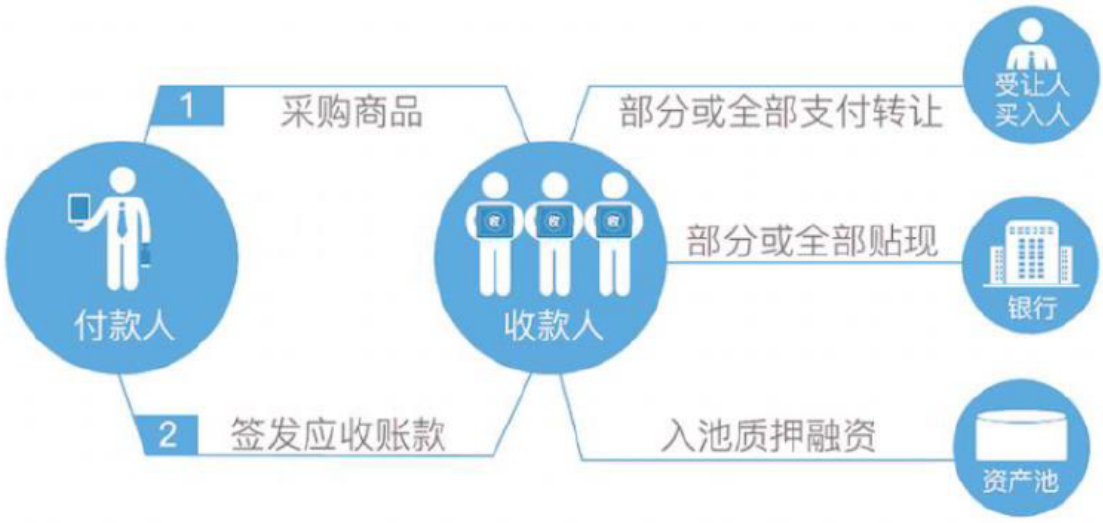
中山大学数据科学与计算机学院本科生实验报告

(2019年秋季学期)

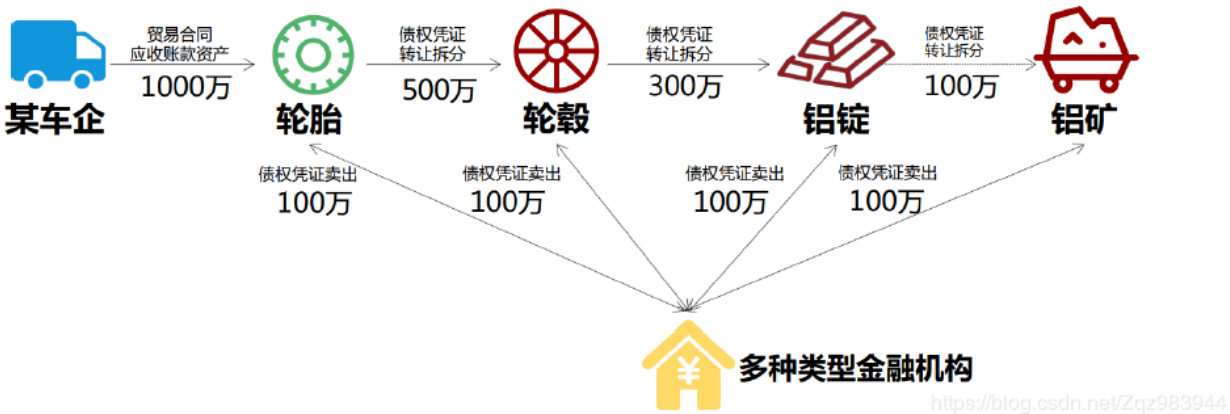
课程名称	区块链原理与技术	任课老师	郑子彬
年级	17级	专业（方向）	软件工程
学号	17343154	姓名	张嵌竹
电话	18022496500	Email	1093317614@qq.com
开始日期	2019.12.7	完成日期	2019.12.13

一、项目背景

基于已有的开源区块链系统 FISCO-BCOS（https://github.com/FISCO-BCOS/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。
区块链+供应链金融：



场景介绍：



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

存储设计

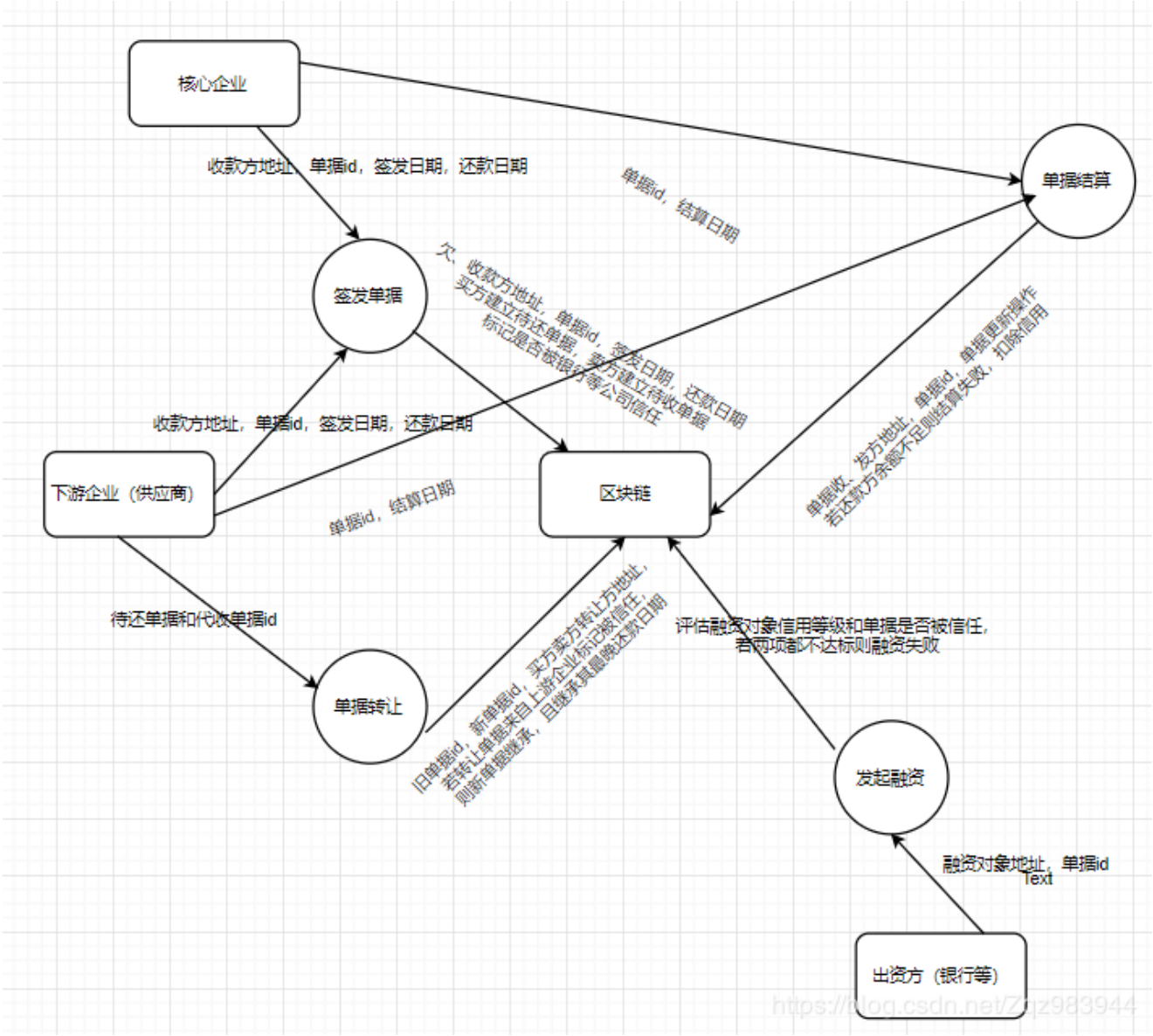
去中心化存储，合约中的公司和收据信息全部存储在区块链上，用户通过前端功能进行操作。
数据结构

```
int[3] credits=[999,100,50];
struct company{
    uint ID; //公司id
    string name;//公司名字
    uint attr;//公司性质(0:银行；1：核心企业；2：供应商)
    uint balance;
    int credit;//公司信用
    mapping(uint => receipt) sellReceipts;//出售的单据
    mapping(uint => receipt) purchaseReceipts;//购买的单据
}
mapping(address => company) public companies;//每个公司对应地址
```

```
mapping(string => address) NamesOfCompanies;//地址和名字的映射
company[] comArry;//避免有相同id的公司注册

struct receipt {
    uint ID;//单据ID
        string from ;// 欠款人
        string to ;// 收款人
        uint amount; // 金额
    string signTime;//签订时间
    string expiredTime;//还款到期时间
        bool status; // 状态，例如银行参与确认了这笔交易，则为true，可信度更高
    //如果查到其信誉值达到标准或者是手里有值得信任的收据则银行可以与其打款
}
```

- credits：公司注册信用，用于评估银行融资，若为核心企业则注册信用为100，不需要被信任的应收单据也可以被融资，若为下游企业注册信用为50，若其申请融资的应收单据不被信任则不得融资，公司信用在还款时会产生变化
- sellReceipt：公司应收账款单据
- purchaReceipt：公司应付账款单据
- status：单据状态，创建时标记是否为银行参与信任，融资时可以作为凭证

数据流图：



核心功能

公司注册

```
function giveRightToCom(address companyAddr,uint companyid,string memory com

    company storage temp = companies[companyAddr];
    temp.ID=companyid;
    temp.name=companyName;
    temp.attr=comattr;//公司的性质是在创建它的时候给的
    temp.balance=ba;
    temp.credit=credits[comattr];
    companies[companyAddr]=temp;
    NamesOfCompanies[companyName]=companyAddr;

    return (true);
}
```

传入参数有注册公司地址，公司id，公司名称，公司性质，公司注册资金

签发单据

```
function purchase(uint recID,address supplier,uint mount,bool stat,string me
    if(companies[supplier].ID == 0){
        return (0);
    }
    company storage f = companies[msg.sender];
    company storage t = companies[supplier];
    receipt memory r = receipt({
        ID: recID,
        from: f.name,
        to: t.name,
        amount: mount,
        signTime: _signTime,
        expiredTime: _expiredTime,
        status: stat//交易有没有被作证是在交易被创建的时候确定的
    });

    t.sellReceipts[recID]=r;
    f.purchaseReceipts[recID]=r;
    return (f.purchaseReceipts[recID].ID);
}
```

传入参数有单据id，收款人地址，交易金额，交易状态，签发时间和应还款时间，建立新单据receipt，并将其加入买方和卖方对应的数据结构里

单据转让

```
function receiptTrans(uint mount,uint oldSellID,uint oldPurID,uint newID,str


    address former= NamesOfCompanies[companies[msg.sender].sellReceipts[oldS
    address latter= NamesOfCompanies[companies[msg.sender].purchaseReceipts[



    if(companies[msg.sender].sellReceipts[oldSellID].ID == 0x0){
        return (false,"Sell Receipt with company doesn't exsit!",0x0);
    }
    if(companies[msg.sender].purchaseReceipts[oldPurID].ID == 0x0){
        return (false,"Purchase Receipt with company  doesn't exsit!",0x0);
    }
    if(companies[msg.sender].sellReceipts[oldSellID].amount<mount){
        return (false,"Trade Receipt less than required!",0x0);
    }
```

```
        if(companies[former].purchaseReceipts[newID].ID!=0x0 || companies[latter
            return(false,"New Receipt ID already exsits!",0x0);
        }
        companies[former].purchaseReceipts[oldSellID].amount-=mount;
        companies[msg.sender].sellReceipts[oldSellID].amount-=mount;
        if(companies[msg.sender].sellReceipts[oldSellID].amount==0){
            delete companies[msg.sender].sellReceipts[oldSellID];
            delete companies[former].purchaseReceipts[oldSellID];
        }
        companies[latter].sellReceipts[oldPurID].amount-=mount;
        companies[msg.sender].purchaseReceipts[oldPurID].amount-=mount;
        if(companies[msg.sender].purchaseReceipts[oldPurID].amount==0){
            delete companies[msg.sender].purchaseReceipts[oldPurID];
            delete companies[latter].sellReceipts[oldPurID];
        }

        receipt memory r = receipt({
            ID: newID,
            from: companies[former].name,
            to: companies[latter].name,
            amount: mount,
            signTime: transTime,
            expiredTime:  companies[latter].sellReceipts[oldPurID].expiredTime,/
            status: companies[former].purchaseReceipts[oldSellID].status//交易有
        });
        companies[latter].sellReceipts[newID]=r;
        companies[former].purchaseReceipts[newID]=r;
        return (true,"Transfer Receipt succussfully!",newID);

    }
```

传入参数有转发金额，应收单据id，应付单据id，新单据id，转让时间
应收单据，应付单据对应交易双方单据金额减去转发金额，若剩余金额为0则销毁单据，在交易上游企业和下游企业之间建立新的转发单据，继承原单据的受信任状态和应还款时间
企业还款

```
    function settle(uint sReceipt,string memory timeN) public returns(bool,strir
        company storage thisCom=companies[msg.sender];
        company storage creditor=companies[NamesOfCompanies[companies[msg.sender
        if(thisCom.balance<thisCom.purchaseReceipts[sReceipt].amount){
            thisCom.credit-=10;
            return (false,"company's balance is not enough to settle this debt."
        }
        thisCom.balance-=thisCom.purchaseReceipts[sReceipt].amount;
```

```
        creditor.balance+=thisCom.purchaseReceipts[sReceipt].amount;
        delete thisCom.purchaseReceipts[sReceipt];
        delete creditor.sellReceipts[sReceipt];
        if(keccak256(abi.encodePacked(companies[msg.sender].purchaseReceipts[sRe
            thisCom.credit-=10;
            return (true,"company settle debt out of expired time");
        }
        thisCom.credit+=10;
        return (true,"company settle debt succussfully!");
    }
```

传入参数为还款单据id和还款日期
企业发起还款时检查企业余额，若少于单据余额则还款失败，还款成功时还应检查还款日期是否迟于单据注明的应还日期，若逾期还款则扣除企业评估信用。
银行融资

```
    function financing(address receiver,uint fReceipt) public returns(int){
        if(companies[msg.sender].attr!=0){
          //  return (false,"Only banks can give balances to companies.");
          return (1);
        }
        if(companies[receiver].credit<60&&companies[receiver].sellReceipts[fRece
           // return (false,"Only finance to trusted companies");
           return (2);
        }

        companies[receiver].balance+=companies[receiver].sellReceipts[fReceipt].
        //债权转让
        company storage debtor = companies[NamesOfCompanies[companies[receiver].
        debtor.purchaseReceipts[fReceipt].to = companies[msg.sender].name;
        delete companies[receiver].sellReceipts[fReceipt];
        //return (true,"Finance succussfully!");
        return (3);
    }
```

传入参数有融资企业地址，收据凭证
检查融资企业信用积分和收据凭证是否可信，若企业信用积分低于最低融资积分且收据凭证不可信则不得融资，融资成功后银行向企业注入单据待收资金，且原应付款企业的债权方转变为银行
余额查询

```
    function getBalance() public returns(uint){
        return (companies[msg.sender].balance);
```

```
    }
```

返回交易发起者的企业余额

合约部署

```
<script src="./web3.js"></script>
<script>
    let Web3 = require('web3');
let web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));
//导入合约的ABI文件
let abi = JSON.parse('[{"constant":false,"inputs":[{"name":"receiver","type":"ac
let SContract = web3.eth.contract(abi);

let contractInstance = SContract.at('0xF1fEE55b2451E076533af55f363772a3624786eE'
```

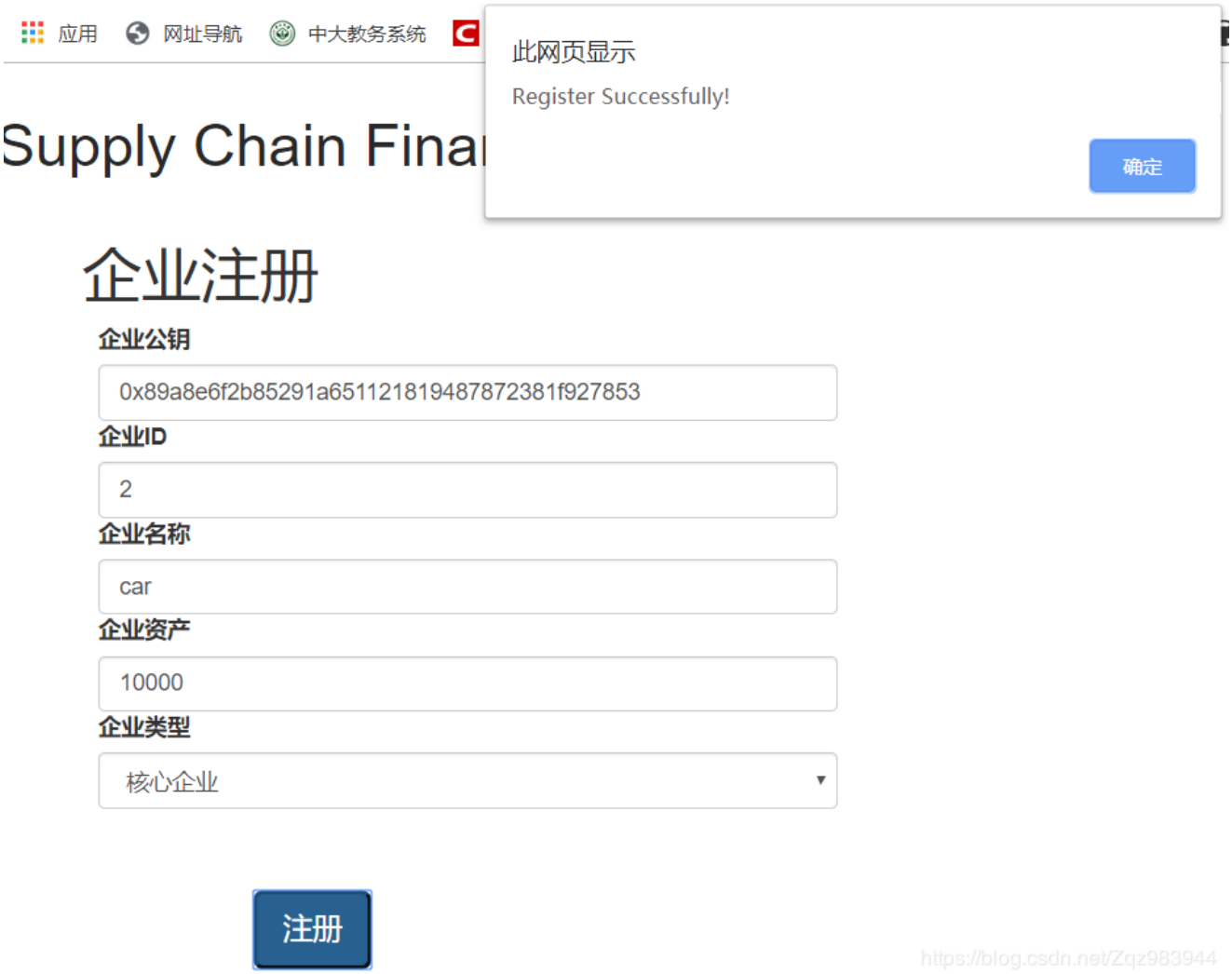
创建服务器并解析

```
var http = require('http');
var fs = require('fs');
var url = require('url');

// 创建服务器
http.createServer( function (request, response) {
   // 解析请求，包括文件名
   var pathname = url.parse(request.url).pathname;
   // 输出请求的文件名
   console.log("Request for " + pathname + " received.");
   // 从文件系统中读取请求的文件内容
   fs.readFile(pathname.substr(1), function (err, data) {
      if (err) {
         console.log(err);
         // HTTP 状态码: 404 : NOT FOUND
         // Content Type: text/plain
         response.writeHead(404, {'Content-Type': 'text/html'});
      }else{
         // HTTP 状态码: 200 : OK
         // Content Type: text/plain
         response.writeHead(200, {'Content-Type': 'text/html'});

         // 响应文件内容
         response.write(data.toString());
      }
      //  发送响应数据
      response.end();
```

```
    });
}).listen(4000);

// 控制台会输出以下信息
console.log('Server running at http://127.0.0.1:4000/index.html');
```

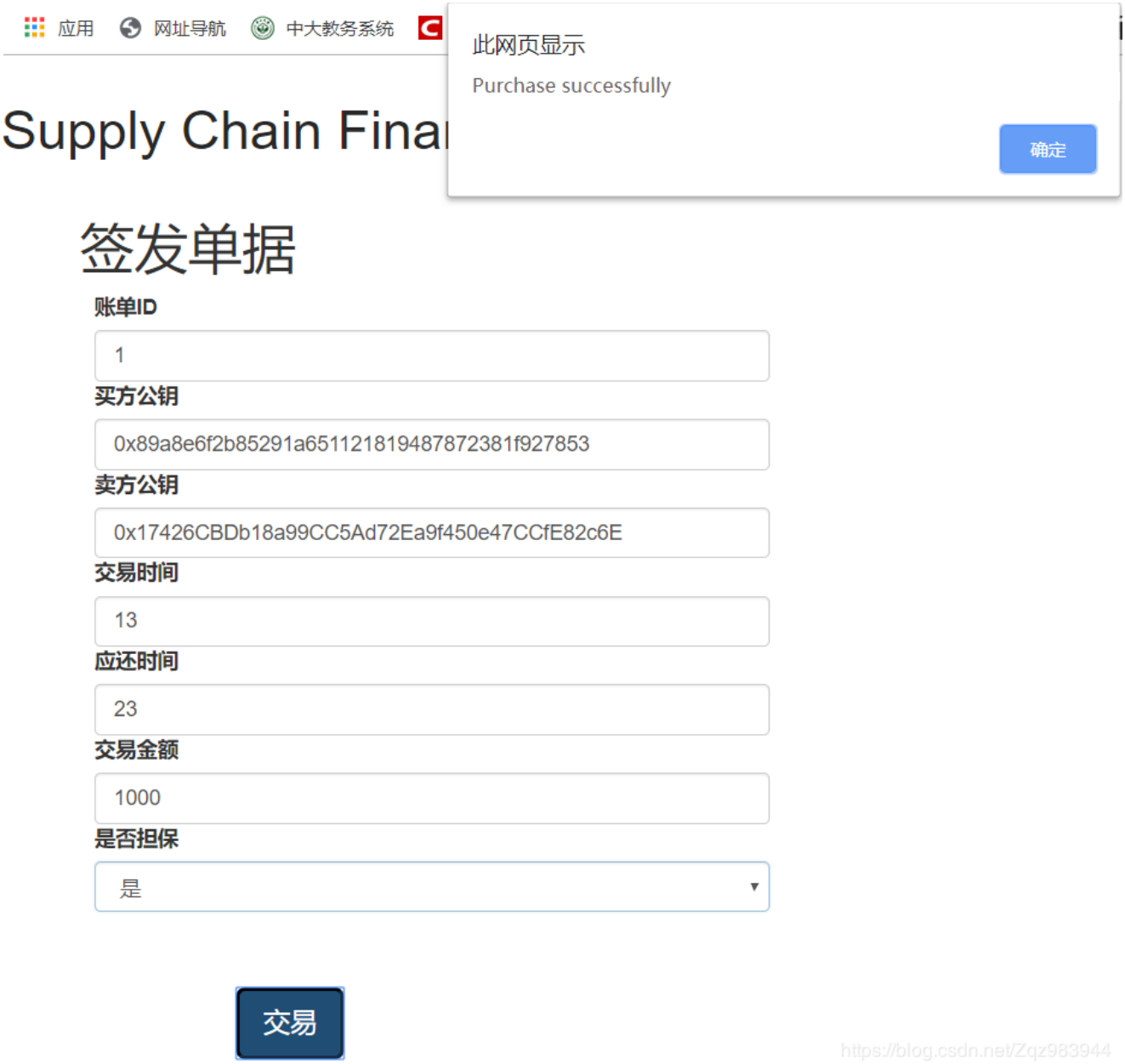
三、功能测试

企业注册

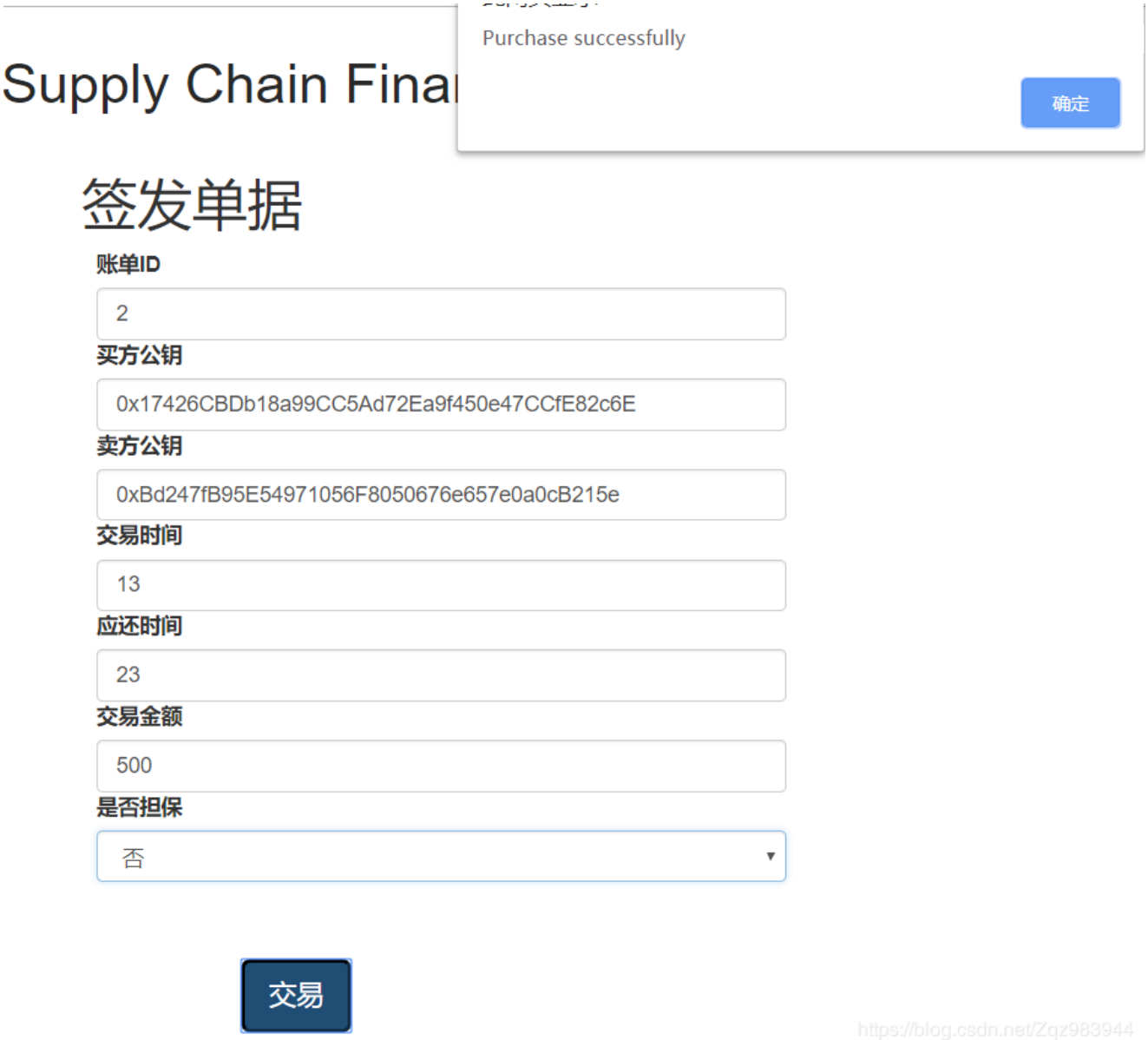


前端网页输入企业公钥、企业id、企业名称、企业资产和企业类型即可注册

签发单据

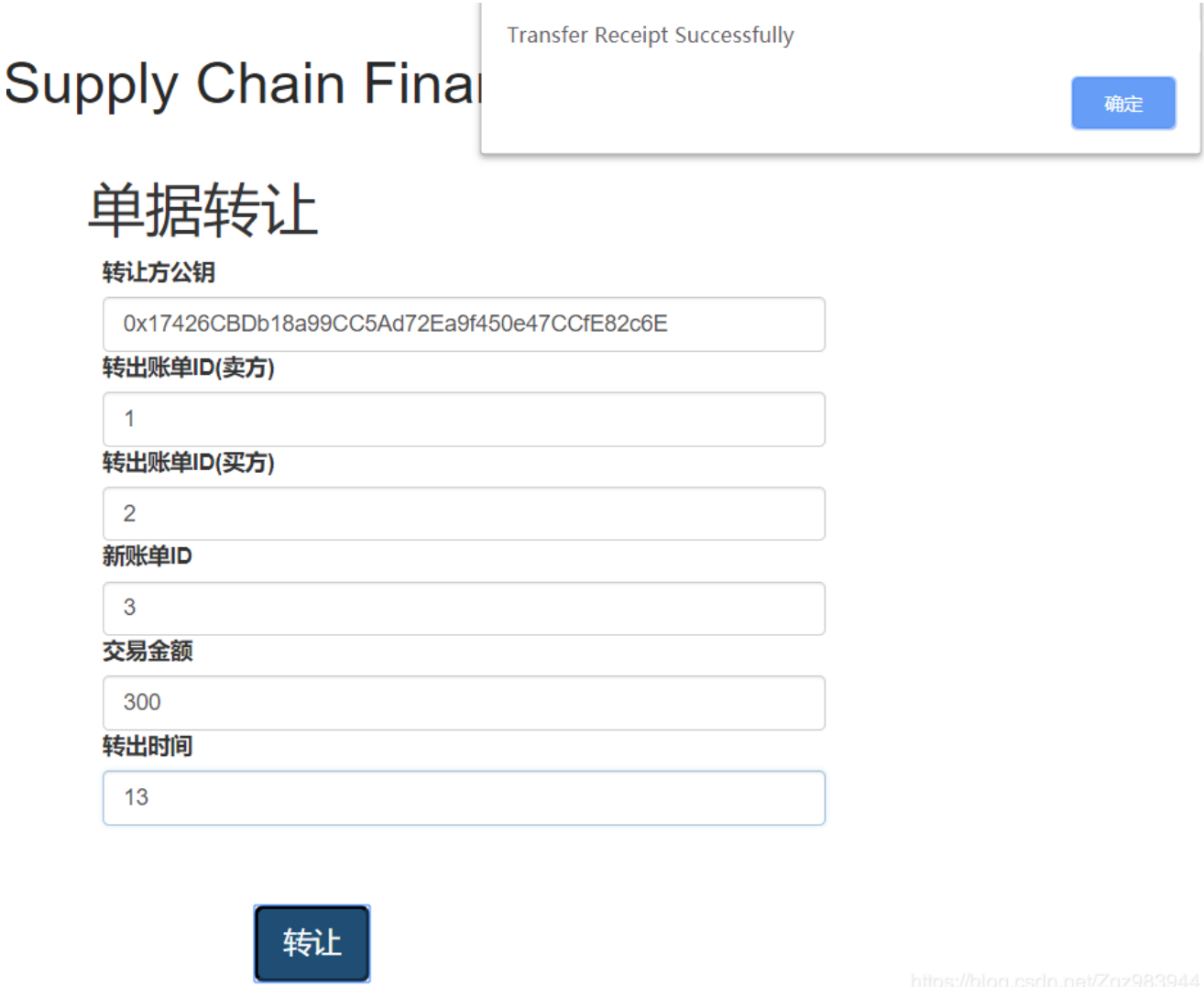


输入买方和卖方地址以及交易id，交易时间和应还时间可以发起交易，本例中即为汽车公司向轮胎公司购买1000元的货品，签发收据，并标记为被信任交易



轮胎公司向轮毂公司发起交易金额为500的交易并未被担保

转发单据



轮胎公司将与轮毂公司的500元的单据拆分为汽车公司与轮毂公司300，轮胎公司与轮毂公司200

转让后的单据为：

id	应付款方	应收款方	金额	还款期限	是否被信任
1	汽车公司	轮胎公司	700	23	是
2	轮胎公司	轮毂公司	200	23	否
3	汽车公司	轮毂公司	300	23	是

企业还款

汽车公司发起还款与轮毂公司的单据

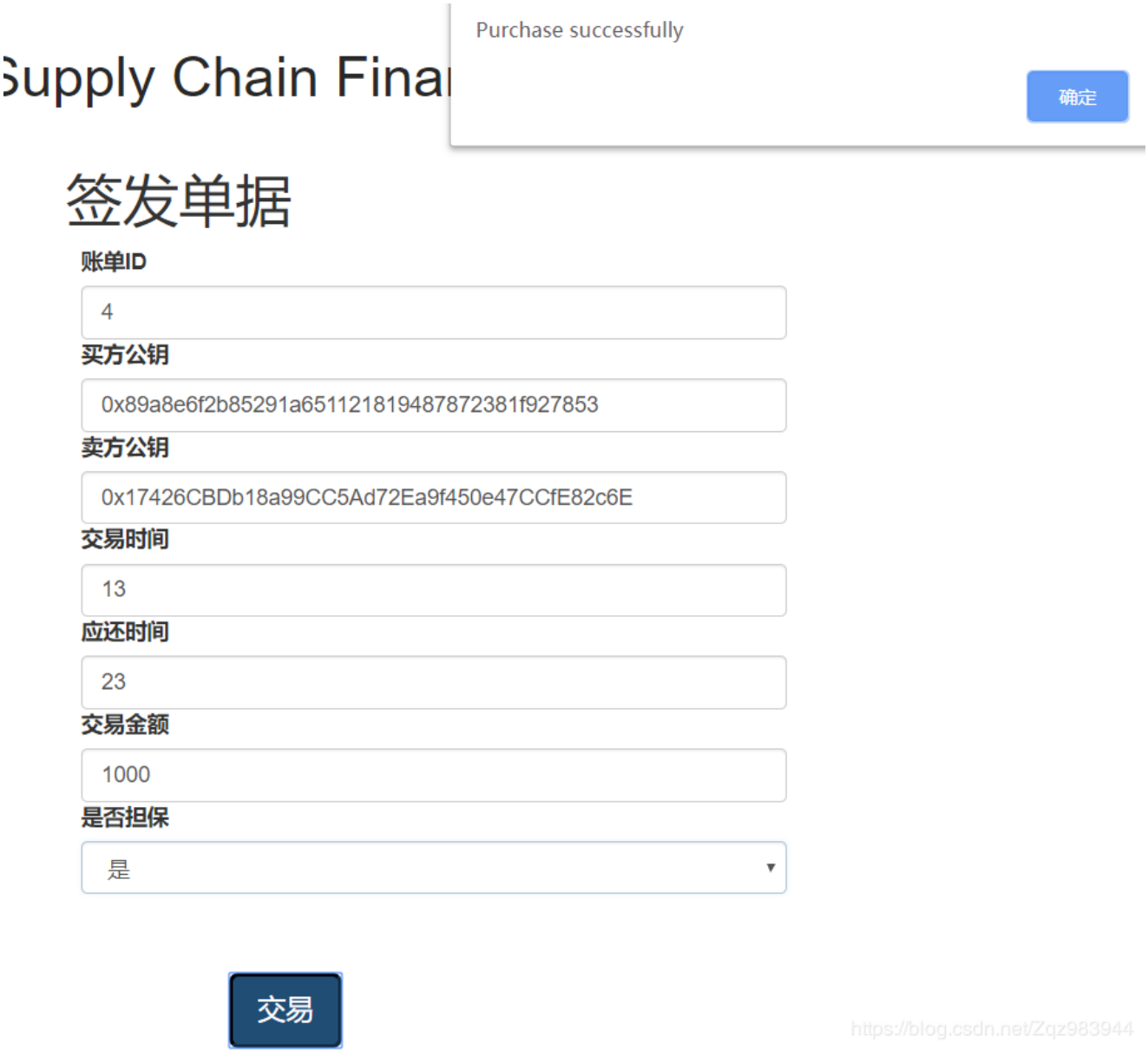
查询汽车公司余额（这里后来初始资金改成100000了所以还款300后还剩99700)

银行融资

银行向轮胎公司融资

融资之前轮胎公司余额

签发单据



发起融资


Finance successfully
确定
Supply Chain Finar
银行融资
银行公钥
0xDa3cB6CaFed3eDEA36dba98dc1a56b2d847FaB1D
企业公钥
0x17426CBDb18a99CC5Ad72Ea9f450e47CCfE82c6E
融资账单ID
4
融资

融资成功


101500
确定
Supply Chain Finar
余额查询
企业公钥
0x17426CBDb18a99CC5Ad72Ea9f450e47CCfE82c6E
查询

四、界面展示

初始界面

Supply Chain Financing System

企业注册 签发单据 单据转让 余额查询 企业还款 银行融资

企业注册

Supply Chain Financing System

企业注册

企业公钥

addr

企业ID

id

企业名称

name

企业资产

property

企业类型

银行

注册

签发单据

Supply Chain Financing System

签发单据

账单ID

id

买方公钥

addr

卖方公钥

addr

交易时间

sell time

应还时间

expire time

交易金额

money

是否担保

否

交易

单据转让

Supply Chain Financing System

单据转让

转让方公钥

Company Addr

转出账单ID(卖方)

oldSell id

转出账单ID(买方)

oldPur id

新账单ID

new id

交易金额

money

转出时间

transfer time

转让

余额查询

Supply Chain Financing System

余额查询

企业公钥

addr

查询

企业还款

Supply Chain Financing System

企业还款

买方公钥

addr

账单ID

id

还款时间

settle time

还款

银行融资

Supply Chain Financing System

银行融资

银行公钥

bank addr

企业公钥

com addr

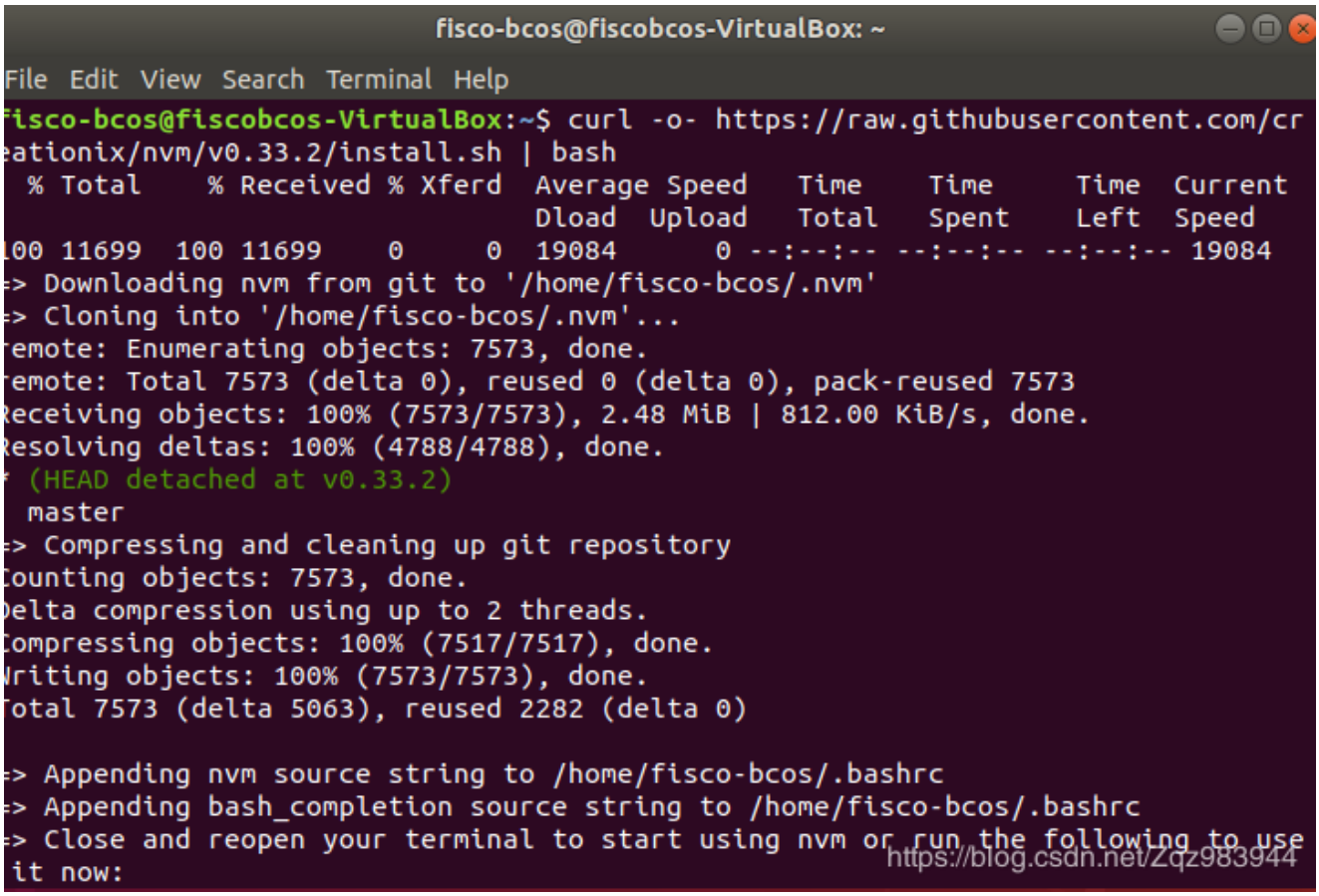
融资账单ID

id

融资

五、部署过程

环境配置，node.js环境，使用nvm快速部署




```
fisco-bcos@fiscobcos-VirtualBox:~$ source ~/.$(basename $SHELL)rc
fisco-bcos@fiscobcos-VirtualBox:~$ nvm install 8
Downloading and installing node v8.16.2...
Downloading https://nodejs.org/dist/v8.16.2/node-v8.16.2-linux-x64.tar.xz...
######################################################################## 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v8.16.2 (npm v6.4.1)
Creating default alias: default -> 8 (-> v8.16.2)
fisco-bcos@fiscobcos-VirtualBox:~$ nvm use 8
Now using node v8.16.2 (npm v6.4.1)
```

安装第三方依赖

```
fisco-bcos@fiscobcos-VirtualBox:~$ git clone https://github.com/FISCO-BCOS/nodejs-sdk.git
Cloning into 'nodejs-sdk'...
remote: Enumerating objects: 74, done.
remote: Counting objects: 100% (74/74), done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 371 (delta 31), reused 43 (delta 19), pack-reused 297
Receiving objects: 100% (371/371), 120.79 KiB | 119.00 KiB/s, done.
Resolving deltas: 100% (195/195), done.
fisco-bcos@fiscobcos-VirtualBox:~$ cd nodejs-sdk
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk$ npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
added 699 packages from 379 contributors and audited 40388 packages in 197.181s
found 0 vulnerabilities


      New minor version of npm available! 6.4.1 → 6.13.4
    Changelog: https://github.com/npm/cli/releases/tag/v6.13.4
```


启动节点

```
fisco-bcos@fiscobcos-VirtualBox:~/fisco$ bash nodes/127.0.0.1/start_all.sh
try to start newNode
try to start node0
try to start node1
try to start node2
try to start node3
 node1 start successfully
 node0 start successfully
 newNode start successfully
 node3 start successfully
 node2 start successfully
```


部署合约

```
[group:1]> deploy SupplyChain.sol
contract address: 0x0ae253c1d1f88eac5a1265877a6ef5dafdec29b2
```

```
fisco-bcos@fiscobcos-VirtualBox:~/nodejs-sdk/packages/cli$ ./cli.js deploy Hello World
```

获得账户公钥

```
fisco-bcos@fiscobcos-VirtualBox:~/fisco/console$ ./sol2java.sh org.com.fisco

Compile solidity contract files to java contract files successfully!
fisco-bcos@fiscobcos-VirtualBox:~/fisco/console$ bash get_account.sh
[INFO] Account Address   : 0xf1ec2d2a84b611755354844dd2af8480bc0ab2da
[INFO] Private Key (pem) : accounts/0xf1ec2d2a84b611755354844dd2af8480bc0ab2da.p
em
[INFO] Public  Key (pem) : accounts/0xf1ec2d2a84b611755354844dd2af8480bc0ab2da.p
ublic.pem
fisco-bcos@fiscobcos-VirtualBox:~/fisco/console$ bash get_account.sh
[INFO] Account Address   : 0x9af591c9187191116c734bce6c06b386bb3fb442
[INFO] Private Key (pem) : accounts/0x9af591c9187191116c734bce6c06b386bb3fb442.p
em
[INFO] Public  Key (pem) : accounts/0x9af591c9187191116c734bce6c06b386bb3fb442.p
ublic.pem
fisco-bcos@fiscobcos-VirtualBox:~/fisco/console$ bash get_account.sh
[INFO] Account Address   : 0x033c976c7e4e13c37d601bf74d495e6d72839926
[INFO] Private Key (pem) : accounts/0x033c976c7e4e13c37d601bf74d495e6d72839926.p
em
[INFO] Public  Key (pem) : accounts/0x033c976c7e4e13c37d601bf74d495e6d72839926.p
ublic.pem
```


六、心得体会

本次作业对上一次作业没有完善的合约功能进行了改进

- 在数据结构上，把Receipt放到了每个公司的结构体里，而不是单列一个映射，这样方便在单据转让和银行融资的时候判断企业的可信度和单据的来源。
- 增加了公司信誉度和单据可靠度的评估，上次作业由于不是很明白是什么意思，就只是创建了那个数据结构，并没有对其进行赋值和使用。
- 增加了单据的创建和应还款日期，这样方便企业还款的时候判断是否单据已逾期，从而对企业的信誉度做出改变，若按时还款则信誉度增加否则信誉度减少
- 银行融资的逻辑改变了，上一次银行融资就是单纯的把企业的资产增加，后来了解到银行融资应该是根据企业的应收账款进行融资，于是改成企业利用其应收账款单据向银行融资，而后银行变为单据的债权方，企业余额增加单据金额，这样才会有题目中说的下游企业可以通过应收账款单据向银行发起融资。比如轮毂公司开始没有受信任的单据，作为下游企业的信誉评估也不够，无法向银行发起融资，当轮胎公司发起单据转让后，轮毂公司就有了来自核心企业汽车公司的受信任的单据，于是可以向银行发起融资。
- 增加了单据的id，上次在实验报告中有写，上次还款的代码逻辑是输入应收款企业的地址，这样不能确认要还款的是两家公司之间的哪一个单据，增加单据id之后可以直接对单据进行操作，并且识别出交易双方，更加符合现实情况

遇到的问题

- 合约部署就不说了，全是问题

- 前端和链端不同步，最开始在前端进行操作，链端显示事务发送成功，但是调用getBalance之后发现没有改变，上网查了一下要 `sendTransaction` 才会对数据进行改变
- 通过前端调用合约函数返回的是交易事务的哈希值，不能直接想要输出合约return的值

总的来说通过本次大作业，我对区块链的前端后端链端的连接和交互方式有了一定的了解，从第二次完全不知道那几个公司是什么关系到后来可以自己思考并对合约做出调整，总共用了好几百的快高部署和调试，我也在一次次的失败中对去中心化应用更加熟悉。不过之前没有接触过web开发，做这些还是有点太费力，不知道为什么我一把js文件和html拆开就运行不了了然后只能每一个html文件都跟着一大串script，然后用户界面也不是很美观，实在是没有精力弄了。值得庆幸的是上次弄清楚了webase平台怎么使用，然后合约调试的时候更加方便，不用每次都在链端输一堆命令行。

通过这次作业我感受到了区块链去中心化应用的魅力所在，供应链金融能够助理中小企业融资，帮助中小企业评估风险控制机制，同时不同参与者使用一致的数据来源也保证了供应链信息的可追溯性，实现供应链透明化，降低了融资成本，支持信用资产拆分能够实现供应链线上资产高效流转，但同时应该注意链上数据隐私保护，如进行同态加密，属性加密等，这里我就做得不是很好，可以通过一个单据获得对方公司的所有信息，这其实隐含大量安全问题。
区块链课程也要走近尾声了，希望这次作业是激起我未来学习区块链的一大动力。