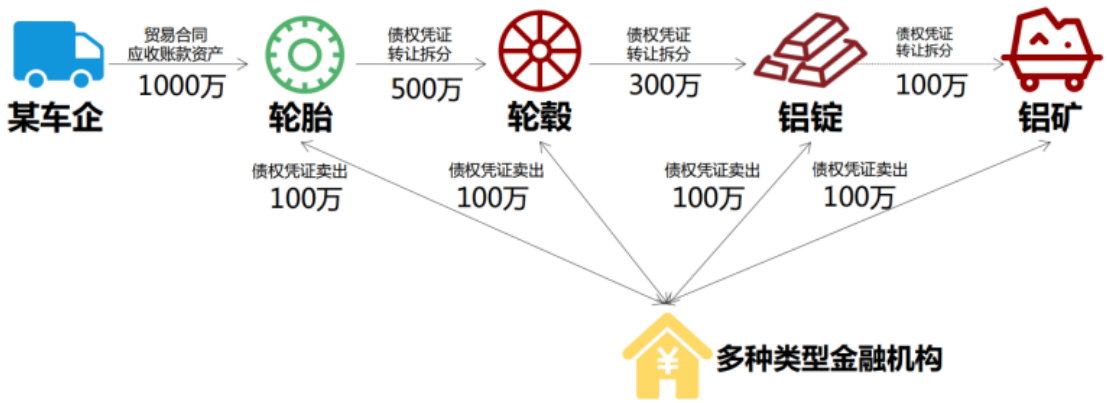
中山大学计算机学院本科生实验报告（2021年秋季学期）

课程名称：区块链技术原理

大作业项目：基于区块链的供应链金融平台

姓名	年级专业	学号
冼子婷	19级软件工程	18338072
廖雨轩	19级软件工程	18322043
胡文浩	19级软件工程	18346019

一、项目背景



- **传统供应链金融**： 某车企（宝马）因为其造车技术特别牛，消费者口碑好，所以其在同行业中占据绝对优势地位。因此，在金融机构（银行）对该车企的信用评级将很高，认为他有很大的风险承担的能力。在某次交易中，该车企从轮胎公司购买了一批轮胎，但由于资金暂时短缺向轮胎公司签订了 1000 万的应收账款单据，承诺 1 年后归还轮胎公司 1000 万。这个过程可以拉上金融机构例如银行来对这笔交易作见证，确认这笔交易的真实性。在接下里的几个月里，轮胎公司因为资金短缺需要融资，这个时候它可以凭借跟某车企签订的应收账款单据向金融结构借款，金融机构认可该车企（核心企业）的还款能力，因此愿意借款给轮胎公司。但是，这样的信任关系并不会往下游传递。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。
- **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。

二、方案设计

项目要求

- 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。

具体设计

系统流通资源

- 信用值（credit）：由于信用本身不可衡量，因此在系统中使用**信用值**可以度量的概念来表示企业的可信用程度。简单来说，信用值表示这个企业在规定有限时间内能够偿还的估计金额。信用值的使用需要经过收款方的同意，才能够生成一笔与信用值相关的账单。
- 资金（funding）：资金是一个企业或者银行能够在这个系统中自由使用的资源，其可以由另外的系统，将一定的法定货币如人民币、美金等转换为其中的等价的资金值。

系统角色

- 系统管理员（administrator）：对整个系统进行管理，控制整体信用值的分发以及监控企业之间的借贷情况。在实际情况中，系统管理员一般由政府等监管机构担任。考虑到这类监管机构拥有其他强制手段来进行系统中的交易控制，因此在系统中只给管理员提供一些更高级的查询功能，但是将不会参与或者干涉其中的交易情况。

```
struct Administrator {
    address addr;            // 管理员地址
    uint256 creditProvided; // 管理员发放的信用值数目
}
```

- 银行（bank）：为了简化概念和方便实现，在我们的系统中的金融机构仅使用银行作为代表。作为金融机构，银行可以对企业进行认证和信用值分发的操作

```
struct Bank {
    address addr;       // 地址
    string  name;       // 名称
    uint256 credit;     // 可发放信用值
    uint256 funding;    // 可用资金
}
```

- 企业（company）：企业可分为以下两种类型
  - 核心企业（core company）：核心企业在金融机构中有相当的信用程度，金融机构可认为其有能力进行还贷的能力，因此可以允许其发起的融资请求
  - 普通企业（normal company）：普通企业的信用程度仍未达到金融机构的认可标准，暂时不可向金融机构发起融资请求。若后续该企业得到金融机构的认可，也可以升级为核心企业。

```
struct Company {
    address addr;           // 地址
    string  name;           // 名称
    uint256 companyType;    // 企业类型
    uint256 credit;         // 信用值
    uint256 funding;        // 可用资金
}

uint256 companyTypeNormal = 0;  // 普通企业
uint256 companyTypeCore = 1;    // 核心企业
```

存储设计

Table.sol

FISCO-BCSO 中提供了 `Table.sol` 的合约，这个合约可以作为分布式数据库使用，它实现并提供了数据库的基本 CRUD 功能。因此直接在我们系统中直接引用这个合约，直接作为我们的存储系统。接下来只需要设计相对应的存储结构即可，无需额外部署数据库对地址信息、信用值等账户信息或者交易、账单等进行记录。

为了实现系统功能，使用其中的 TableFacotry 创建以下记录表

```
TableFactory tf = TableFactory(0x1001);
tf.createTable(bankTable, "1", "addr,addrStr,name,credit,funding");
tf.createTable(companyTable, "1",
"addr,addrStr,name,companyType,credit,funding");
tf.createTable(txTable, "1",
"txID,from,fromStr,to,toStr,amount,message,txType,txState,billID");
tf.createTable(billTable, "1",
"billID,from,fromStr,to,toStr,amount,createdDate,endDate,message,lock,billState,
billType");

// 当做索引
tf.createTable(roleTable, "addr", "role");
```

交易事件上链

合约中定义了以下 `event`， `event` 为区块链对合约操作进行上链的接口，使用 `emit` 进行调用即可让区块链节点对这个事件打包进入到区块中并广播和共识，这个事件记录到链上。

```
// 需要写入到区块链的事件
event Registration(address operatorAddr, address addr, string role);
event ProvideCredit(address operatorAddr, address addr, uint256 amount);
event ProvideFunding(address operatorAddr, address addr, uint256 amount);
event WithdrawCredit(address operatorAddr, address addr, uint256 amount);
event Financing(address operatorAddr, address bankAddr, bool useBill, string
message);
event ConfirmFinancing(address bankAddr, uint256 txID, bool accepted);
event Repay(address operatorAddr, address addr, uint256 amount);
event TransferBill(address operatorAddr, address from, address newTo, uint256
billID);
event TransferFunding(address operatorAddr, address to, uint256 amount);
```

交易与账单结构

交易

交易在系统中会对关键操作进行记录，可用于提供接口给监管机构进行操作查询。

```
struct Transaction {
    uint256 txID;           // 交易 ID
    address from;           // 付款人地址
    address to;             // 收款人地址
    uint256 amount;         // 交易总额
    string  message;        // 对该交易的一些额外备注信息

    uint256 txType;         // 交易类型
    uint256 txState;        // 交易状态
    int256 billID;          // 使用账单融资时所关联的 billID
}

uint256 txTypeNormal = 0;           // 正常交易
uint256 txTypeCreditFinacing = 1;   // 使用信用点融资
uint256 txTypeBillFinacing = 2;     // 使用账单融资


uint256 txStatePending = 0;     // 正在处理
uint256 txStateRefused = 1;     // 拒绝本次交易
uint256 txStateAccepted = 2;    // 接收本次交易
```

账单

账单仅限于被接受的交易，相当于交易双方签订成功的交易，**账单才表示双方之间存在借账关系。**

```
struct Bill {
    uint256 billID;         // 账单 ID
    address from;           // 付款人地址
    address to;             // 收款人地址
    uint256 amount;         // 账单额
    string  createdDate;    // 创建日期
    string  endDate;        // 还款日期
    string  message;        // 账单备注信息

    uint256 lock;           // 是否锁定，表示该账单是否用于进行转移操作
```

```
    uint256 state;          // 账单状态
    uint256 billType;       // 账单类型，只有付款人为核心企业才能用于融资
}

uint256 billStateUnpaid = 0;    // 账单未还
uint256 billStatePaid = 1;      // 账单已还

uint256 billUnlocked = 0;
uint256 billLocked = 1;

uint256 billTypeNormal = 0; // 普通账单
uint256 billTypeCore =1;    // 核心企业为付款人的账单
```

核心代码+数据流图

功能一：实现采购商品—签发应收账款交易上链

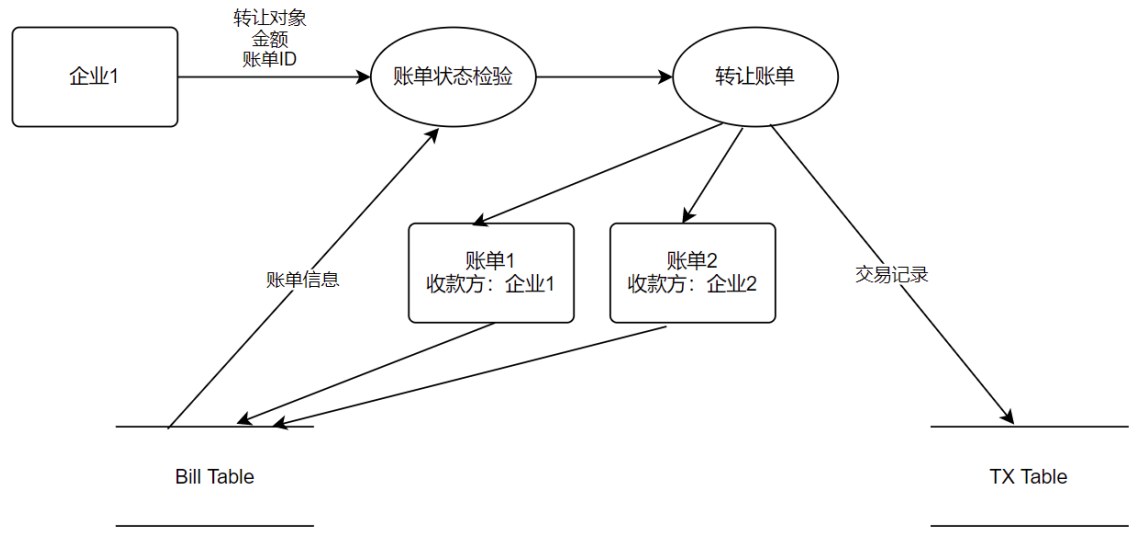
商品采购直接进行签发账单即可。签发应收账单时会生成交易和对应的账单，核心企业进行签发时会消耗相应的信用值，以此生成的账单将能够用于向银行进行融资操作，普通企业虽然也能够签发账单，但是由于其在银行中的信用不够，因此这种账单不能够用于融资

```
// 赊账，签发应收账单
function transferFunding(address operatorAddr, address addr, uint256 amount,
string message, string createdDate, string endDate) public {
    company = getCompany(operatorAddr);
    if(company.companyType == companyTypeCore) {
        require(company.addr == operatorAddr, "transferFunding: this addr is
another company");
        require(company.credit >= amount, "transferFunding: no enough credit");
        updateCompany(operatorAddr, amount, false, 0, false);
        insertTx(operatorAddr, addr, amount, message, txTypeCreditFinacing,
txStateAccepted, 0);
        insertBill(operatorAddr, addr, amount, createdDate, endDate, message,
billTypeCore);
    } else {
        insertTx(operatorAddr, addr, amount, message, txTypeNormal,
txStateAccepted, 0);
        insertBill(operatorAddr, addr, amount, createdDate, endDate, message,
billTypeNormal);
    }
    emit TransferFunding(operatorAddr, addr, amount);
}
```

功能二：实现应收账款的转让上链

账单转让首先会将原有账单变为已还的状态并锁定。随后将其中的交易额进行对应的划分，并生成两份新的账单，这两份账单的收款方为原账单持有人和对应转让的目标。转让账单的操作所生成的两个账单将会保留原账单的性质，即原账单若为核心企业签发，那么新生成账单也可以用于向银行融资，若原账单为普通企业签发则无法进行融资。

```
    // 账单转移
function transferBill(address operatorAddr, address to, uint256 amount, string
message, uint billID, string createdDate, string endDate) public {
    bill = getBillByID(billID);
    require(bill.billState == billStateUnpaid, "transferBill: this bill has been
paid");
    require(bill.lock == billUnlocked, "transferBill: this bill has been
locked");
    require(bill.to == operatorAddr, "transferBill: cannot operator other's
bill");
    require(bill.amount >= amount, "transferBillL: no enough amount");
    insertTx(operatorAddr, to, amount, message, txTypeNormal, txStateAccepted,
billID);
    updateBill(billID, billLocked, billStatePaid);
    insertBill(bill.from, operatorAddr, bill.amount - amount, createdDate,
endDate, "transfer bill", bill.billType);
    insertBill(bill.from, to, amount, createdDate, endDate, "transfer bill",
bill.billType);
    emit TransferBill(operatorAddr, operatorAddr, to, billID);
}
```



功能三：利用应收账款向银行融资上链

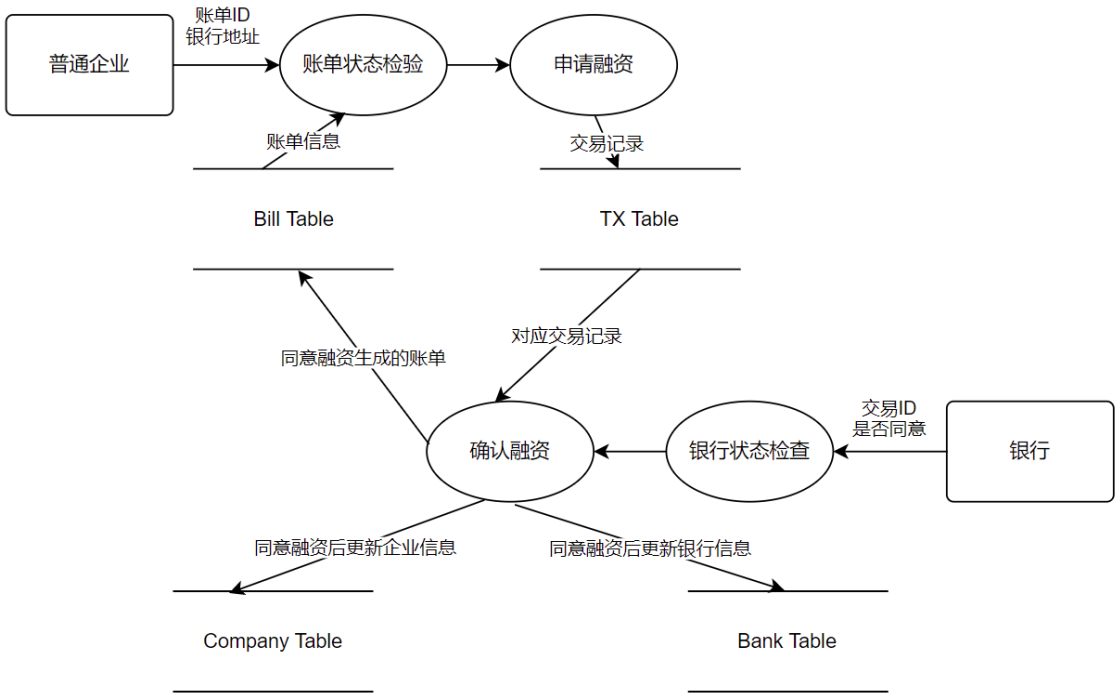
使用账单融资时，银行只能接受由核心企业签发的应收账单。进行融资时首先将对应账单进行锁定，此时会生成一个企业向银行申请融资的交易记录，银行将会对这个交易进行判断，选择同意或者拒绝这个交易。当银行同意这个交易后，将会把原账单设置为已还款，并且生成新的账单，付款方为原账单的付款方。

```
// 融资
function financing(address operatorAddr, address bankAddr, uint256 amount,
string message, bool useBill, uint256 billID) public {
    company = getCompany(operatorAddr);
    string memory targetRole = getRole(bankAddr);
    require(equal(targetRole, "bank"), "financing: target sholud be a bank");
    // 核心企业使用信用点融资
    if(!useBill) {
        require(company.credit >= amount, "financing: no enough credit");
        // 扣留信用值，产生未确认交易
        updateCompany(operatorAddr, amount, false, 0, false);
```

```
        insertTx(operatorAddr, bankAddr, amount, message, txTypeCreditFinacing,
txStatePending, 0);
    } else {
    // 普通企业使用账单融资
        bill = getBillByID(billID);
        // 非核心企业为付款的账单不能用作融资
        require(bill.billType == billTypeCore, "financing: this bill is not from
a core company");
        require(bill.lock == billUnlocked, "financing: this bill has been
locked");
        require(bill.billState == billStateUnpaid, "financing: this bill has
been paid");
        updateBill(billID, billLocked, billStateUnpaid);
        insertTx(operatorAddr, bankAddr, bill.amount, message,
txTypeBillFinacing, txStatePending, billID);
    }
    emit Financing(operatorAddr, bankAddr, useBill, message);
}
```

```
// 融资确认
function confirmFinancing(address bankAddr, uint256 txID, bool accepted, string
createdDate, string endDate) public {
    transaction = getTxByID(txID);
    bank = getBank(bankAddr);
    require(transaction.txState == txStatePending, "ConfirmFinancing:
transaction has been confirmed");
    require(bankAddr == transaction.to, "ConfirmFinancing: cannot confirm
other's finacing");
    if(transaction.txType == txTypeCreditFinacing){
        if(accepted) {
            require(bank.funding >= transaction.amount, "ConfirmFinancing: no
enough funding");
            updateBank(transaction.to, 0, false, transaction.amount, false);
            updateCompany(transaction.from, 0, false, transaction.amount, true);
            updateTx(transaction, txStateAccepted, transaction.billID);
            insertBill(transaction.from, transaction.to, transaction.amount,
createdDate, endDate, transaction.message, billTypeCore);
        } else {
            // 归还锁定的信用点
            updateCompany(transaction.from, transaction.amount, true, 0, false);
            updateTx(transaction, txStateRefused, transaction.billID);
        }
    } else if(transaction.txType == txTypeBillFinacing) {
        if(accepted) {
            require(bank.funding >= transaction.amount, "ConfirmFinancing: no
enough funding");
            bill = getBillByID(transaction.billID);
            updateBank(transaction.to, 0, false, transaction.amount, false);
            updateCompany(transaction.from, 0, false, transaction.amount, true);
            updateTx(transaction, txStateAccepted, transaction.billID);
            emit TransferBill(bankAddr, bill.from, transaction.to,
transaction.billID);
            // 结束申请贷款的账单，并新建账单
            updateBill(transaction.billID, billLocked, billStatePaid);
            insertBill(bill.from, transaction.to, transaction.amount,
createdDate, endDate, transaction.message, billTypeCore);
        } else {
```

```
            // 解锁申请的账单
            updateBill(transaction.billID, billUnlocked, billStateUnpaid);
            updateTx(transaction, txStateRefused, transaction.billID);
        }
    } else {
        require(false, "ConfirmFinancing: error transaction type");
    }
    emit ConfirmFinancing(bankAddr, txID, accepted);
}
```

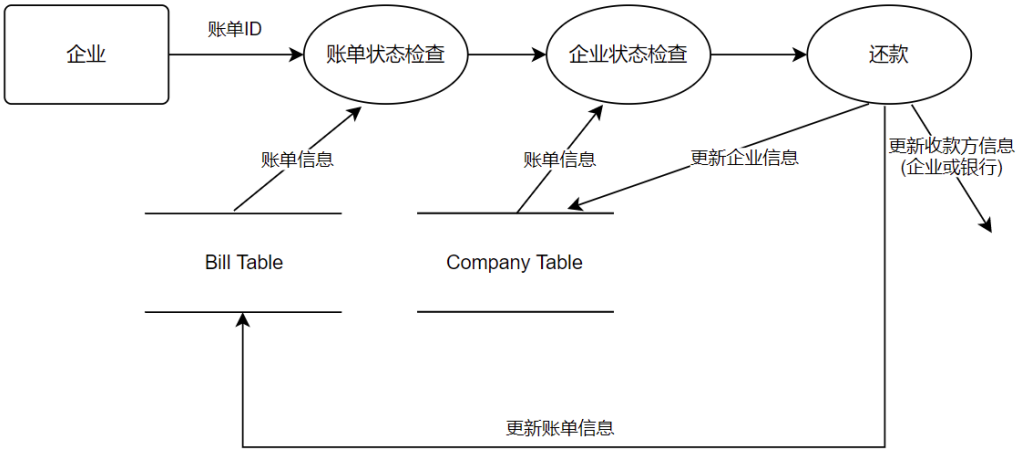


功能四：应收账款支付结算上链

通过还款渠道，企业能够对相应的账单进行结算。此功能将会检查当前企业的资金是否足以支付这一个账单，若资金充足的则能够将相对应的资金量转移给收款方，并将账单更新为已还款。

```
// 还款或者还融资
function repay(address operatorAddr, uint256 billID) public {
    company = getCompany(operatorAddr);
    bill = getBillByID(billID);
    require(operatorAddr == bill.from, "repay: cannot repay other's bill");
    require(bill.billState == billStateUnpaid, "repay: bill has been repaied");
    require(company.funding >= bill.amount, "repay: no enough funding");
    string memory toRole = getRole(bill.to);
    if(equal(toRole, "bank")) {
        // 还融资(只有核心银行才会还融资，普通银行使用账单融资不需要还)
        updateCompany(operatorAddr, bill.amount, true, bill.amount, false);
        updateBank(bill.to, 0, false, bill.amount, true);
    } else {
        // 核心企业恢复信用值，普通企业不需要
        if(company.companyType == companyTypeNormal) {
            updateCompany(operatorAddr, 0, false, bill.amount, false);
        } else {
            updateCompany(operatorAddr, bill.amount, true, bill.amount, false);
        }
        updateCompany(bill.to, 0, false, bill.amount, true);
    }
```

```
    insertTx(operatorAddr, bill.to, bill.amount, "还账", txTypeRepayment,
txStateAccepted, bill.billID);
    updateBill(billID, billLocked, billStatePaid);
    emit Repay(operatorAddr, bill.to, bill.amount);
}
```

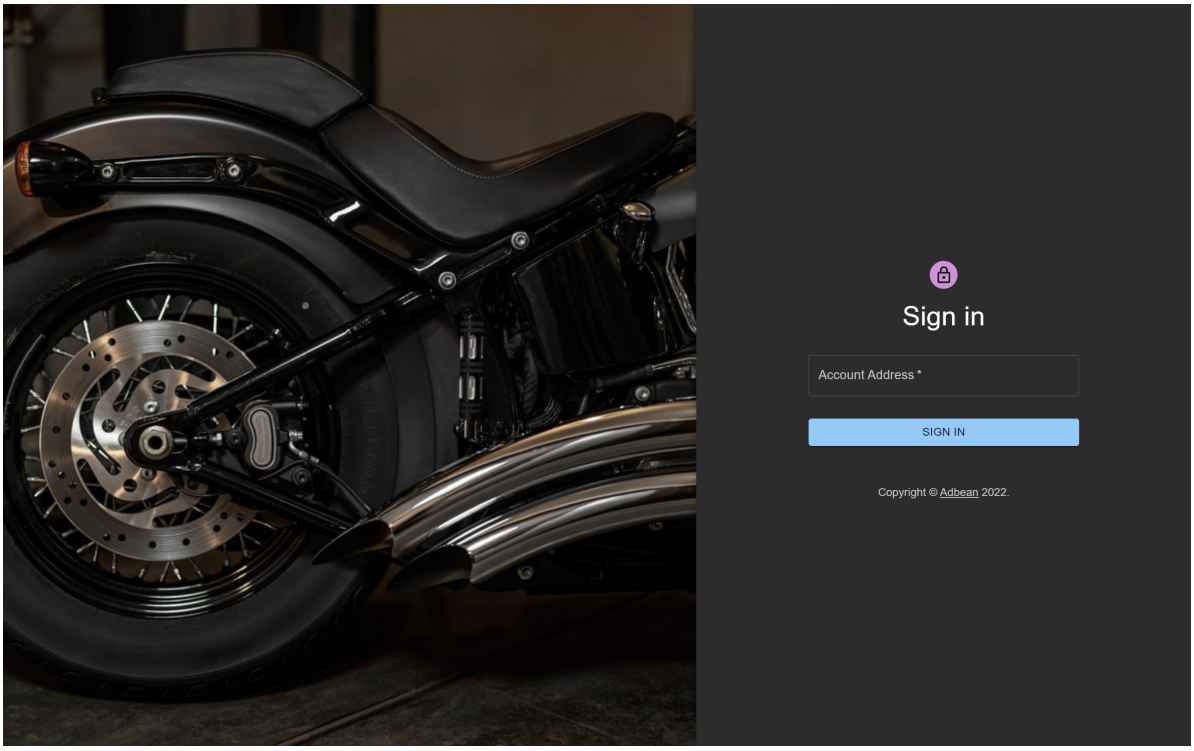


加分项说明

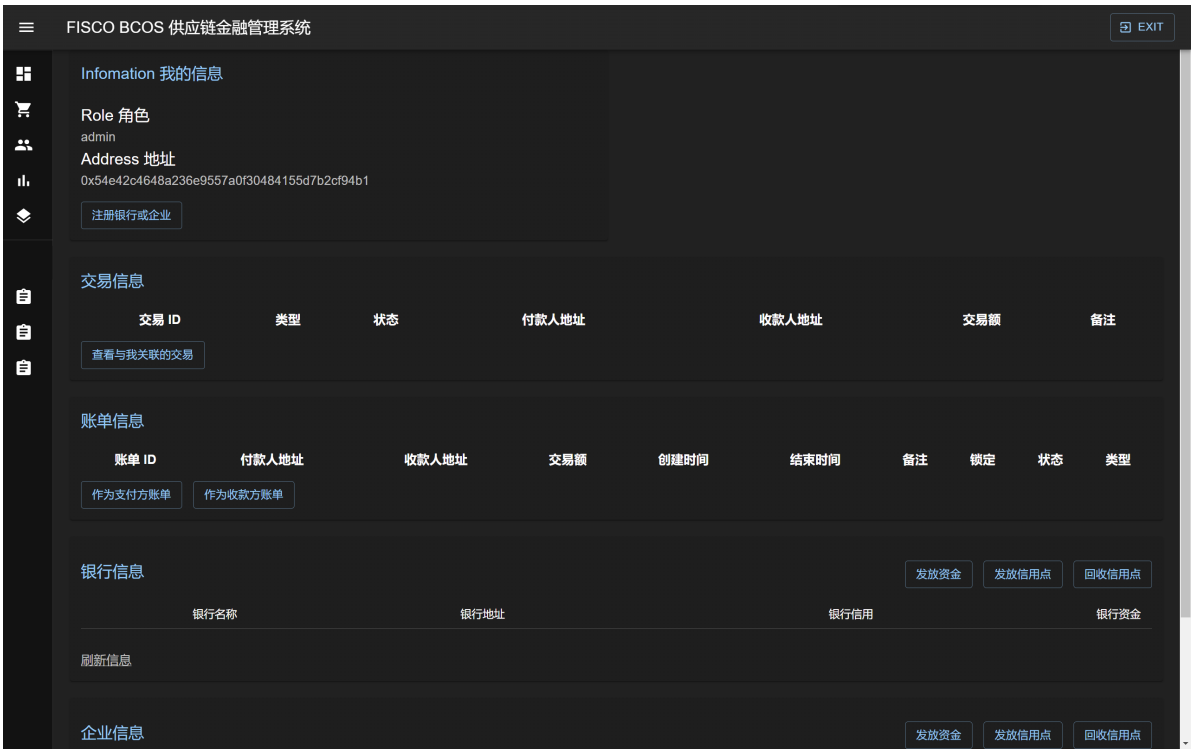
- 前端界面：使用 react 框架编写了一套简单易用且美观的用户界面。通过几个简单步骤即可完成转账、赊账和融资等操作
- 管理员：系统中添加了管理员的角色，令整个系统的运作能够被管理员监控。在系统中虽然没有实现一些具体的功能，但是这个系统设计可以对管理员功能进行扩展，为其添加更多的监管相关的功能。

界面展示

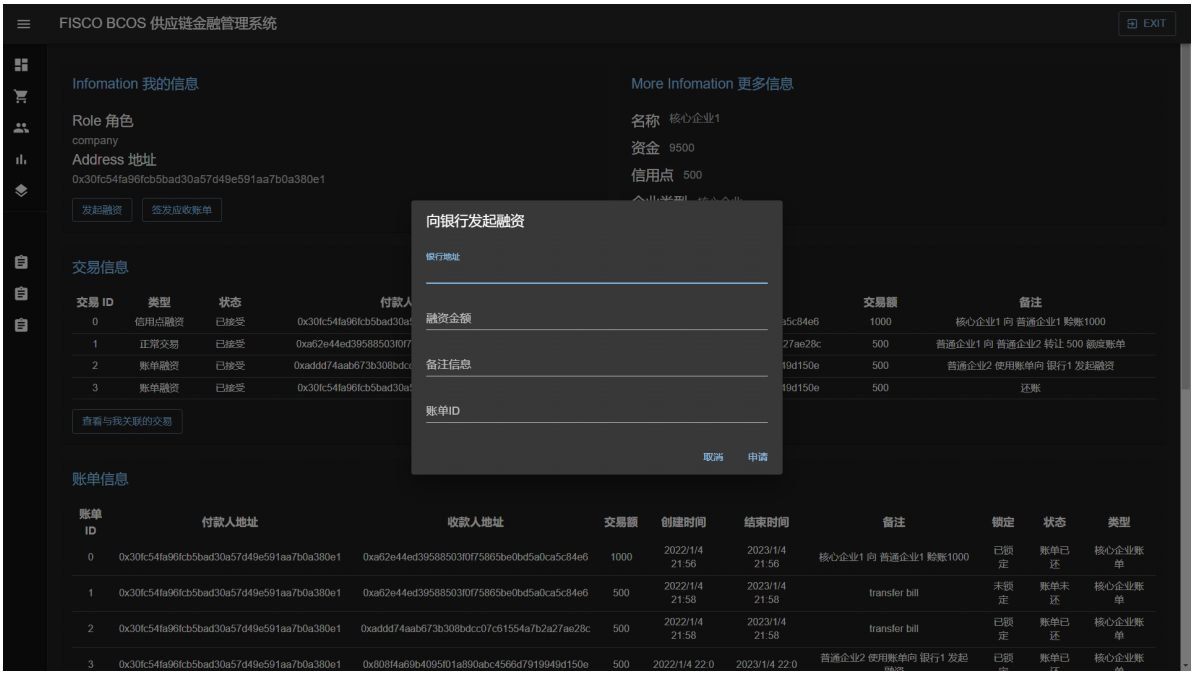
登录界面



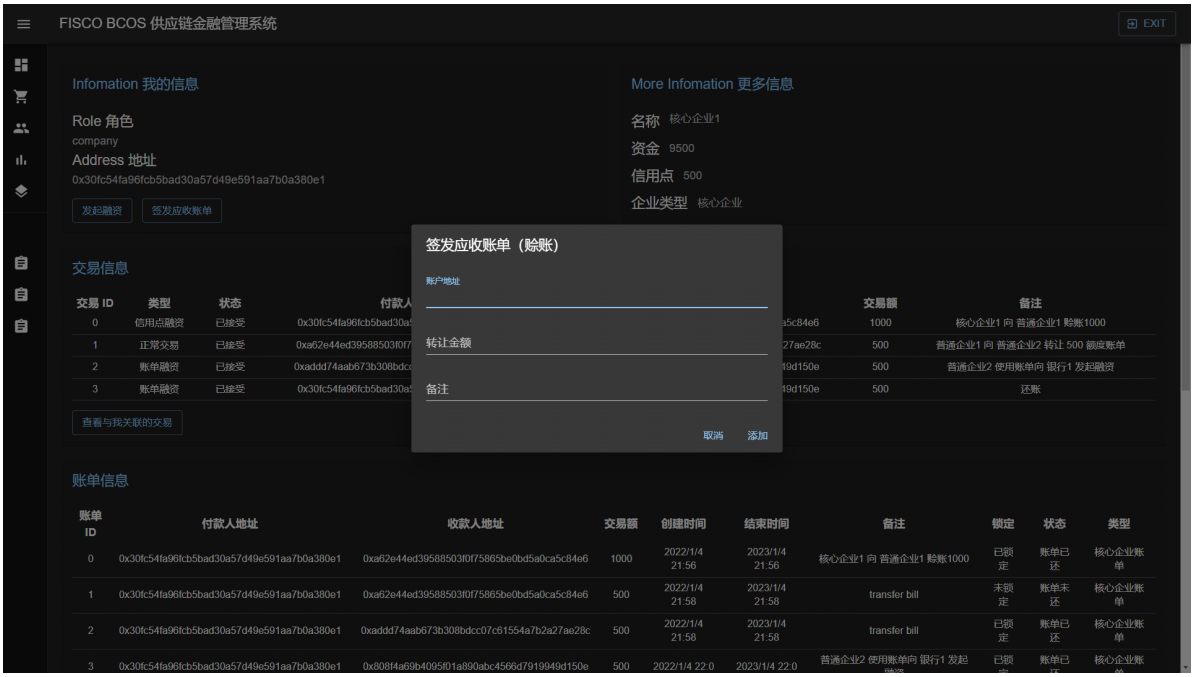
用户主界面



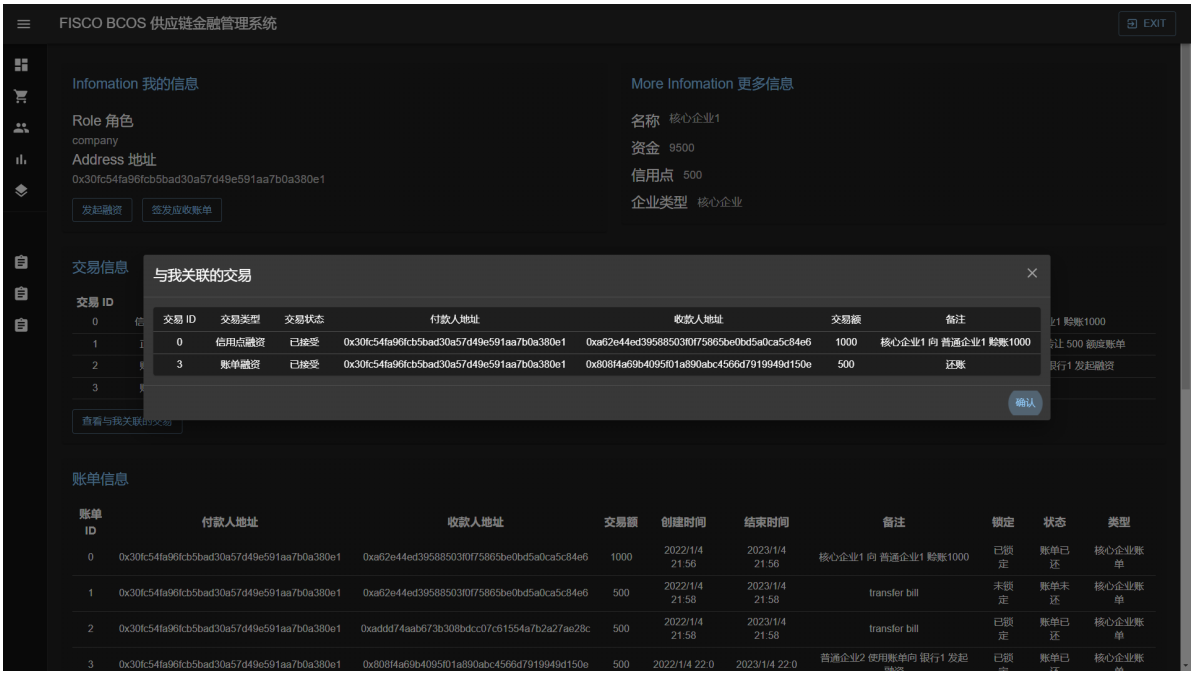
进行融资



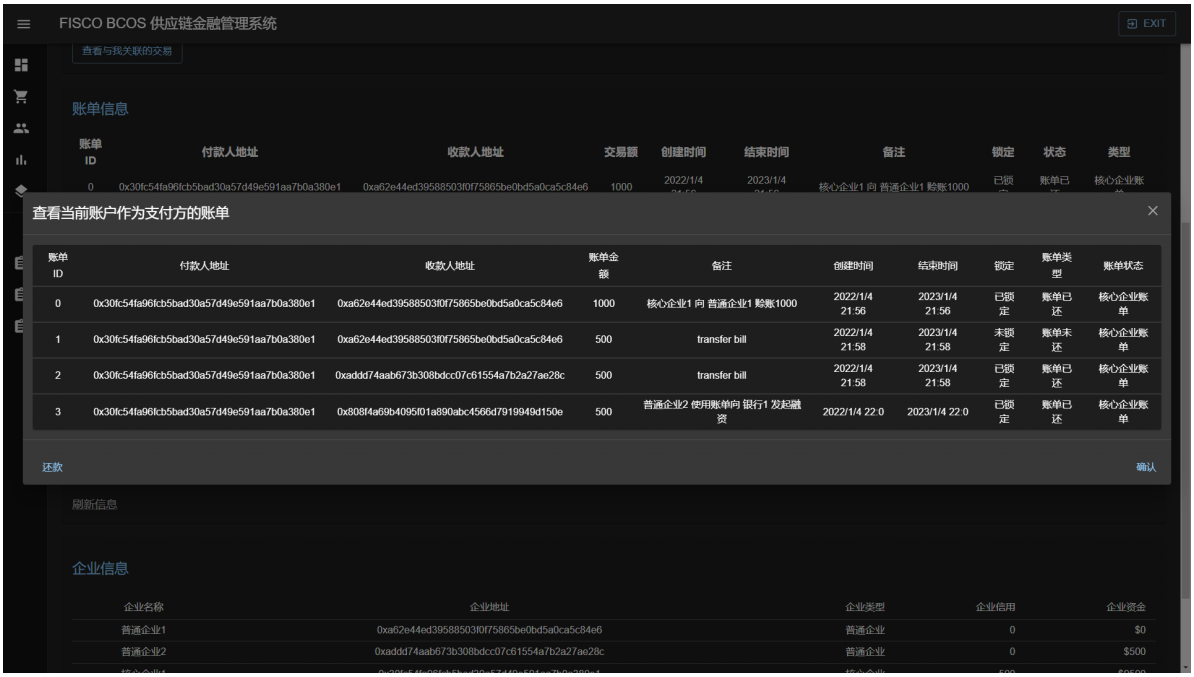
签发账单



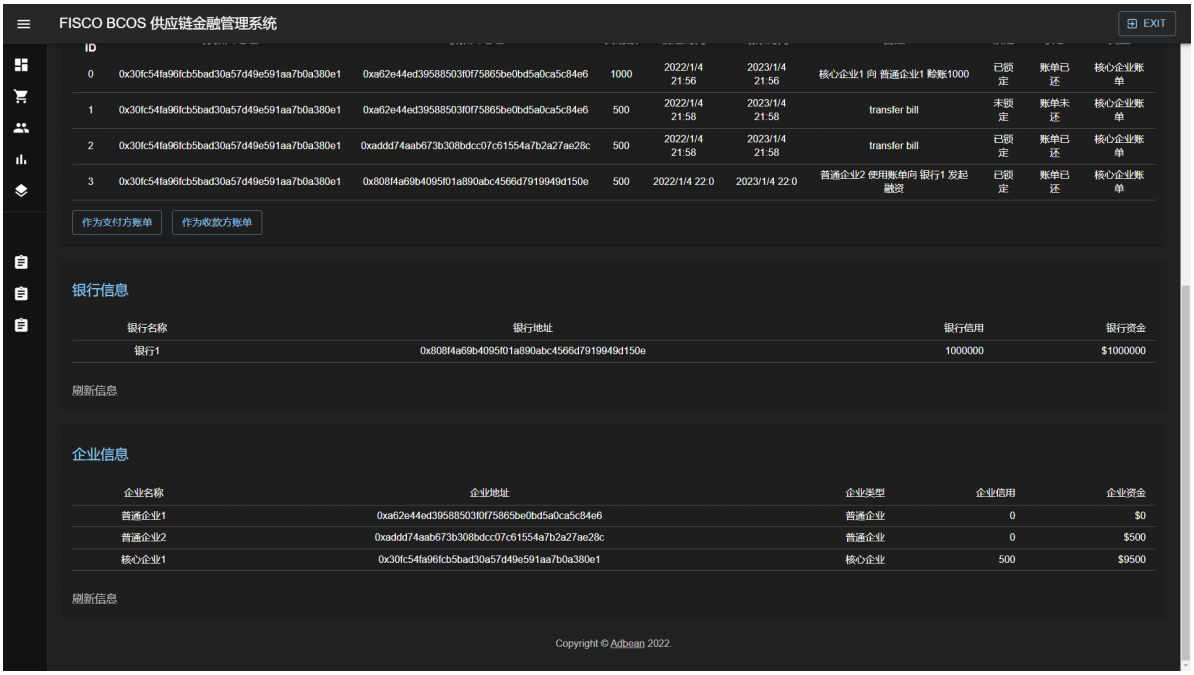
关联交易



自己为支付方的账单



银行与企业



项目心得

冼子婷：这次的大作业，让我第一次接触到了金融和计算机融合的领域。加入了区块链后的供应链金融，相比于传统的供应链金融，解决了信息孤岛的问题，降低了融资成本。同时，区块链的分布式、难篡改的特点，还保证供应链金融上下游信信任穿透,信息共享。经过这次的大作业，我对区块链有了更进一步的认识。

胡文浩：在金融交易中，由于其中角色的层级不同而导致部分角色在其中会处于劣势地位。通过在金融系统中引入区块链，我们能够将原本无法衡量的一些属性进行量化，通过区块链网络来从加速区域中所有成员的共识，将各个成员的信用组成一条链，从而促进资金的流通以及周转。区块链技术的发展大家有目共睹，相信未来会有越来越多的区块链应用落地到各个领域并发挥它们的重要作用。

廖雨轩：通过本次项目，让我对联盟链有了一定的认识，以及智能合约在联盟链中的重要性。相较于传统的区块链，联盟链通过注册许可预选结点，构成共同体形成联盟，并且通过指定相应的规则，使得联盟链的交易量大大提升，远大于公有链的吞吐量，在金融行业中有着广泛的前景。在实现项目时，也感受到了联盟链与区块链的不同，尤其在效率上，FISCO 采用并行计算的 PBFT 和标准 RAFT，提高了共识

效率的同时，也大大加快了出块速度，智能合约也更加容易部署等等。