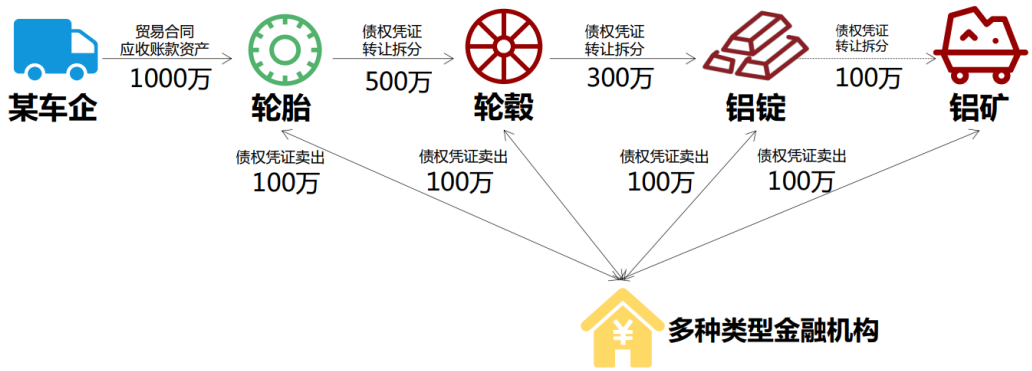
基于区块链的供应链金融平台实验报告

项目设计说明和功能测试文档：根据提供的供应链场景，基于FISCO-BCOS设计相关的智能合约并详细解释智能合约是如何解决提出的问题；同时将智能合约部署至链上（单节点or多节点），并调用相关函数，详细说明上述的功能具体是如何实现的。（截图说明调用结果）

小组成员学号	小组成员姓名	联系电话
18339001	曹斯华	18207593253
18339005	仇幸丹	13660481970
18339006	黄小娜	17827063340

一、项目背景

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、智能合约介绍

智能合约是指把合同/协议条款以代码的形式电子化地放到区块链网络上。通过调用相关条款的接口实现执行，也可以把它当成是一种可自动执行的程序片段。作为区块链的参与者，智能合约既可以接收和存储价值，也可以向外发送信息和价值。

FISCO BCOS平台支持两种智能合约类型：Solidity智能合约与预编译智能合约Solidity与Java类似。代码写好后，都需要通过编译器将代码转换成二进制，在Java中，编译器是Javac，而对于Solidity，是solc。生成后的二进制代码，会放到虚拟机里执行。Java代码在Java虚拟机（JVM）中执行，在Solidity中，是一个区块链上的虚拟机

EVM。目的，是给区块链提供一套统一的逻辑，让相同的代码跑在区块链的每个节点上，借助共识算法，让区块链的数据以统一的方式进行改变，达到全局一致的结果。

三、基于FISCO-BCOS相关的智能合约设计

基本功能如下：

功能一：实现采购商品一签发应收账款交易上链（实现采购商品）。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。

其他，此外还添加了信用值和还款截止时间，以此来实现贷款自动发放、到期自动结算、超期未还惩罚等功能。

首先要创建一个数据结构用于定义借据/欠条（这是整个供应链金融平台的基础），然后编写两个函数：一个是欠条的创建函数——供债权人创建欠条，相当于在空白欠条中填入债务人、债务金额等关键信息，将其发布到链上；另一个是欠条的有效性确认函数——供债务人对他的债权人发布的欠条信息进行确认。任何公司都可以调用该函数声称某人欠他任意数额的钱，但该欠条的有效性必须经过债务人的确认。确认表明债务人认可欠条的信息，承认自己欠借据上写的债权人相应债务金额的款项，并对其上的信用担保、归还期限等信息予以认可。

```
//借据的数据结构
struct Receipt {
    uint Contract_id;  //the id of the contract
    uint amount;       //the amount of money
    uint deadline;     //deadline
    address Debtors;   //those who owe the debt
    address Creditors; //those who have the debt
    bool confirmation; //Whether confirmed by the Debtors or not
    bool confidence;   //evaluated by the bank
    bool default; //true when the Receipt is unpaid after the deadline
}
```

变量类型	变量名	变量含义
uint	Contract_id	智能合约ID
	amount	借款数额
	deadline	归还欠款日期
address	Debtors	债务人
	Creditors	债权人
bool	confirmation	是否经债权人确认

	confidence	债务人的信用/还款能力
	default	记录是否到期未还

接下来要明确银行对债务人还款能力的评估部分。（即Receipt中的confidence变量由银行评估后设置）在一张欠条中，confidence属性为true明银行认为可该债务人的信用，对该债务人有能力偿还相应数额的钱有信心，银行的权威性使这张欠条被认为是低风险的（大概率能得到偿还）。这就将银行的风险评估与具体公司独立了出来，后续只需加入欠条的拆分转移功能，就可以将该信用传递下去。

第二步，创建一个数据结构用于定义每家公司的相关信息（公司名字地址以及不公开的账户余额等）。接着，该链上除了存储所有欠条、所有公司信息、所有公司银行账户余额（不公开）的变量外，还需要对每家公司维护一个“信用”变量（还款能力），当它大于某张欠条上的数额时，就相信该公司有能力偿还。

```
struct companyInfo {
    string Company_name;
    string Company_address;
    string More_details;
}
//设置公司名字（初始化）
function setCompany_name(address addr, string name) public returns (bool)
{
        //call by any company, set its info
        require(addr == msg.sender);
        Infos[msg.sender].companyName = name;
        return true;
    }
//设置公司地址（初始化）
function setCompany_address(address addr, string Caddr) public returns
(bool) {
        //call by any company, set its info
        require(addr == msg.sender);
        Infos[msg.sender].companyAddress = Caddr;
        return true;
    }
//设置公司相关信息（初始化）
function setMore_details(address addr, string description_) public returns
(bool) {
        //call by any company, set its info
        require(addr == msg.sender);
        Infos[msg.sender].description = description_;
        return true;
    }
Receipt[] public Receipts;
address[] public companies;  //registered companies' list
mapping (address => uint) balances;  //private
function checkBalance allows company to check its own balance
mapping (address => uint) public credits;  //credits are open to the
public
```

```
mapping (address => companyInfo) public Infos;
```

银行对欠条中债务人的还款能力评估（trust变量）的评估依据来源于债务人公司的的存款规模以及该公司的债务情况。所以银行要掌握每家公司的balances和credits，前者指向该公司的存款规模，后者指向该公司的还款能力（=存款规模-债务+债权）。每当该公司增加或减少一定数量（amount_）的存款时，该公司的balances和credits要同步增加或减少一定数量（amount_）。要将存款规模和还款能力分开为两个变量的原因是，当一家公司成为某张欠条的债务人后，在欠条到期前，他的存款并不会减少，可是他的债务情况增多了，相应的还款能力也就下降了。从上述说明就可以看出，balances和credits的变化并不同步，因此要将它们分开为两个变量。银行可分别通过 newBalance 和 newCredit两个函数来更新这两个值。

首先看balances变量，该变量指向公司的存款规模。显然，公司的存款规模并不是一个常量，而是随时可以变动的，所以应该设置一个函数用于公司资产状况的更新（增加或减少）。另外需要注意的是，1.该更新函数只能由银行这个权威机构调用，个人用户（公司）无权对资产状况进行更改；2.资产增加没有限制，但是资产减少（比如用存款去进行还款）的数额，不能大于存款数额（即存款不能变为负值）。

```
function newBalance(address receiver, uint amount_,
bool add) public returns (bool)
{
        //限制条件：该资产更新函数只能由银行调用
        require(msg.sender == 0xb820b9f01be068e273ac75e530f1f55c70cc648d);
        //该次更新是公司资产有所增加，则存款与信用值均上升相应数额
        if (add == true)
        {
            balances[receiver] += amount_;
            credits[receiver] += amount_;
            return true;
        }
        //如果add! =true，则说明此次更新是公司资产有所减少（可能用于还款），则存
          款与信用值均下降相应数额
        else if (balances[receiver] >= amount_ && credits[receiver] >=
amount_) //减少的金额不能超出存款规模
        {
            balances[receiver] -= amount_;
            credits[receiver] -= amount_;
            return true;
        }
        return false;
    }
```

然后看credits变量，该变量指向公司的还款能力。显然，公司的还款能力也并不是一个常量，而是随时可以变动的，所以应该设置一个函数用于公司还款能力的更新（增加或减少）。另外需要注意的是，该更新函数只能由银行这个权威机构调用，个人用户（公司）无权对还款能力进行更改。

```
function newCredit(address receiver, uint amount_, bool add) public
returns (bool) {
        //银行对公司的还款能力作增加或减少的调整
```

```
        require(msg.sender ==
    0xb820b9f01be068e273ac75e530f1f55c70cc648d);
        //增加还款能力
        if (add == true) {
            credits[receiver] += amount_;
            return true;
        }
        //减少还款能力
        else if (credits[receiver] >= amount_) {
            credits[receiver] -= amount_;
            return true;
        }
        return false;
    }
```

就像我们可以随时查询自己的账户余额一样，也应该允许公司随时查询自己的存款规模，这里用一个require(company == msg.sender)限制公司只能查询自家公司的存款规模，无权对其他公司的存款规模进行访问，因为存款规模是每家公司的隐私（私有变量）。

```
function checkBalance(address company) public returns (uint) {
        require(company == msg.sender);
        return balances[company];
}
```

定义好信用后，就可以实现makeReceipt函数和confirmReceipt 两个函数来创建并发布收据以及确认收据。

makeReceipt函数由债权人调用，它创建一个收据，填入 id、金额、债务人、时间的基本信息，以及是否需要对方有足够的还款能力（此时的存款规模足够大），并将其发布到链上(创建一个实例并添加到receipts数组中)。其中confirmation属性为false，需要等待债务人确认。default属性代表该欠条中债务人是否被银行认可（有能力能够相应数额的钱），该属性是由银行根据债务人的情况确定的，债权人在这里的设置只表达了它对信用担保的要求，即若该属性为 false，债务人不需要银行的认可即可确认这张欠条，否则则需要有相应数额的信用做担保。而属性deadline是一个时间戳，即债权人的还债期限(函数自动计算到期的绝对时间)。

```
function makeReceipt(uint Contract_id_, address receiver, uint amount_,
bool confidence_, uint time) public returns (bool) {
    //call by the Creditors
    //declare a Receipt(the receiver owes me money)
    Receipts.push(Receipt({
        Contract_id: Contract_id_,Debtors: receiver,
        Creditors: msg.sender,amount: amount_,
    confirmation: false,
        confidence: confidence_,//if true,debtor's ample credits is needed
        deadline: now + time,  //time indicates how many time left
        default: false
    }));
    return true;
```

```
}
```

当债权人在空白的欠条上填写完id、金额、债务人、时间这些基本信息之后，债务人要对该欠条的信息进行确认，经过债务人确认的欠条才是一张有效欠条，这表现为bool变量类型的confirmation。

```
function confirmReceipt(uint Contract_id_, address sender_, uint amount_, bool credit_) public returns (bool) {
    uint i;
    for (i = 0; i < Receipts.length; i++) {
        if (Receipts[i].Debtors == msg.sender && Receipts[i].Contract_id == Contract_id_ && Receipts[i].amount == amount_ && Receipts[i].Creditors == sender_ && Receipts[i].credit == credit_) {
            //时间戳与函数调用密切相关
            if (Receipts[i].confidence == false) //不需要进行还款能力验证
            {
              credits[Receipts[i].Debtors] -= Receipts[i].amount
              Receipts[i].confirmation = true;
              return true;
            }
            //if债权人需要检测债务人的还款能力 cre如果
            else if (credits[Receipts[i].Debtors] >= Receipts[i].amount) //债权人有能力归还这笔欠款
            {
                credits[Receipts[i].Debtors] -= Receipts[i].amount;
                credits[Receipts[i].Creditors] += Receipts[i].amount;
                Receipts[i].confirmation = true;
                return true;
            }
        }
    }
    return false;
}
```

下列函数允许公司查看以它为债务人的欠条（返回值为欠条在receipts数组中的下标集合）：

```
function checkReceiptDebtor(address company) public returns (uint[]) {
        require(company == msg.sender);
        //任何公司均可调用该函数来检查是否为收据的债务人
        uint[] ret;
        uint i;
        for (i = 0; i < receipts.length; i++) {
            if (company == receipts[i].Debtor && receipts[i].amount != 0) {
                ret.push(i);
            }
        }
        return ret;
    }
```

下列函数允许公司查看以它为债权人的欠条（返回值为欠条在receipts数组中的下标集合）：

```
function checkReceiptCreditor(address company) public returns (uint[]) {
        require(company == msg.sender);
        //任何公司均可调用该函数来检查是否为收据的债权人
        uint[] ret;
        uint i;
        for (i = 0; i < receipts.length; i++) {
            if (company == receipts[i].Creditor && receipts[i].amount != 0) {
                ret.push(i);
            }
        }
        return ret;
    }
```

由于公司的存款规模和还款能力是动态变化的，所以银行对公司是否有能力偿还某张欠条上的欠款金额也应该进行动态评估。因此，某张欠条上，债务人可能一开始被银行评估为偿还能力不足（confidence=false），但经过存款规模的增加或债务情况的减少，债务人有了足够的偿还能力，这时就可以调用以下函数请银行重新评估（即银行重新考虑在该欠条中对债务人公司的confidence）：

```
function evaluateReceipt(uint rid_, address debtor_) public returns (bool) {
        uint i;
        for (i = 0; i < receipts.length; i++) {
            if (receipts[i].Debtor == debtor_ && receipts[i].rid == rid_ && receipts[i].confidence== false
            && credits[receipts[i].Debtor] >= receipts[i].amount) {
        //原本偿还能力不足的公司经过动态变化后拥有足够的偿还能力
                credits[receipts[i].Debtor] -= receipts[i].amount;
                credits[receipts[i].Creditor] += receipts[i].amount;
                receipts[i].confidence = true;
                return true;
            }
        }
        return false;
    }
```

在本金融供应链平台上，债权具有可转让性，即可以沿着供应链向下流动。例：车企从轮胎公司购买了一批轮胎，向轮胎公司签订了1000万的应收账款单据（1000万的欠条，债务人：车企，债权人：轮胎公司）；轮胎公司可以凭此应收账款单据向轮毂公司购买了一批轮毂，向轮毂公司签订了500万的应收账款单据，这时轮胎公司就可以将车企向其签订的1000万应收账款单据部分转让给轮毂公司（500万的欠条，债务人：车企，债权人：轮胎公司），这中间涉及到了原本的债权拥有者轮胎公司将其一部分债权转让给轮毂公司的过程。

```
function transferReceipt(uint id_, uint newrid, uint newamount, address newcreditor) public returns (bool) {
```

```
        //有转让方调用，即一方面它是单据A的债权人，另一方面它即将成为单据B的债务人，此时它将单据A的债权（全部或部分）转让给单据B的债权人
        uint i;
        for (i = 0; i < receipts.length; i++) {
            //进行所属权的确认
            if (receipts[i].Contract_rid == id_ && receipts[i].Creditor == msg.sender && receipts[i].confirmation== true) {
   //债权人希望进行债权转让，转让是否成功取决于债权人是否拥有足够债权以抵消债务
                if (receipts[i].amount < newamount)
                    return false;  // 债权<债务
                else {//创建新的单据，转让部分债权
                    receipts[i].amount -= newamount;
                    receipts.push(receipt({
                    Contract_id: newrid,
                    Debtor: receipts[i].Debtor,
                    Creditor: newcreditor,//债权转让，债务人不变
                    amount: newamount,
                    confirmation: true,  //无需债务人重复确认
                    confidence: receipts[i].confidence,
                    deadline: receipts[i].deadline,
                    default: false
                }));
                if(receipts[i].confidence== true) {
                    credits[receipts[i].Creditor] -= newamount;
                    credits[newcreditor] += newamount;
                }//债权转让涉及转让双方债券情况的变化，即还款能力要相应更新
                    return true;
                }
            }
        }
        return false;  //找不到要转让债权的单据
    }
```

最后编写一个Repayment函数用于receipts的偿还部分，债务人可以通过向债权人偿还欠条中规定数额的债务。如果receipts中是被判定为违约的，则代表债务人没有做信用担保，因此在还款时就相当于一次普通的转账，信用、资金将一同转移到银行的账户中去。而对于其他已经担保过的欠条，债务人预支信用来支付，因此在还款的时候必须是还钱而不是继续使用信用支付，这样才能保证一致性，还完钱后不需要在 receipts 数组中找到对应位置的欠条并删除。

```
function Repayment(uint rid_, address creditor_, uint amount_) public returns (bool) {
        //用于债务人偿还收据中规定数额的债务
        uint i;
        for (i = 0; i < receipts.length; i++) {
            if (Receipts[i].Debtors == msg.sender && Receipts[i].Contract_id == Contract_id_ && Receipts[i].Creditors == Creditors_ && Receipts[i].amount == amount_) {
                if (balances[msg.sender] < amount_)
```

```
                    return false;  //表明公司无法偿还债务
                else {
                    receipts[i].amount = 0;
                    balances[msg.sender] -= amount_;
                    balances[receipts[i].Creditor] += amount_;
                    if (receipts[i].evaluated == false) {
                        credits[msg.sender] -= amount_;
                    //没有信用担保则还款时，信用、资金将一同转移
                        credits[receipts[i].Creditor] += amount_;
                    }

                    return true;
                }
            }
        }
    }
```

除此之外，为实现银行遍历每家注册公司并对其作为债务人的已经到期的欠条，自动强制执行偿还，若金额不足以偿还，则判定为违约，设置一个PayAuto函数，由银行调用，违约信息会通过default属性在欠条上记录，并扣除债务人 3 倍于欠条金额的信用值作为惩罚。注意银行需要在deadline到期后自动执行任何动作，也就是说银行需要在这个节点不断地进行轮询，每次调用该函数时根据当前时间做判断，并执行相应操作。

```
function PayAuto() public returns (bool) {
//c仅允银行调用，用于判断公司是否有需要归还的债务
//如果有，需要自动还款；如果到期且公司无钱归还，则标记该公司到期未还，且减少其信用值
        require(msg.sender ==
0x38c53e530f1f9f070cc648d1be068e273ac7555c);
        uint i;
        for (i = 0; i < receipts.length; i++) {
            //逾期未还
            if (now > receipts[i].deadline && receipts[i].amount != 0){
                if (balances[receipts[i].Debtor] s>= receipts[i].amount) {
                    receipts[i].amount = 0;
                    balances[msg.sender] -= receipts[i].amount;
                    balances[receipts[i].Creditor] += receipts[i].amount;
                    if (receipts[i].evconfidanc = false) {
                        credits[msg.sender] -= receipts[i].amount;
//未偿还债务，则扣除债务人欠条金额的信用
                        credits[receipts[i].Creditor] s+=
receipts[i].amount;
                    }
                }
                else if (receipts[i].default == false) {
//到期仍未归还且无法进行归还，则记录违约并扣除债务人3倍的欠条金额的信用
                    receipts[i].default = true;
                    credits[receipts[i].Debtors] -= 3 *
receipts[i].amount;
                }
```

```
            }
        }
        return true;
    }
```

四、功能测试

以下测试涉及四个用户：银行，公司A、B、C，

1.银行调用newBalance 函数向A打100 元：

2.银行调用newCredits函数给A增加500信用值：

3.公司A调用checkBalance函数查询其存款：

4.公司A还可以调用checkCredits函数查询银行认可它的还款能力（100+500=600）：

5.公司B作为债权人创建一张单据，填写基本信息：id、债务人、债权金额、是否要求验证还款能力、时间：

查询单据，此时由于该单据还未被公司A确认，因此（第五行）confirmation显示false：

6. A对B填写了基本信息后发布的单据（id为10）进行确认：

该单据经由A确认后，交易内容中的（第五行）confirmation变量变为true：

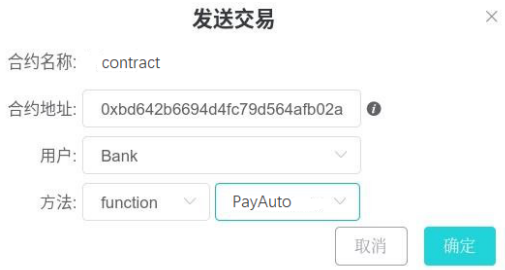
7. B把手上的100元债权，部分转让给C，转让50元，新单据ID为100

C可以查询到B转让给他的债权，债务人：A，债权人：B，金额：50

由于B的100元债权已经得到过A的确认，因此拆分出来的50元债权的confidence和confirmation变量与原100元债权是一致的：

8.C向银行贷款10元（C有B转让的50元债权）：

9.欠条到期，公司A没有准时还钱，因此银行调用PayAuto函数进行到期欠条自动结算：

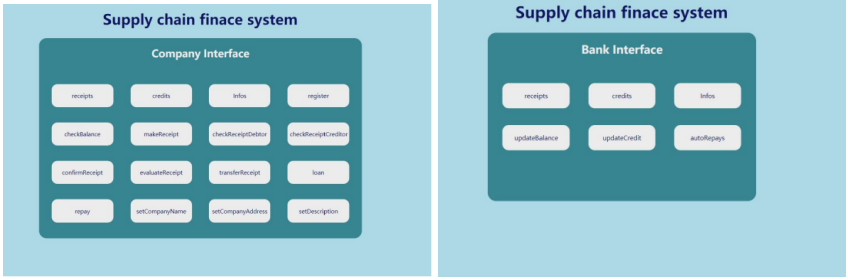


五、界面展示

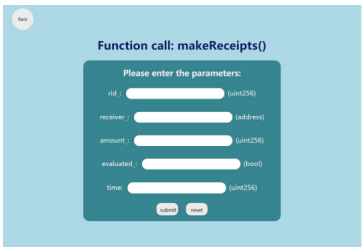
通过webase提供接口，发送post请求，对链端部署的合约进行操作：

```
[cowcow1007@localhost webase-deploy]$ python deploy.py installAll
================================================================
    _    _      _____   ___   _____  _____
   | |  | |    | ___ \ / _ \ /  ___||  ___|
   | |  | | ___| |_/ // /_\ \`--.| |__
   | |/\| |/ _ \ ___ \|  _  |`--. \  __|
   \  /\  /  __/ |_/ /| | | /\__/ / |___
    \/  \/ \___\____/\_| |_\____/\____/

================================================================
==================== envrionment check... ====================
check git...
check finished sucessfully.
check openssl...
check finished sucessfully.
check curl...
check finished sucessfully.
check nginx...
check finished sucessfully.
check java...
check finished sucessfully.
check FISCO-BCOS node port...
check finished sucessfully.
check WeBASE-Web port...
check finished sucessfully.
check WeBASE-Node-Manager port...
check finished sucessfully.
check WeBASE-Front port...
check finished sucessfully.
check database connection...
check finished sucessfully.
==================== envrionment ready... ====================
================================================================
====================    deploy   start... ====================
================================================================
==============      FISCO-BCOS     install... ==============
----------------------------------------------------------------
Generating CA key...
================================================================
Generating keys ...
Processing IP:127.0.0.1 Total:2 Agency:agencyA Groups:1,2
================================================================
Generating configurations...
Processing IP:127.0.0.1 Total:2 Agency:agencyA Groups:1,2
================================================================
Group:1 has 2 nodes
Group:2 has 2 nodes
================================================================
[INFO] Execute the following command to get FISCO-BCOS console
 bash <(curl -s https://raw.githubusercontent.com/FISCO-BCOS/console/master/to
s/download_console.sh)
================================================================
[INFO] FISCO-BCOS Path    : /home/cowcow1007/webase-deploy/nodes/fisco-bcos
[INFO] IP List File       : nodeconf
[INFO] Start Port         : 30300 20200 8545
[INFO] Server IP          : 127.0.0.1:2
[INFO] State Type         : storage
[INFO] RPC listen IP      : 0.0.0.0
[INFO] Output Dir         : /home/cowcow1007/webase-deploy/nodes
[INFO] CA Key Path        : /home/cowcow1007/webase-deploy/nodes/cert/ca.key
================================================================
[INFO] All completed. Files in /home/cowcow1007/webase-deploy/nodes
==============      FISCO-BCOS      start...  ==============
try to start node0
try to start node1
 node1 start successfully
 node0 start successfully
==============      FISCO-BCOS      end...    ==============
================================================================
==============      WeBASE-Web     install... ==============
webase-web.zip编译包已经存在。是否重新下载？[y/n]:


======= script init success! =======
============== WeBASE-Node-Manager  start...  ==============
```

前端界面设计，如下两图所示：



点击按钮会进入相应函数的调用界面，如下图所示：



后端：

```
encrypt-type: # 0:standard, 1:guomi
  encrypt-type: 0

group-channel-connections-config:
  caCert: classpath:ca.crt
  sslCert: classpath:sdk.crt
  sslKey: classpath:sdk.key
  all-channel-connections:
    - group-id: 1 #group ID
      connections-str:
        - 127.0.0.1:20200 # node listen_ip:channel_listen_port

channel-service:
  group-id: 1 # The specified group to which the SDK connects
  agency-name: fisco # agency name

accounts:
  pem-file: 0xcdcce60801c0a2e6bb534322c32ae528b9dec8d2.pem
  p12-file: 0x98333491efac02f8ce109b0c499074d47e7779a6.p12
  password: 123456
```

```
fisco-bcos@fiscobcos-VirtualBox:~/fisco$ cd ../
fisco-bcos@fiscobcos-VirtualBox:~$ cd spring-boot-starter
fisco-bcos@fiscobcos-VirtualBox:~/spring-boot-starter$ ls
build         CONTRIBUTING.md  gradlew      README.md        src
build.gradle  doc              gradlew.bat  release_note.txt
Changelog.md  gradle           logs         settings.gradle
fisco-bcos@fiscobcos-VirtualBox:~/spring-boot-starter$ ./gradlew build
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could not be reused, use --status for details

Deprecated Gradle features were used in this build, making it incompatible with Gradle 6.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/5.6.2/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 1m 6s
3 actionable tasks: 3 executed
fisco-bcos@fiscobcos-VirtualBox:~/spring-boot-starter$ ./gradlew test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 6.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/5.6.2/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 52s
5 actionable tasks: 2 executed, 3 up-to-date
fisco-bcos@fiscobcos-VirtualBox:~/spring-boot-starter$
```

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

Compile solidity contract files to java contract files successfully!
fisco-bcos@fiscobcos-VirtualBox:~/fisco/console$
```

六、心得体会

本次大作业了解到了一款新出的区块链平台FISCO BOCOS，有优点、也有不足，在安装和使用过程中出现了很多问题。但在一次次试错中，通过查找官网和询问他人，我们更深刻地学习了区块链。

本次大作业提供了一个供应链金融场景，要求实现四个基本功能，也可以有一些自己的设计。一开始还不太懂得怎么去实现这些功能，在区块链课程学习中，对区块链以

及智能合约的应用有了更具体的理解，学会了使用信用这个评估标准。 区块链的核心是信用。虽然区块链是去中心化，但在很多金融方面的应用领域，常常利用银行等权威机构。在我的设计中，银行这个超级节点在资金、信用管理以及到期自动结算等方面发挥了重要作用。

对我们小组来说，这次大作业中困难的不仅链上的部分，还有如何用sdk写出后端、链端、前端，在前端后端部分有求助过做过的学长学姐。我们小组成员之前都没有学习过相关的内容，对用到的工具也不是很熟悉，因此常常不知道怎么处理，对我们来说还是非常困难的，如果时间能够再充沛一些，我们应该还能做得更好。除此之外，由于对整个平台的设计框架没有那么清晰，因此在这个实现的过程中，学习以及参考了许多别人的经验；总的来说，通过本次实验更深刻地学习了区块链的概念及其应用，但仍然需要深入学习。