

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	2017 级	专业 (方向)	软件工程
学号	17343046	姓名	黄善恒
电话	13112942938	Email	952551920@qq.com
开始日期	2019.11.10	完成日期	2019.11.28

一、项目背景

区块链+供应链金融：

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。

实现功能：

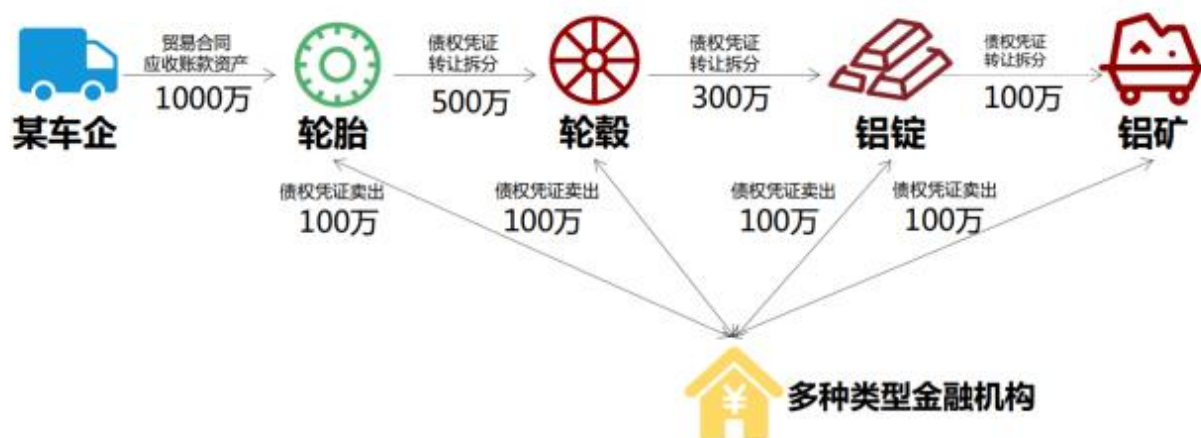
功能一：实现采购商品—签发应收账款 交易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。

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。

二、 方案设计



- 功能说明

- 根据供应链场景，并根据实际情况分析，智能合约需要完成下属 4 个功能
 1. 发起交易，其中可以利用现金直接完成交易或签发应付账款延期支付
 2. 应付账款的转移，可直接利用账款进行交易，可对账款进行拆分重组
 3. 应付账款的套现，向金融机构出售账款获取现金
 4. 应付账款的支付，在账款还款期限后，签发账款的公司需要支付相应的金额，可以用现金或其名下的应付账款支付。
- 根据实际情况，向智能合约添加下述功能以使得整个供应链运行更正常，能多次调试：
 1. 向供应链中增加公司，并需要根据实际情况判断判断该公司是否有签发应付账款的能力
 2. 发起货物交易（买/卖），将公司名下的货物套现增加公司现金流，以维持整个供应链的上下关系的稳定以及正常运转

- 功能实现

- 实现上述功能的前提是有第三方机构进行管理，此处建立了名为“金融机构”的用户并将智能合约部署到其上并记为“agencyAddr”保存用户地址。
- 根据功能的实现情况，建立两类结构体分别为“Company”记录公司信息以及“Credit”记录账款信息，其相应的代码如下

```
1. struct Company{ //在第三方处注册的公司才能进行有效的债权转让
2.     address companyAddr;
3.     bytes32 companyName;
4.     bool canTrust; // 由 Agency 判断公司是否可信
5.     bool isUsed;
6.     uint receiveableLength; //由于账款长度变化 以此记录长度 而非直接用.length
7.     uint cashAccount; //现金
8.     uint cargoAccount; // 货物资产
9. }
10. struct Credit{ //记录账款
11.     address to;
12.     address from1;
```

```

13.         uint amount;
14.         uint buildDate; //账款创建日期
15.         uint payDate; //日期限制 测试以秒为单位 便于快速观察
16.     }

```

- 其中“Company”结构体信息分别记录了公司对应的用户地址、公司名、公司是否可信、当前公司是否被创建（用于“mapping”的判断）、公司名下账款数目、公司现金数目、公司货物数目

- “Credit”结构体信息则分别记录了账款的归属者、账款的签发者、账款金额大小、账款签发日期、账款还款期限（约定多久后还款而非还款的日期）

- 由于“solidity”语言对二维数组并不兼容，因此设置了“mapping”来保存对应的信息，上述所说的“isUsed”变量即为验证当前地址是否对应了特定公司。其中建立了如下两个变量：

```

1. mapping(address => Credit[]) public accountsReceivable; // 应收账款
2. mapping(address => Company) public companys;

```

- 在必要的变量/结构体创建完成后，按照上述所说功能依次论述其实现原理和方案。

1. 发起交易 transaction

- 当处于供应链的公司向其他公司发起交易时，存在两种情况，一是直接使用现金交易（“status1 == true”），此时直接修改二者的“cashAccount”和“cargoAccount”变量值即可，为了体现真正的交易关系以及维持供应链能持续运转，此处设定“货物总以 2 倍价格成交”，即相应的实现如下：

```

1.     if(status1 == true){
2.         require(companys[fromAddr].cashAccount >= account && companys[to].cargoAccount >
           (account / 2));
3.         companys[fromAddr].cargoAccount += account; // 对于 from1 而言 货物的价格就是买来的
           价格 因此+account 而不/2
4.         companys[fromAddr].cashAccount -= account;
5.         companys[to].cargoAccount -= account / 2; //认为利润为 2 倍的货物价格 因此买 500w 的
           货物实际上只减少了 250w 的货物存量
6.         companys[to].cashAccount += account;
7.     }

```

- 二是签发应付账款延迟付款即（“status==false”）。此时需要验证签发账款的公司是否可信即判断“canTrust”值后创建相应的账款添加到对应公司名下的“accountsReceivable”内，对应的实现如下：

```

1.     else{
2.         require(companys[fromAddr].canTrust); //此时需要 from1 地址公司的 canTrust 为
           true 才能贷款
3.         require(companys[to].cargoAccount >= (account / 2));
4.         credit = Credit({
5.             to: to,
6.             from1: fromAddr,
7.             amount: account,
8.             buildDate: now,
9.             payDate: payTime
10.        });
11.         addCredit(to);

```

```

12.         companys[fromAddr].cargoAccount += account;
13.         companys[to].cargoAccount -= account / 2; //认为利润为 2 倍的货物价格 因此买
           500w 的货物实际上只减少了 250w 的货物存量
14.     }

```

2. 账款转移 transfer

- 账款转移函数就是将 B 名下的由 A 签发的账款部分或全部转移到 C 名下，此时由于归属权是固定的，并不需要处理转让特定的账款，只需要是 B 名下 A 签发的账款即可，因此遍历 B 名下账款，找到满足特定数额的账款，即下述代码：

```

1.     for(uint i = 0; i < companys[toOld].receiveableLength; i++){
2.         if(accountsReceivable[toOld][i].from1 == fromAddr){
3.             curAccount += accountsReceivable[toOld][i].amount;
4.             index.push(i); // 记录下转让方需要处理的特定债权下标
5.             if(curAccount >= account){
6.                 break;
7.             }
8.         }
9.     }

```

- 在满足条件即对应的账款额度不小于需要转让的账款数额下，对债权进行转移。存在两种情况

1. 找到的需要进行的账款总金额恰好等于需要转让的金额
2. 账款总金额大于需要转让的金额

- 在第一种情况下，直接对``accountsReceivable``进行处理即可。由于``solidity``数组不支持删除元素，因此利用了``receiveableLength``变量灵活处理，将需要删除的账款移除到数组末尾，而数组前``receiveableLength``才为有效账款。账款的转移包含调用了``addCredit``函数往 C 公司名下添加转款以及处理 B 公司账款减小``receiveableLength``值，对应代码如下：

```

1.     else{
2.         for(i = index.length - 1; i >= 0; i--){ //从后往前避免出错
3.             credit = accountsReceivable[toOld][index[i]];
4.             addCredit(toNew);
5.             accountsReceivable[toOld][index[i]] = accountsReceivable[toOld][tmpLength
           h + i - index.length];
6.             if(i == 0 || i >= index.length){
7.                 break;
8.             }
9.         }
10.        companys[toOld].receiveableLength -= index.length;
11.    }

```

- 其中``addCredit``函数内还处理了账款的归属权到 C 处，代码如下：

```

1.     function addCredit(address addr) internal{ //增加债权 反正都是债主 没必要检查地址
2.         credit.to = addr; //处理方便一些
3.         if(companys[addr].receiveableLength < accountsReceivable[addr].length){
4.             accountsReceivable[addr][companys[addr].receiveableLength] = credit;
5.         }else{
6.             accountsReceivable[addr].push(credit);
7.         }
8.         companys[addr].receiveableLength += 1;
9.     }

```

- 在第二种情况下，并不能直接转让账款，需要处理 B 的账款信息，减少数额的同时往 C 添加账款，对应的代码如下：

```
1.     if(curAccount > account){//部分债权转移需要考虑怎么转 避免覆盖
2.         Credit creTmp = accountsReceivable[toOld][index[0]];
3.         accountsReceivable[toOld][index[0]] = accountsReceivable[toOld][index[index.length - 1]];
4.         accountsReceivable[toOld][index[index.length - 1]] = creTmp; //调转
5.         for(i = index.length - 1; i >= 1; i--){ //后 length - 1 个是整个债券转让 第 1 个是部分债权转让
6.             credit = accountsReceivable[toOld][index[i]];
7.             addCredit(toNew);
8.             accountsReceivable[toOld][index[i]] = accountsReceivable[toOld][tmpLength + i - index.length];
9.             if(i == 0 || i >= index.length){
10.                 break;
11.             }
12.         }
13.         companys[toOld].receiveableLength = companys[toOld].receiveableLength - index.length + 1;
14.         credit = accountsReceivable[toOld][index[0]];
15.         credit.amount = credit.amount - (curAccount - account); // 总的-剩余的=转移的
16.         accountsReceivable[toOld][index[0]].amount = curAccount - account; //剩余的债权 curAccount - account
17.         addCredit(toNew);
18.     }
```

3. 账款套现 saleCredit

- 在供应链过程中，可能发生公司急需现金周转，此时允许公司将归属的账款转让给金融机构依次直接获得现金。账款的套现实际上也涉及到账款的转移，但对象是“金融机构”，因此需要额外的为“金融机构”创建公司归属并赋予足够大的资金以供程序的正常运行。因此首先在“constuctor”构造函数中需要进行相应的处理，相应的代码如下：

```
1.     constructor(){
2.         agencyAddr = msg.sender;
3.         companys[agencyAddr] = Company({
4.             companyAddr: agencyAddr,
5.             companyName: bytes32("Agency"),
6.             canTrust: true,
7.             isUsed: true,
8.             receiveableLength: 0,
9.             cashAccount: 1000000, //足够大
10.            cargoAccount:1000000
11.        });
12.    }
```

- 套现的过程只需要调贷款转移的函数“transfer”，并传入“agencyAddr”作为参数即可，随后增加现金即完成相应的套现操作。

```
1.     if(surplus == account){
2.         for(uint i = 0; i < index.length; i++){
3.             transfer(checkAddr, agencyAddr, accountsReceivable[checkAddr][j].from1, accountsReceivable[checkAddr][j].amount);
4.         }
5.     }else{
6.         for(i = 0; i < index.length - 1; i++){
7.             transfer(checkAddr, agencyAddr, accountsReceivable[checkAddr][j].from1, accountsReceivable[checkAddr][j].amount);
8.         }
9.     }
```

```

8.         }
9.         transfer(checkAddr, agencyAddr, accountsReivable[checkAddr][index.length -
1].from1, accountsReivable[checkAddr][index.length - 1].amount - (surplus -
account));
10.    }
11.    companys[checkAddr].cargoAccount += account;

```

4. 支付账款 repayment

- 账款的签定时通过``now``获取当前时间保存到``builddate``变量中，其中``payDate``规定了还款的时间跨度信息，此处以秒为单位即``payDate * 1``便于观察效果。

- 函数``repayment``通过检查当前传入的地址对应的公司名下所有贷款信息，当发现有到期账款时则向签发贷款的公司索取支付。此时，存在 2 种情况

1. 签发贷款公司现金足以支付
2. 签发贷款公司现金不足以支付，不足部分以其名下应付账款支付

- 在第一种情况下直接扣除特定公司现金即可，显影的代码如下：

```

1.  if(companys[credit.from1].cashAccount >= credit.amount){
2.      companys[credit.from1].cashAccount -= credit.amount; //用现金还款
3.      companys[credit.to].cashAccount += credit.amount;
4.  }

```

- 在第二种情况下，需要遍历应付账款直至补齐差额后，扣除该公司所有现金的同时将对应的账款转移给收取者对应的公司名下。此时依然需要注意的是转移整个转款还是部分转移，与``transfer``操作类似。

```

1.  for(uint j = 0; j < companys[credit.from1].receiveableLength; j++){ //credit.from1 是还
    款人
2.      surplus += accountsReivable[credit.from1][j].amount; //记录还款人需要进行其名下的债
    权转让的债款下标
3.      index.push(j);
4.      if(surplus >= credit.amount){
5.          break;
6.      }
7.  }
8.  require(surplus >= credit.amount); //还款失败
9.  companys[credit.from1].cashAccount = 0;
10. if(surplus > credit.amount){
11.     for(j = 0; j < index.length - 1; j++){ //不区分 fromAddr 来转让债权
12.         transfer(credit.from1, checkAddr, accountsReivable[credit.from1][j].from1, ac
            countsReivable[credit.from1][j].amount); //全部转让
13.     }
14.     transfer(credit.from1, checkAddr, accountsReivable[credit.from1][index.length -
1].from1, accountsReivable[credit.from1][index.length - 1].amount - (surplus -
credit.amount)); //转让部分
15. }else{
16.     for(j = 0; j < index.length; j++){ //不区分 fromAddr 来转让债权
17.         transfer(credit.from1, checkAddr, accountsReivable[credit.from1][j].from1, ac
            countsReivable[credit.from1][j].amount); //全部转让
18.     }
19. }

```

5. 增加公司 addCompany

- 向供应链中添加节点公司需要进行的操作，其中必须由``金融机构``发起调用。其中，需要传入一定的参数，此时默认当``cashAccount+cargoAccount``值大于``1000``时，认为

该公司可信，允许签发账款，对应的代码如下：

```
1.      bool trust = false;
2.      if(cash + cargo >= 1000){
3.          trust = true; //当公司注册资产大于 1000 时，认为该公司可信
4.      }
5.      companys[addr] = Company({
6.          companyAddr: addr,
7.          companyName: name,
8.          canTrust: trust,
9.          isUsed: true,
10.         receiveableLength: 0,
11.         cashAccount: cash,
12.         cargoAccount: cargo
13.     });
```

6. 货物买卖 saleCargo/stoc*

- 在供应链运行的过程中，如果单一的签发账款，上流资金会持续往下流流动，如果没有额外的增加现金的操作会令供应链不能正常运行，因此增加货物买卖函数使得供应链能维持自身稳定
- 其中，进货操作``stock``为减少现金增加所有的货物金额，此时是``1:1``的模型
- 出货操作``saleCargo``为减少货物金额增加现金，货物的售卖是有利润的，初始设定为 2，因此二者比例是``1:2``。
- 代码较为简单，因此此处不贴上

相应的，此处采用了 **Node.js** 方式编写后端的调用，按照教程下载相应的 **SDK** 后编写下述代码即可开始后端调用程序。

```
1.  var express      = require('express');
2.  var app           = express();
3.  var bodyParser    = require('body-parser')
4.  var cli = require('./cli')
5.  var fs = require("fs")
6.  app.use(bodyParser.urlencoded({extended:false}));
7.
8.  const path = require('path');
9.  const utils = require('./api/common/utils');
10.
11. const { Web3jService, ConsensusService, SystemConfigService } = require('./api');
12. const { ContractsDir, ContractsOutputDir } = require('./api/web3j/constant');
13.
14. const web3 = new Web3jService();
15. const consensusService = new ConsensusService();
16. const systemConfigService = new SystemConfigService();
17.
18. var contractAddr;
```

以调用交易函数 transaction 为例，其对应的简单调用代码为：

```
1.  app.get('/register',function(req,res){
```



```

2.     fs.readFile("./resources/register.html", function(err, data){
3.         if(err){
4.             return console.log(err);
5.         }else{
6.             res.send(data.toString());
7.         }
8.     })
9.     console.log('register sendding')
10. })
11.
12. app.post('/transaction',function(req,res){
13.     var body = req.body;
14.     var state = false;
15.     if(body.status){
16.         state = true;
17.     }
18.     cli.main('call Agency ' + contractAddr + 'addCompany ' + body.to + ' ' + body.from + ' ' + state + ' ' + body.payTime);
19.     console.log('交易');
20.     res.send(eval(body));
21. })
22. console.info('listen 3000')
23. var server=app.listen(3000) ;

```

三、 功能测试

以铝锭公司为例调用“addCompany”函数，传入特定参数，如下图，其中用户必须设定为“金融机构”，当参数输入正确时会显示“交易成功”，否则显示“交易失败”，其调用过程以及交易内容分别如下：

发送交易

×

合约名

称: test_9

合约地

址: 0x5216d38b015c438888e0d862 ⓘ

用户: 金融机构 ▾

方法: function ▾ addCompan ▾

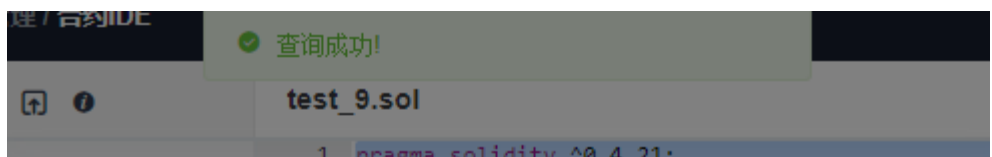
参数:

addr	0xc50d35f9181689d
name	D
cash	200
cargo	200

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：array1,array2。string等其他类型也不用加上引号。

取消 确定

- 通过``companys``变量输入对应的地址查看新加入的铝锭公司的对应信息，输出如下，表示建立成功



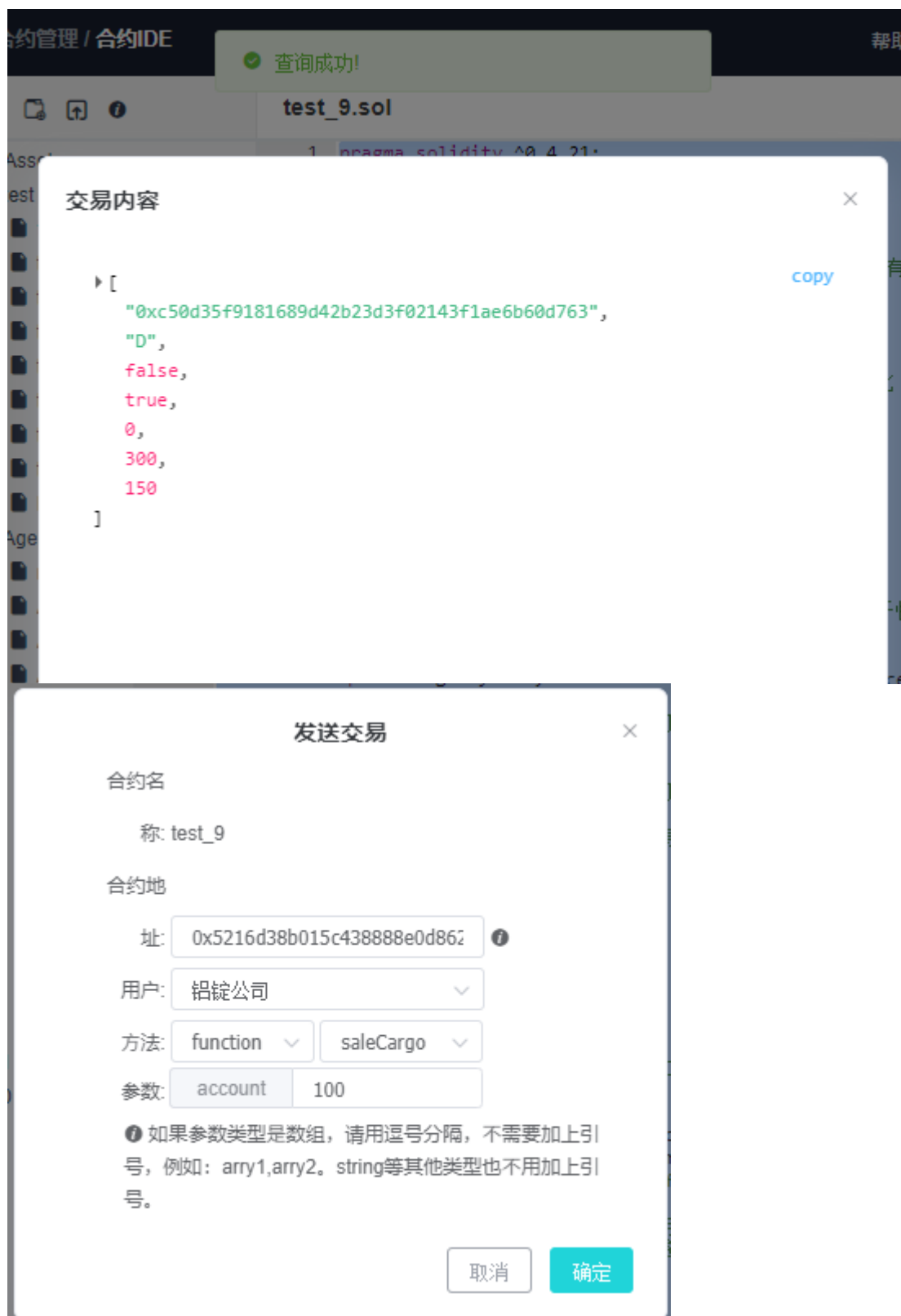
交易内容

×

```
[  
  "0xc50d35f9181689d42b23d3f02143f1ae6b60d763",  
  "D",  
  false,  
  true,  
  0,  
  200,  
  200  
]
```

copy

- 此时对该公司调用货物买卖函数即“saleCargo”，输入参数“100”表示出售数额 200 的货物，其调用过程与调用完成后铝锭公司对应的信息分别如下，可以发现货物金额减少 50 的同时现金金额增加 100，与预期相同



- 在运行过程前已有“轮胎公司”，其对应的“company”信息如下，可发现该公司可信因此利用其发布贷款检测账款函数

交易内容

```
[  
  "0x26f2f8a890665cedef15c58a47ac352a0411af65",  
  "B",  
  true,  
  true,  
  1,  
  0,  
  950  
]
```

copy

- 从上图发现“轮胎公司”此时现金为 0 的同时存在一个应付账款待收取，通过“accountReceivable”查看对应账款信息，即下图，可发现该账款来自车企且金额为“200”



The screenshot shows a window titled "交易内容" (Transaction Content) with a close button (X) in the top right corner. The window contains a JSON array of transaction data. The data is as follows:

```
[  
  "0x26f2f8a890665cedef15c58a47ac352a0411af65",  
  "0x451ca263d2599ff53a835d423df25a2b6495003b",  
  200,  
  1573694419815,  
  1  
]
```

There is a "copy" button in the top right corner of the window. The background shows a code editor with a file named "test_9.sol" and some Solidity code.

- 此时，“轮胎公司”调用“transaction”函数令轮胎公司向铝锭公司通过签发应付账款的方式发起交易，金额为“100”，调用过程如下图。

发送交易

×

合约名

称: test_9

合约地

址: ⓘ

用户:

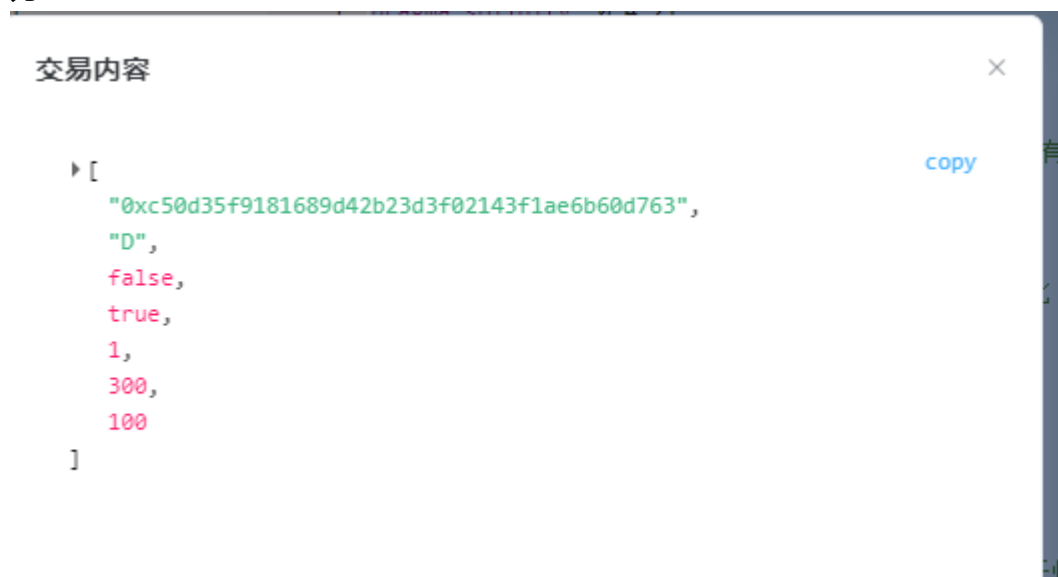
方法:

参数:

to	0xc50d35f9181689d42l
fromAddr	0x26f2f8a8906f
account	100
status1	false
payTime	<input type="text" value="1"/>

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如: arr1,arr2。string等其他类型也不用加上引号。

- 显示交易成功后查看铝锭公司的公司信息，输出如下，可看出名下的应付账款长度从“0”变为“1”



- 由于设定的“paydate”为 1，即 1 秒后账款到期，以“铝锭公司”地址调用“repayment”函数表示检测其公司名下的账款并收取费用，其调用过程和调用后查看该公司名下的合约信息分别如下两图，可以发现由于“轮胎公司”的现金不足以支付以此转移了来自“车企”的应付

账款到“铝锭公司”名下，由于该金额小于整个应付账款，因此只发生了部分转移即拆分的情况

交易内容

```
[
  "0xc50d35f9181689d42b23d3f02143f1ae6b60d763",
  "0x451ca263d2599ff53a835d423df25a2b6495003b",
  100,
  1573694419815,
  1
]
```

copy

4

address public agencyAddr;

发送交易

合约名

称: test_9

合约地

址: 0x5216d38b015c438888e0d862

用户:

金融机构

方法:

function

repayment

参数:

checkAddr

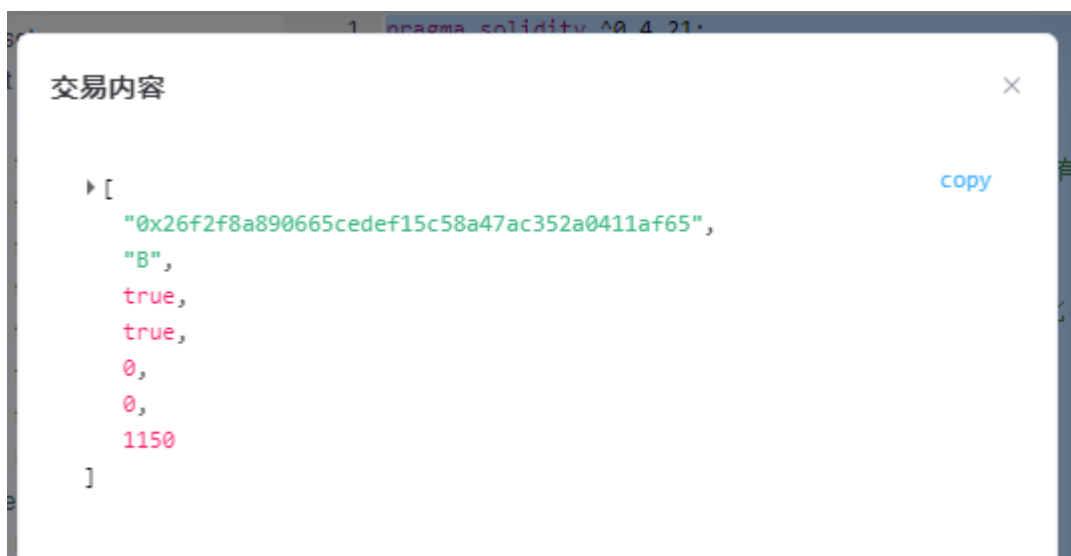
0xc50d35f918

如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：array1,array2。string等其他类型也不用加上引号。

取消

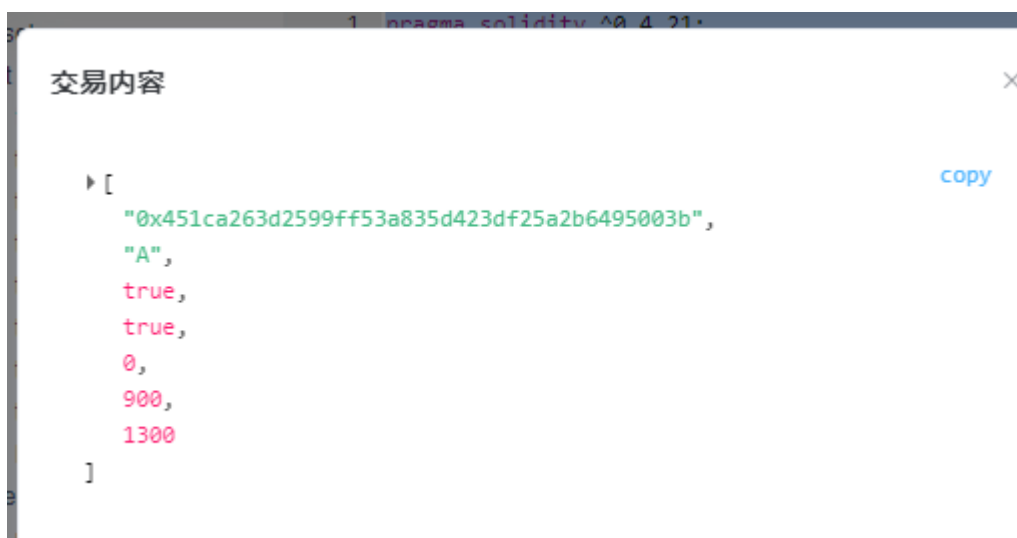
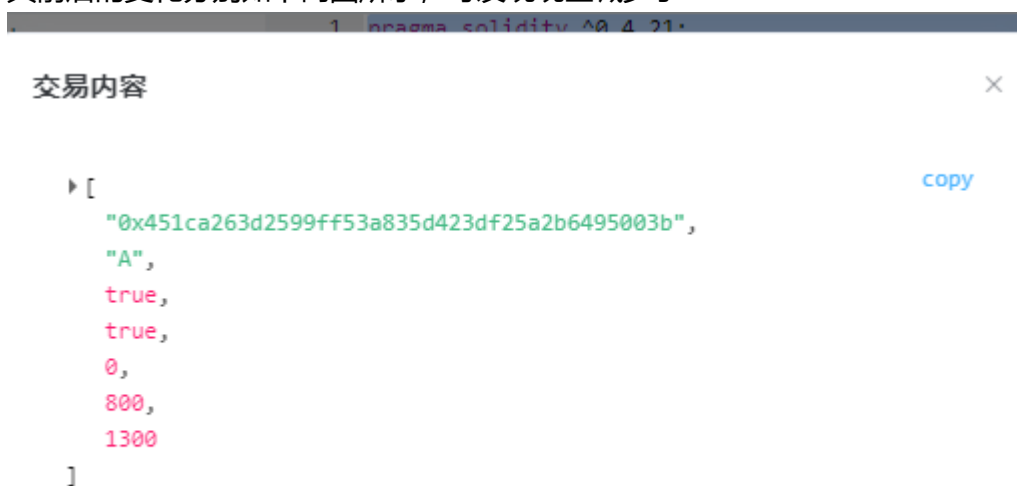
确定

- 重复上述步骤，再次发起金额为“100”的交易，此时查看“轮胎公司”的信息，由于名下的应付账款金额恰好为“100”，因此发生整个账款的转移，账款长度从“1”变为“0”



- 再次令

铝锭公司调用`repayment`函数，此时从`车企`处结算现金，此时由于车企现金足够，其前后的变化分别如下两图所示，可发现现金减少了`100`



- 铝锭公司调用函数`stock`购入货物，金额设定为`100`，前后变化如下两图所示，可发现货物金额增加`100`

交易内容

```
[
  "0xc50d35f9181689d42b23d3f02143f1ae6b60d763",
  "D",
  false,
  true,
  0,
  500,
  50
]
```

copy

交易内容

```
[
  "0xc50d35f9181689d42b23d3f02143f1ae6b60d763",
  "D",
  false,
  true,
  0,
  400,
  150
]
```

copy

- 再次调用``saleCargo``出售货物，金额设定为 100，结果如下图所示，可发现货物金额减少``50``而现金金额增加``100``

交易内容

```
[
  "0xc50d35f9181689d42b23d3f02143f1ae6b60d763",
  "D",
  false,
  true,
  0,
  500,
  100
]
```

copy

四、 界面展示

在初步调用前，输入 3000/agency 即可部署合约，命令台输出信息即表示部署成功，其后通过参数获取合约地址并保存完成其后操作，以其中两个功能为例展示界面。

```
ac LayerHandler [as handler] Request { /home/fisco-bcos-virtualbox-2/nodejs-sdk/packages/cli$ node ./agency.js  
{"contractAddress":"0x3bd38d86e26b073b4c30942f5906040c22031ee0","status":"0x0"}
```

输入网址：3000/register 进入公司注册界面

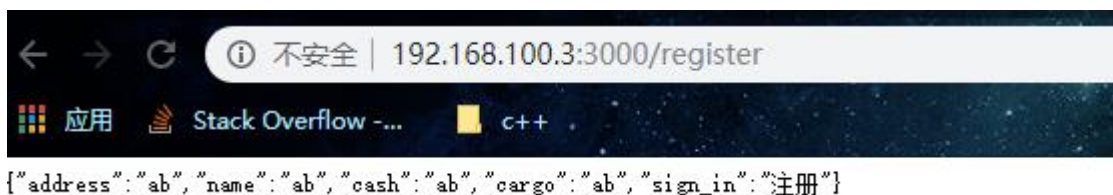
```
fisco-bcos@fisco-bcos-VirtualBox:~/nodejs-sdk/packages/cli$ node ./agency.js  
listen 3000
```

根据提示分别输入注册公司所需的参数信息即可注册成功



地址: 公司名: 注册资金: 货物金额:

其后会返回 json 格式的信息



```
{ "address": "ab", "name": "ab", "cash": "ab", "cargo": "ab", "sign_in": "注册" }
```

类似的，输入 网址：3000/transaction 进入交易界面



卖方: 买方: 金额: 方式: 期限:

输入相应信息即可完成交易。

由于前端调用部分未能完全理解，未能在网址上显示公司状态的变化，因此只能在 fisco-bcos 上查看。

五、 心得体会

之所以选了区块链技术这门课，根本是源于对区块链技术的兴趣。课上了解大致的原理后，在 FISCO 平台上基于教程实现基本的区块链的实现，对所学内容有了更深层次的了解，但实践之后发现对区块链的兴趣并没有对理论本身兴趣浓厚，以至于完成本次作业并未有足够的耐心去完成。在完成基本功能时，唯一感兴趣的是代码实现前对整个供应链的构建以及思考如何贴合实际，以至于可以在后续的实现中完成整个供应链的平衡和自动运转。在入手编写代码时，发现对 Sol 语言的掌握不足以至于部分功能未能实现，主要表现在部分变量以及语句的实现过程中发生错误而未能检查出错误原因，其中一个错误调试一整天后才发现是数组在函数内声明会出现错误，于是耐心消失殆尽。完成了基本的功能后只补充了部分功能以使供应链能在人为干涉下运行而其他功能未能实现，具体效果参见上述内容。观察教程后发现教程所述并不清晰，在勉强使用 node SDK 完成简易的后端操作后，前端操作的部署表示心有余而力不足，因此只能简单的编写简易 html 来代替。总的来说，区块链的理论很有意思，对其技术背后形成的原因以及寻找、修补漏洞思考颇多，但真正入手后觉得对这上面的兴趣并无想象的那么浓厚，更多的是对这个技术是为什么形成的、机制有什么问题的兴趣而非这个技术是如何实现的。