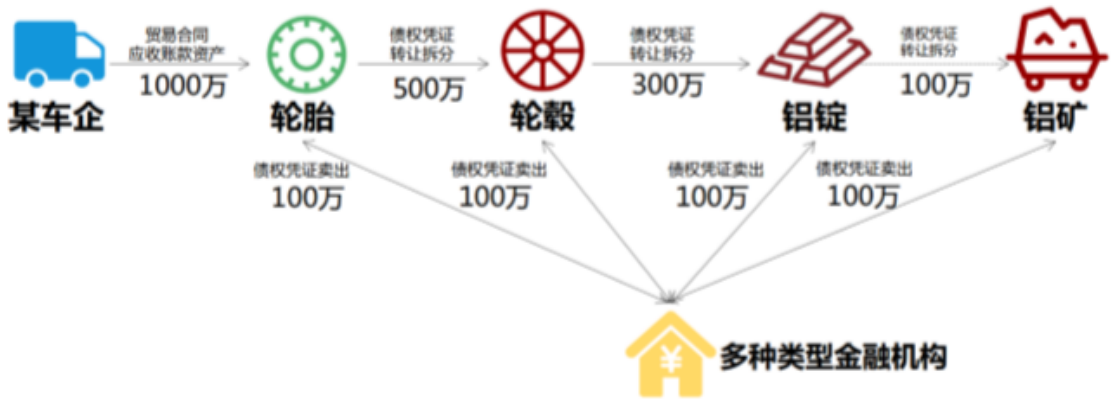
数据科学与计算机学院本科生实验报告（2019年秋季学期）

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	17343037	姓名	海明皓
电话	18054235757	Email	1492012973@qq.com
开始日期	2019/12/8	完成日期	2019/12/13

一、项目背景

本项目是基于已有的开源区 块 链 系 统 FISCO-BCOS，以联盟链为主，开发的基于区块链智能合约的供应链金融平台，实现了供应链应收账款资产的溯源、流转。

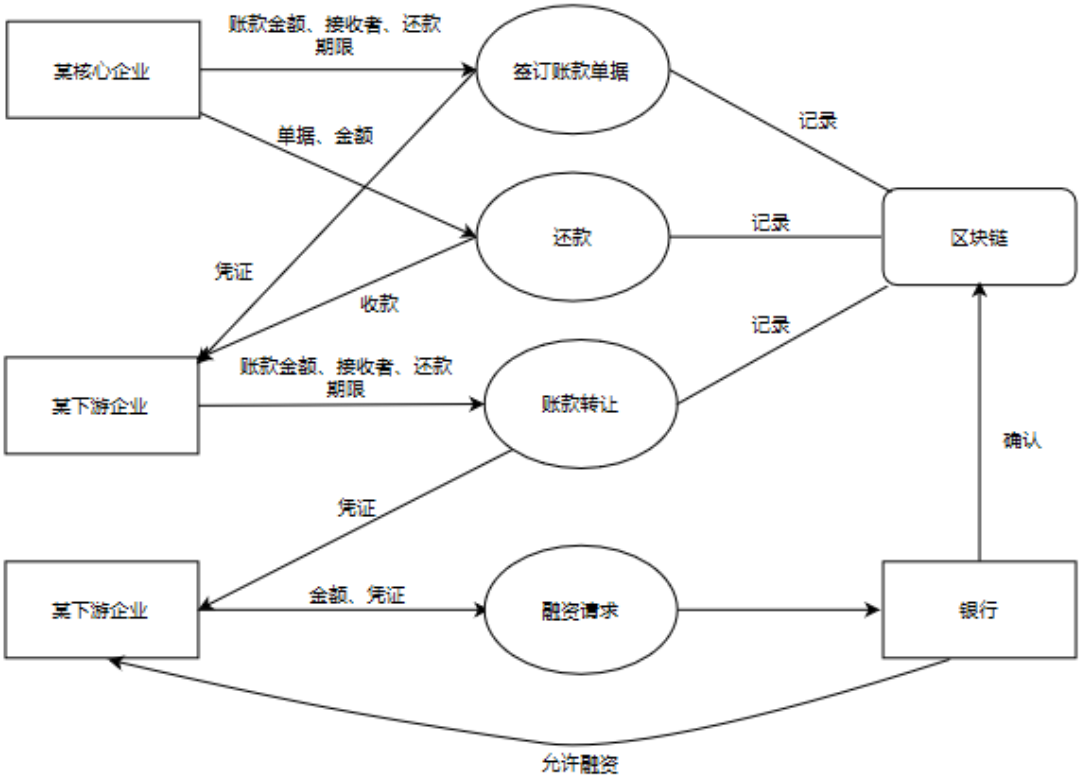
比如某场景如下图所示：



我们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

为了完成这四个核心功能，我们设计数据流图如下所示：



- 结构体与数据结构声明

  我们定义了公司的信息如下，包括了公司的名字、地址、单据、单据数量、以及是否是核心企业：

```
struct Company
{
    string name;                    // name
    string addr;                    // address
    Receipt[] receipts;             // receipts
    uint receiptNum;                // number of receipts
    bool isCore;                    // whether is core company
}
```

  我们定义的单据信息如下，包括了欠款人、收款人、金额、状态、是否已还：

```
struct Receipt
{
    address fromCompany;            // debt side
    address toCompany;              // receiving side
    uint money;                     // amount of money
    string status;                  // status
    bool isPayed;                   // whether the receipt is payed
}
```

  为了让公司的内存地址和其结构体一一对应，我们定义了一个map结构如下所示：

```
mapping (address => Company) public companys;
```

  为了在后续的测试功能上，方便体现出信息，我们定义了一个公司和金额对应的map结构，以及公司和信用额度对应的map结构：

```
mapping (address => uint) public balance;
mapping (address => uint) public credit;
```

最后，是定义了当前的核心公司以及银行的内存地址：

```
address private coreCompany;
address private bank;
```

在构造函数中，我们对其进行初始化。银行的金额设置成99999（较大），并未每个公司设置了初始的信用额度和金额：

```
constructor() public
{
    coreCompany = msg.sender;
    bank = 0x902252c74d3d055a21dfbd01e445a5a0c4303f9d;
    // set the address of bank
    balance[bank] = 99999;
    // init the amount of bank

    companys[0x643d8fbbd51e55643717ff33d1d03300c82d50e4].name = "车企公司";
    companys[0x643d8fbbd51e55643717ff33d1d03300c82d50e4].addr = "xx省xx市xx区
xx号";
    companys[0x643d8fbbd51e55643717ff33d1d03300c82d50e4].isCore = true;
    credit[0x643d8fbbd51e55643717ff33d1d03300c82d50e4] = 2000;
    balance[0x643d8fbbd51e55643717ff33d1d03300c82d50e4] = 2000;

    // ...
}
```

- 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。

  在这个功能中，首先我们需要判断当前公司是否有足够的信用额度来签订这笔单据：

```
if (credit[msg.sender] < amount)
    revert("Your credit is not enough");
```

如果有足够的信用额度，我们就将单据创建好并放进receiver的数组中进行保存，将欠款人和收款人、金额等信息设置好：

```
companys[receiver].receipts.push(Receipt({
    fromCompany:msg.sender,
    toCompany:receiver,
    money:amount,
    status:"",
    isPayed:false
}));
companys[receiver].receiptNum ++;
```

如果收款人为银行，我们就变动金额数量，否则就更改信用额度：

```
if (receiver == bank)
{
    balance[msg.sender] += amount;
    balance[receiver] -= amount;
}
else
{
    credit[msg.sender] -= amount;
    credit[receiver] += amount;
}
```

- 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。

  这个功能的实现，主要是看当前公司持有的所有单据中是否有足够的钱来创建一笔新的单据：

```
uint length = companys[msg.sender].receiptNum;
for (uint i = 0; i < length; ++ i)
{
    if (companys[msg.sender].receipts[i].money >= amount)
    {
        // ...
        break;
    }
}
```

  如果if语句为真，我们就利用功能一的函数，来创建一笔新的单据，并将原先单据中的金额数量改变：

```
companys[msg.sender].receipts[i].money -= amount;
signReceipt(receiver, amount);
```

- 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。

  和功能二类似，我们同样遍历当前公司所持有的所有单据，如果某单据中有足够多的钱，那么银行就准许这笔融资请求：

```
uint length = companys[msg.sender].receiptNum;
for (uint i = 0; i < length; ++ i)
{
    if (companys[msg.sender].receipts[i].money >= amount)
    {
        // ...
        break;
    }
}
```

  此时，应该将单据的首款人变成银行：

```
companys[msg.sender].receipts[i].money -= amount;
signReceipt(bank, amount);
```

- 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。

  我们遍历当前公司欠下游某企业所有的单据，找到那些还没有还款的单据：

```
uint length = companys[receiver].receiptNum;
for (uint i = 0; i < length; ++ i)
{
    if (companys[receiver].receipts[i].fromCompany == msg.sender &&
!companys[receiver].receipts[i].isPayed)
    {
        // ...
    }
}
```

如果当前公司没有足够多的金额来还款的话，函数停止：

```
if (balance[msg.sender] < amount)
    revert("Your balance is not enough");
```

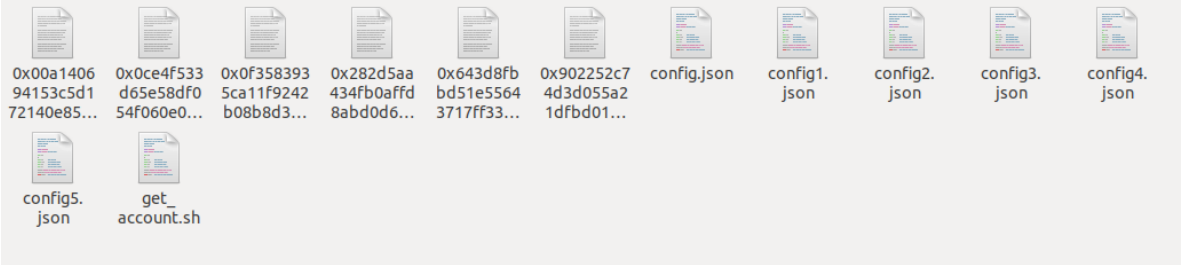
否则，就对应更改单据信息，并更新公司对应的金额和信用额度：

```
companys[receiver].receipts[i].isPayed = true;
// pay the money
balance[msg.sender] -= amount;
balance[receiver] += amount;
// recovery the credit
credit[msg.sender] += amount;
credit[receiver] -= amount;
```

三、源代码说明

在本次的设计上，前端我们使用了Vue框架搭建，后端使用了nodejs搭建服务端，链端是使用了提供的fisco-bcos作为链端。在后端和链端的交互上，我使用了nodejs的SDK。

由于本次一共设置了六个不同的账户，分别是银行、车企公司、轮胎公司、轮毂公司、铝锭公司、铝矿公司：



我们在服务端需要根据前端的请求，设置不同的合约调用者：

```
function changeConfig(name)
{
    Configuration.reset();
    switch(name)
    {
        case "车企公司":
            Configuration.setConfig(path.join(__dirname,
'../conf/config1.json'));
            break;

        // ...
    }
```

```
    cnsService.resetConfig();
    permissionService.resetConfig();
    web3jService.resetConfig();
}
```

在调用合约的时候，我们可以根据官方文档中的调用api：

getSystemConfigByKey	获取系统配置	系统配置关键字，目前 ： - tx_count_limit - tx_gas_limit
sendRawTransaction	发送交易	交易的RLP编码
deploy	部署合约	合约路径 输出路径
call	调用合约	合约地址 调用接口的ABI 参数列表

使用发送交易的api来调用之前合约中的函数：

```
function deploy(functionName, parameters)
{
    cnsService.queryCnsByNameAndVersion("SupplyChain", "0.2").then(queryResult
=> {
        var contractAddress = queryResult[0].address;
        var abi = JSON.parse(queryResult[0].abi);
        var index = functionName.indexOf('(');
        var name = functionName.substr(0, index);
        var item = findItem(abi, name);
        web3jService.sendRawTransaction(contractAddress, functionName,
parameters).then(result => {
            // ...
        });
    });
}
```

在一些请求中，我们可能需要获取到合约中的变量，比如执行完交易之后，我们需要获取本次交易的信息。我们首先在合约中保存一个变量，方便我们后续进行获取：

```
Receipt public receipt;

receipt.fromCompany = msg.sender;
receipt.toCompany = receiver;
receipt.money = amount;
receipt.status = "ok";
receipt.isPayed = false;
```

然后我们在服务端中，使用`sendRawTransaction`方法执行合约中的函数，我们创建了一个data变量用以保存本次交易的所有信息，并将其传送回前端：

```
web3jService.sendRawTransaction(contractAddress, functionName,
parameters).then(result => {
    // console.log(result);
    let output = result.output;
```

```
    if (output !== '0x') {
        output = utils.decodeMethod(item, output);
    }
    console.log(output);
    var data = {
        fromCompany: '',
        toCompany: '',
        money: '',
        status: '',
        isPayed: '',
        msg: ''
    };
    data.fromCompany = output.fromCompany;
    data.toCompany = output.toCompany;
    data.money = output.money;
    data.status = output.status;
    data.isPayed = output.isPayed;
    res.writeHead(200, {'Content-Type': 'text/plain; charset=utf-8'});
    if (result.status == '0x0') {
        data.msg = "success";
        res.write(JSON.stringify(data));
    } else {
        data.msg = "failed";
        res.write(JSON.stringify(data));
    }
    res.end();
});
```

要注意的是，output需要解析出来，我们首先需要获取函数对应的abi，也就是item变量，首先我们根据传入函数的左括号来提取出函数名：

```
var abi = JSON.parse(queryResult[0].abi);
var index = functionName.indexOf('(');
var name = functionName.substr(0, index);
var item = findItem(abi, name);
```

然后，调用findItem函数，利用循环解析出我们所需要的abi：

```
function findItem(abi, name)
{
    for (var obj of abi)
    {
        if (obj.name == name)
        {
            return obj;
        }
    }
}
```

四、功能测试

我们首先打印出了所有公司的信息，如下所示：

公司名	公司地址	公钥地址信息	是否为核心企业
银行	xx省xx市xx区xx号	0x902252c74d3d055a21dfbd01e445a5a0c4303f9d	/
车企公司	xx省xx市xx区xx号	0x643d8fbbd51e55643717ff33d1d03300c82d50e4	是
轮胎公司	xx省xx市xx区xx号	0x282d5aa434fb0affd8abd0d6fcba7e6cca2945e3	否
轮毂公司	xx省xx市xx区xx号	0x0f3583935ca11f9242b08b8d38126a004818f7a0	否
铝锭公司	xx省xx市xx区xx号	0x0ce4f533d65e58df054f060e07c4175c1d16256d	否
铝矿公司	xx省xx市xx区xx号	0x00a140694153c5d172140e859a555febeb0656e8	否

然后分别查看车企公司、轮胎公司和轮毂公司的金额和信用额度：

935ca11f9242b08b8d38126a004818f7a0

查询

金额	信用度
1000	1000

可以看见，现在车企公司的信用额度和金额都是2000万元，而轮胎公司和轮毂公司的信用额度和金额都是1000万元。

然后我们模拟执行一笔交易，假定车企公司从轮胎公司采购一批轮胎并签订了应收账款单据，这笔交易的金额为1000万元：

车企公司

aa434fb0affd8abd0d6fcba7e6cca2945e3

1000

确认

执行完成之后，我们分别查看车企公司和轮胎公司的信用额度和金额：

bbd51e55643717ff33d1d03300c82d50e4

查询

金额	信用度
2000	1000

金额	信用度
1000	2000

此时，因为这笔交易的产生，车企公司信用额度减少了，而轮胎公司的信用额度增加了，也就是说其可以利用这笔单据进行转让或者进行融资。

我们首先模拟单据的转让，假定此时轮胎公司向轮毂公司转让500万元：

由于此时轮胎公司的信用额度满足要求，所以交易被允许。

我们查询此单据的信息，如下所示：

欠款人	收款人	金额(万元)	状态	是否已还
0x282d5aa434fb0affd8abd0d6fcba7e6cca2945e3	0x0f3583935ca11f9242b08b8d38126a004818f7a0	500	已认证	否

更新

轮胎公司此时还有1500万元的信用额度：

我们执行融资，如果融资额度高出信用额度，则融资失败：

如果融资金额为1000万元，小于1500万元，故融资成功：

五、界面展示

信息概览页面：

金额查询页面：

单据签署页面：

单据查询页面：

融资申请页面：

六、心得体会

一开始接触大项目的时候，还不知道什么叫做融资、什么叫做应收账款的转让，经过了本次实验，我对整个供应链的运行流程熟悉了不少，基本上了解了相关的概念，知道了核心企业什么时候进行签发应收账款、得到单据的下游企业如何使用这笔账款进行融资、转让等等，以及银行这类第三方机构在供应链中的作用等等。

大项目的过程中，我们需要自己完成一个供应链智能合约的编写，也让我在写合约的同时熟悉了solidity语言，能够按照自己的需求完成相对应的功能，比如交易上链、融资请求等等。

最后的部分，我们需要自己完成一个前端和后端的交互，并在后端调用api进行合约的部署以及使用。在服务计算课程中，正好写过一个小型的前后端分离的项目，所以也为本次的大作业打下了一些基础。调用api的过程中，因为一开始对api不太熟悉，逐渐调试了很久，在调试的过程中，也逐渐熟悉了前端和后端的交互、后端和链端的交互。在官方的cli命令行中，也有样例可以借鉴使用，虽然过程比较辛苦但最后还是完成了。

总的来说，本次大作业还是收获丰富，在完成作业的过程中，练习了自己独立阅读文档、查询资料的能力，又锻炼了自己的代码能力，之前并没有接触过solidity，nodejs也接触的比较少，本次能够完成一个完整的前后端，还是很有收获。