier;    // 验证方银行
  uint value;          // 账单金额
  uint replayData;     // 应还日期
}

enum Tag { Bank, Enterprise, Supplier }        // 三种 tag 属性表示银行，供应商和企业

struct Organization {
  Tag tag;             // 标识身份
  uint balance;        // 一种信用额度，初始为 0，别人欠钱则增加相应的金额，凭此融资
  uint leftValue;      // 组织资产
  bool isGuaranteed;   // 是否担保过，只能是企业被担保，担保是初始信用额度 balance+1000
  address guaranteeBank;       // 担保银行
  mapping (address => uint) finaning; // 对银行来说，是融资公司地址和对应金额，对公司来说是融资银行地址和对应金额，公司可以进行多次融资前提是 balance 不小于 0
}
```

```
address admin;          // 创建合约的账户，又是管理者
address[] registerList;         // 已经注册在此节点的公司
Bill[] billList;                // 所有的账单
mapping (address => Organization) organizations;        // 账户地址对应的公司
```

Part2 函数设计

整体设计思路

银行　企业　供应商

交易函数　逻辑判断链

1. 函数的调用者->bank
2. 首先假设 a 向 b 打欠条首先不能存在其他欠条，或者说其他欠条都已经还完了
3. 要做交易，a 向 b 打欠条，不管 b 的身份如何让 b 同意，需要银行担保，银行为什么替 a 担保，因为 a 是大企业，有一定的风险承担能力，如果 a 是供应商的话，银行为什么要替 a 担保，前提是 a 有其他人欠他的钱，这样打欠条的话，不能超过 a 所持有的所有未收债款。这是银行担保供应商的限制，银行给企业担保的限制是什么（我们假定大企业不需要融资），比如银行给企业的担保额度为 1000 元，则限制为大企业所持有的所有未收债款+1000>大企业给其他企业打欠款。同时企业最多只能有 1 家银行担保，但是打欠条的担保银行可以有很多。给一个担保函数，设置企业的 balance 为 1000，设置担保银行
4. 上面的担保函数这样设置，第一个欠条验证银行给大企业担保，如果大企业还没有担保的话
   a) 我们可不可以将担保函数设置为融资函数，首先大企业只能融资一次，

融资函数

1. 函数的调用者->bank 也是机构请求融资的那个银行
2. 如同上一问我们假设大企业不需要融资，或者说没有意义，需要判断融资对象为供应商
3. a 向银行融资，银行为什么给 a 融资，那必定是 a 持有未收债款>a 欠的所有债款，剩下的便是融资额度

转让函数 a b c value 执行步骤和逻辑条件

1. b 欠 c 钱，value<=这笔钱
2. a 欠 b 钱，value<=这笔钱
3. 进行 bill 处理，a，b，c 的 bill 的修改，同时还有银行的 bill 修改，balance 不变，如果 bill 金额为 0，则删除

清算函数

1. 到时间进行资产的转移
2. 有个问题是这里的资产到底是什么资产，我们设置的钱还是以太币，最后经过讨论决定还是先是设置 leftValue 变量作为资产值，资产的清算是对 leftValue 变量的修改

下面是函数以及代码注释

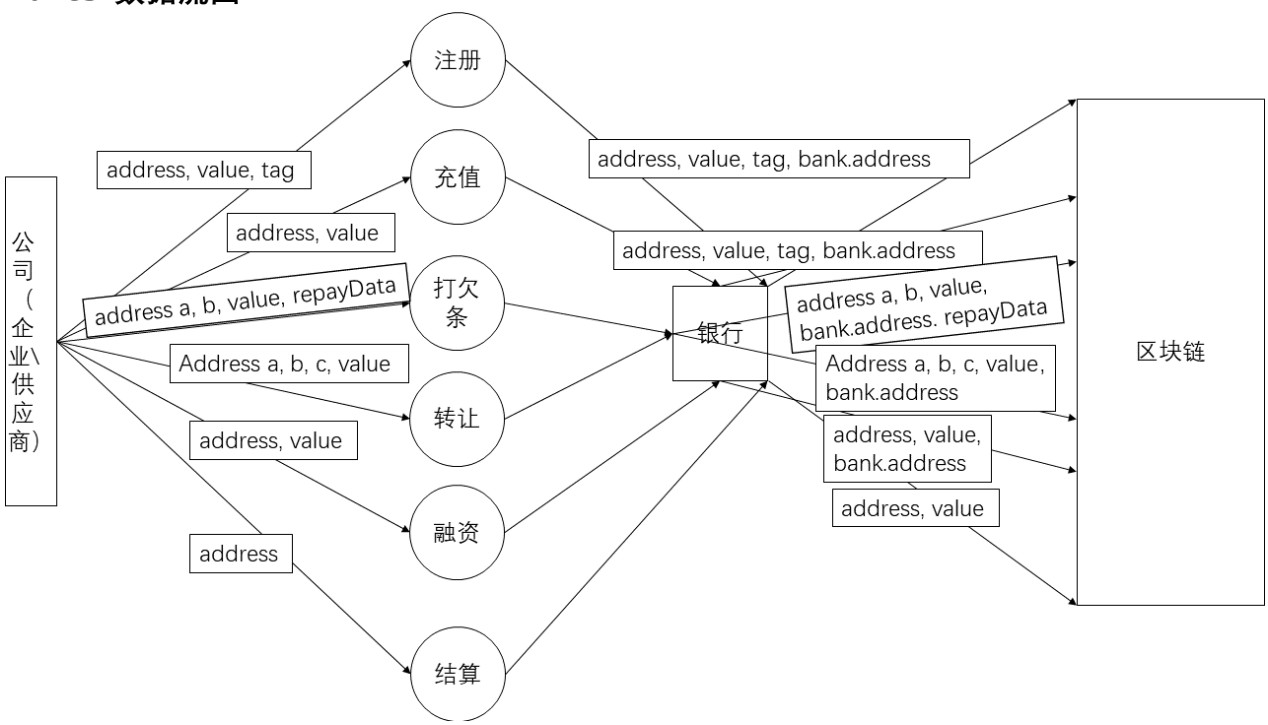
```
function Register (address a, uint tag, uint value) public returns(bool, string memory) {
    require (msg.sender == admin);
    for (uint i = 0; i < registerList.length; ++i) {
      if (a == registerList[i]) {
        return (false, "Has been registered");
      }
    }
    organizations[a] = Organization(Tag(tag), 0, value, false, address(0));
    registerList.push(a);
    return (true, "Registered Successfully");
```

```
  }
  modifier onlyBank () {
    for (uint i = 0; i < registerList.length; ++i) {
      if (msg.sender == registerList[i] && organizations[msg.sender].tag == Tag.Bank) {
        _;
      }
    }
  }

function Transaction (address a, address b, uint value, uint time_limit) public onlyBank
returns(bool, string memory) {
    // a, b is not bank && msg.sender has been registered and is bank
    if (organizations[a].tag == Tag.Bank || organizations[b].tag == Tag.Bank) {
      return (false, "Ower and Borrower can not be Bank");
    }
    // enterprise need guararntee
    if (organizations[a].tag == Tag.Enterprise && organizations[a].isGuaranteed ==
false) {
      organizations[a].isGuaranteed = true;
      organizations[a].guaranteeBank = msg.sender;
      organizations[a].balance += 1000;
    }
    //
    if (organizations[a].balance < value) {
      return (false, "balance < value, not enough credits to owe");
    }

    Bill memory bill = Bill(a, b, msg.sender, value, now+time_limit);
    // bill
    billList.push(bill);
    // balance
    organizations[a].balance -= value;
    organizations[b].balance += value;
    return (true, "Transaction Successfully");
  }

  // Suppose there can be only one bill at a time between a and b
  // msg.sender as b, a->b and b->c to a->c

function Transfer (address a, address c, uint value) public returns(bool, string memory)
{
    if (organizations[a].tag == Tag.Bank || organizations[c].tag == Tag.Bank ||
organizations[msg.sender].tag == Tag.Bank) {
      return (false, "Both parties can not be Bank");
    }
    uint xIndex; // in ab
    uint xValue;
    uint yIndex; // in bc
    uint yValue;
    for (uint i = 0; i < billList.length; ++i) {
      if (billList[i].owner == a && billList[i].borrower == msg.sender) {
        xIndex = i;
        xValue = billList[xIndex].value;
      }
      if (billList[i].owner == msg.sender && billList[i].borrower == c) {
        yIndex = i;
        yValue = billList[yIndex].value;
      }
```

```
    }

    // xValue >= yValue >= value
    if (value > yValue) {
      return (false, "value > yValue(b owe c)");
    }
    if (value > xValue) {
      return (false, "value > xValue(a owe b)");
    }
    // change 2 bill, add 1 biil ,every bill according to three organ
    // if value in bill = o, kill it
    // biil in ab
    billList[xIndex].value -= value;
    // bill in bc
    billList[yIndex].value -= value;
    // add bill in ac
    address bank = billList[xIndex].verifier;
    uint replayData = billList[yIndex].replayData;
    billList.push(Bill(a, c, bank, value, replayData));

    if (billList[xIndex].value == 0) {
      delete billList[xIndex];
    }
    if (billList[yIndex].value == 0) {
      delete billList[yIndex];
    }

    return (true, "Transfer Successfully");
  }

function Finaning (address a, uint value) public onlyBank returns(bool, string memory)
{
    if (organizations[a].tag != Tag.Supplier) {
      return (false, "Only Supplier can make finaning");
    }
    if (organizations[a].balance < value) {
      return (false, "balance <= value, not enough credits");
    }
    organizations[a].balance -= value;
    organizations[msg.sender].balance += value;
    organizations[a].finaning[msg.sender] += value;
    organizations[msg.sender].finaning[a] += value;
    return (true, "Finaning Successfully");
  }

  function Liquidation (address a, address b) public onlyBank returns(bool, string
memory) {
    if (organizations[a].tag == Tag.Bank || organizations[b].tag == Tag.Bank) {
      return (false, "Settlement of both sides can not be bank");
    }
    for (uint i = 0; i < billList.length; ++i) {
      if (billList[i].borrower == a && billList[i].owner == b && billList[i].verifier ==
msg.sender && billList[i].replayData <= now ) {
        if (organizations[b].leftValue < billList[i].value) {
          return (false, "The ower has not enough money to pay!");
        }
        organizations[b].leftValue -= billList[i].value;
        organizations[a].leftValue += billList[i].value;
```

```
        organizations[b].balance += billList[i].value;
        organizations[a].balance -= billList[i].value;
        delete billList[i];
        return (true, "Liquidation Successfully");
      }
    }
  }

function charge (address a, uint value) public returns(bool, string memory) {
    if (msg.sender != admin) {
      return (false, "Only admin can charge for organizations");
    }
    if (organizations[a].tag == Tag.Bank) {
      return (false, "Bank can not charge");
    }
    organizations[a].leftValue += value;
    return (true, "Charge Successfully");
  }
}
```

Part3 数据流图

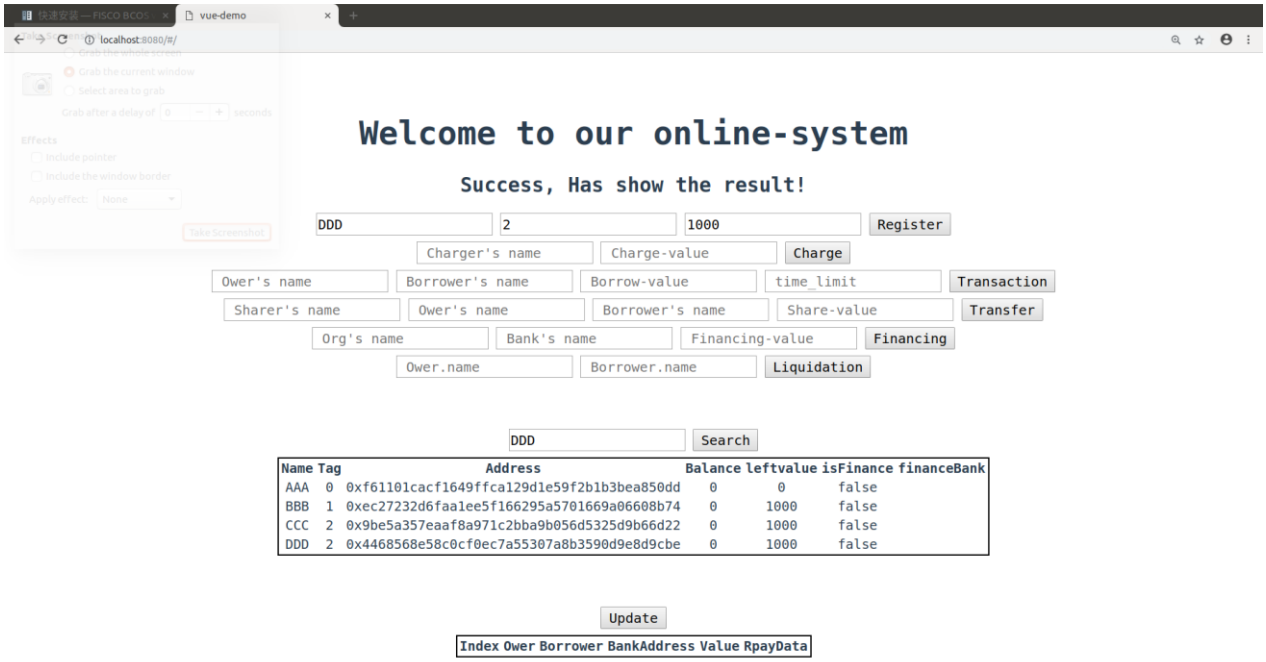


三、 功能测试

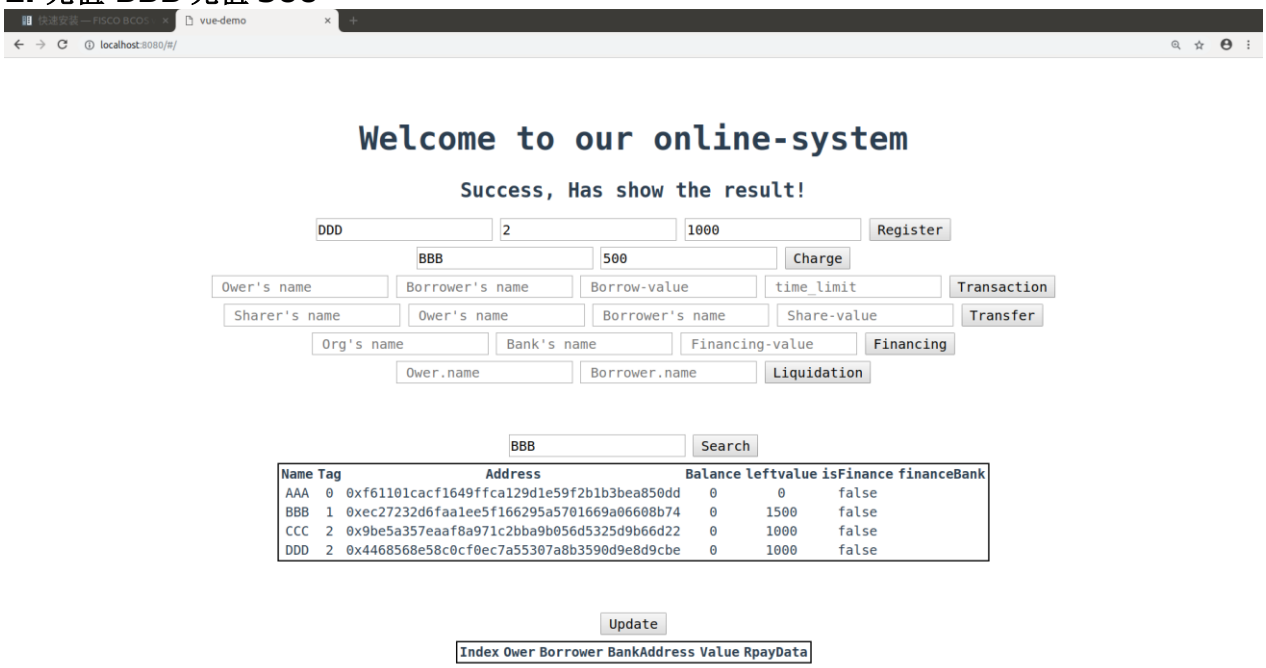
以下操作步骤包含所有功能

1. 注册 AAA 0 0(tag = 0 表示银行，1 表示企业，2 表示供应商)

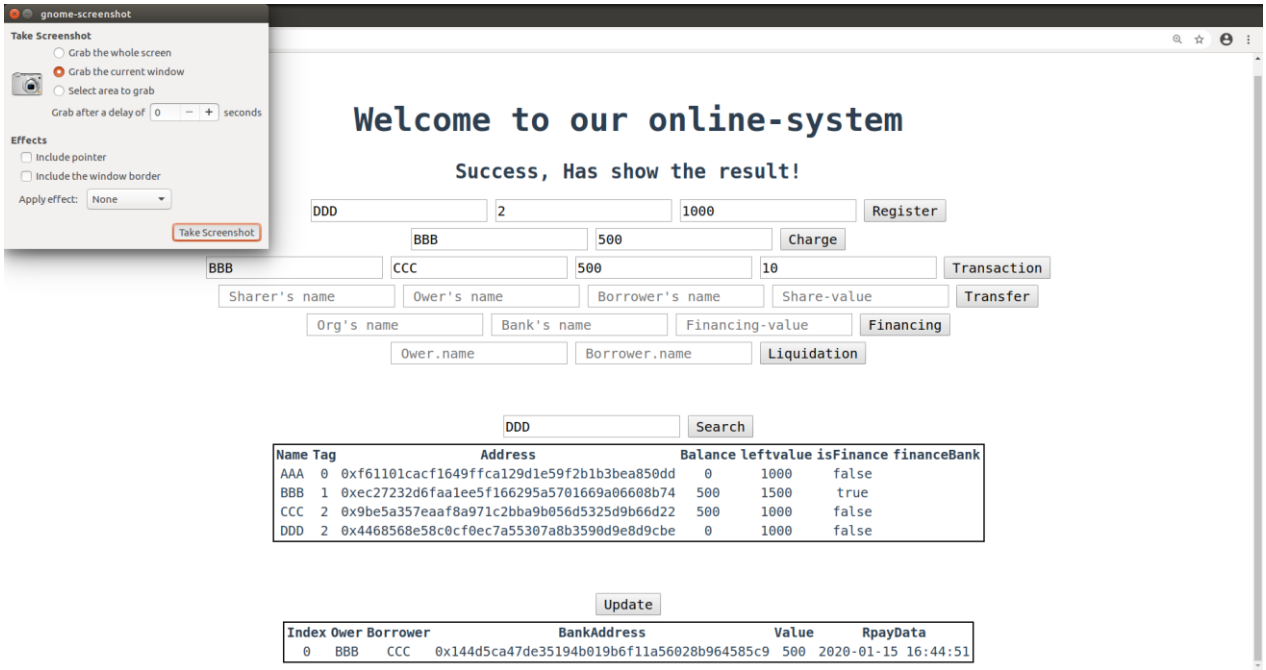
注册 BBB 1 1000 CCC 2 1000 DDD 2 1000(1000 代表初始资金，用于结算)



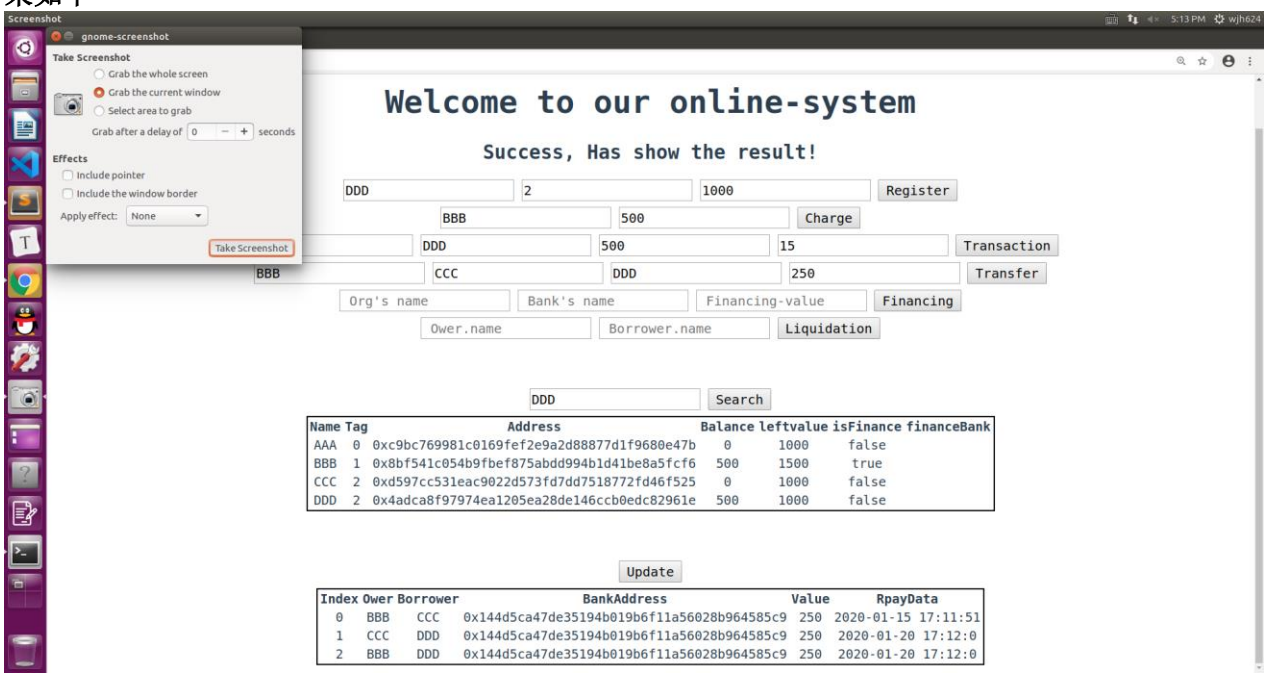
2. 充值 **BBB** 充值 **500**



3. 打欠条 **BBB** 向 **CCC** 打欠条，**500**，**10** 天（注意 **BBB** 的 **balance** 之所以为 **500**，因为企业 **tag** 为 **1**，在第一次打欠条的时候，银行会自动为其担保 **1000**，然后 **1000-500=500**），而 **CCC** 的 **balance** 因为 **BBB** 欠他钱，所以 **0+500=500**）

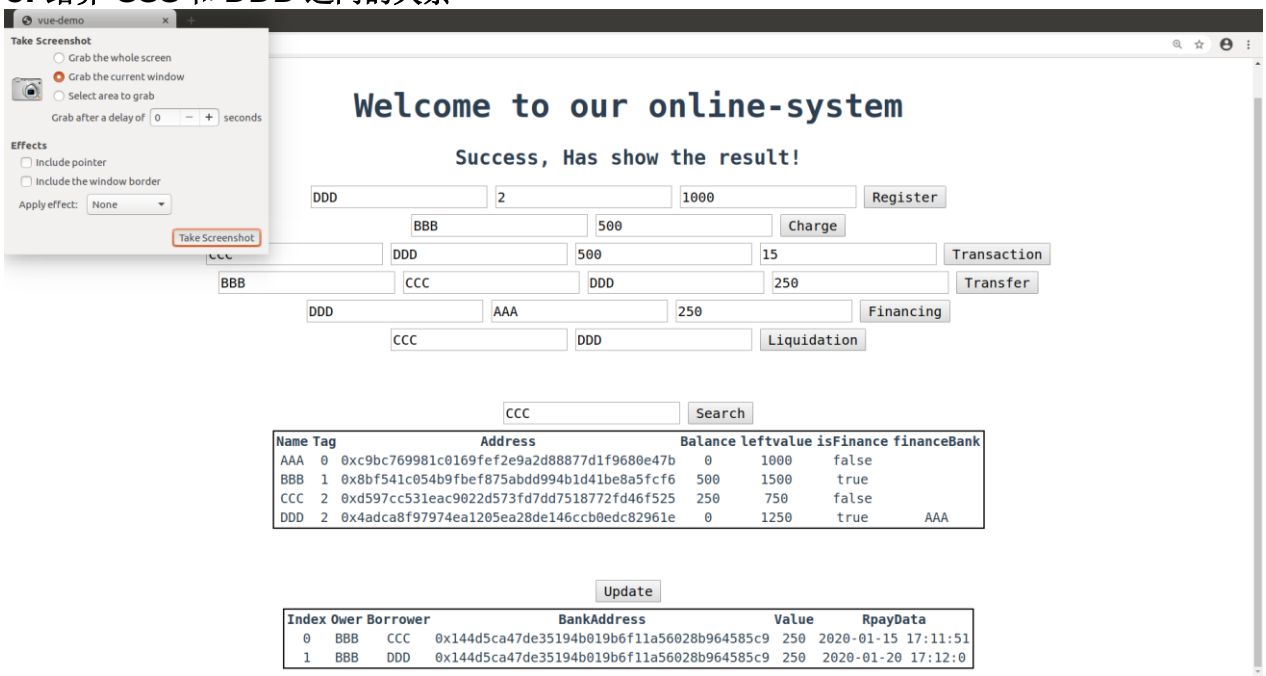


4. 分担欠条 **BBB->CCC(500) CCC->DDD(500)，BBB** 替 **CCC** 为 **DDD** 分担 **250**，结果如下

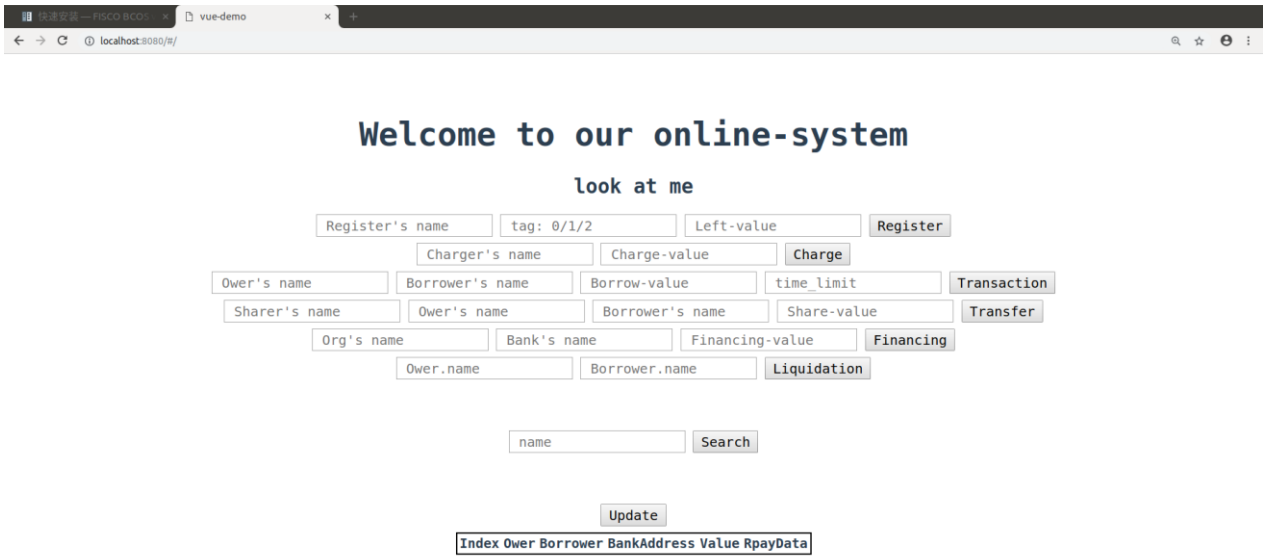


5. 融资 **DDD** 向 **AAA** 银行融资 **250**，**DDD** 的 **balance** 会减少 **250**

6. 结算 CCC 和 DDD 之间的欠条



四、 界面展示



五、 心得体会

前后端的连接太难了，开始完全不知道应该怎么办，真是一天一天生生磨出来进度。整个项目做完还是有些问题的，需要以后解决。

最为困难的是前端，后端和链端的连接，看官方教程，都是个说个的，感觉一个顶三个那种，而对于另一个方面的连接却都是很少提及。在网上看博客的例子，虽然有实例，但是过于简单而且没有详细的说明原因，我个菜鸟看的云里雾里，整个项目的推进进度真是一天一天磨出来的。

从确定 vue 前端和，后端是 fisco 官网中的 nodejs jdk，然后之后显示在虚拟机安装相应的安装包，之后显示配置，每一次 npm install 即使用的淘宝的源，都要花很长时间。然后就是学习 vue 和 express，如何在本地启动客户端服务器，并且和前端跨域相连。

最后调试了好一段时间才好。自己前后端都不太熟练，还需要不断练习。

作业要求：

1. 命名要求：学号_姓名，例如 16340000_林 XX。
2. 实验报告提交格式为 pdf。
3. 实验内容不允许抄袭，我们要进行代码相似度对比。如发现抄袭，按 0 分处理。