项目设计说明及功能测试文档

18340097 李岩
18340100 李泽桐
18340105 廖家源

一、项目设计说明

1. 项目背景

> 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，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。

基于项目的背景，目标是构建基于区块链的供应链金融平台来解决上述提出的问题。这个平台应该提供的基本功能有

1. 实现采购商品——签发应收账款交易上链
2. 实现应收账款的转让上链
3. 利用应收账款向银行融资上链
4. 应收账款支付结算上链

2. 项目设计

针对上述的项目背景，采用 FISCO-BCOS 平台的 CRUD 接口，使用 solidity 语言设计智能合约 SupplyChain 来满足功能需求。

本设计计划通过维护两个 Table 来实现事务逻辑，一个是账户表 Account，另外一个是账单表 Bill。

对于**账户表 Account**，其用来存储在平台上运作的企业和金融机构的账户。表中的记录格式为

$$(id,\ role,\ credit)$$

id 是账户的地址，即区块链上的账户的 address，唯一标识一个企业或金融机构；role 是此账户在平台上的角色，取值为企业 enterprise 或 金融机构 institution，用以区分账户的身份；credit 是此账户代表的企业的信用分（本设计中不考虑金融机构的信用分，统一为 100），用来让金融机构作为第三方来评判其是否有能力签发账款单据给其他公司。

对于**账单表 Bill**，其用来存储在平台上各种交易产生的账款单据。表中的记录格式为

$$(lender,\ borrower,\ witness,\ amount,\ timestamp,\ deadline)$$

lender 是被借款人的账户地址，borrower 是借款人的账户地址，witness 是见证人的账户地址，amount 是金额，timestamp 是账单产生的时间戳，deadline 是账单被支付的最后期限。

基于上述两个表，定义此平台的交易规则：

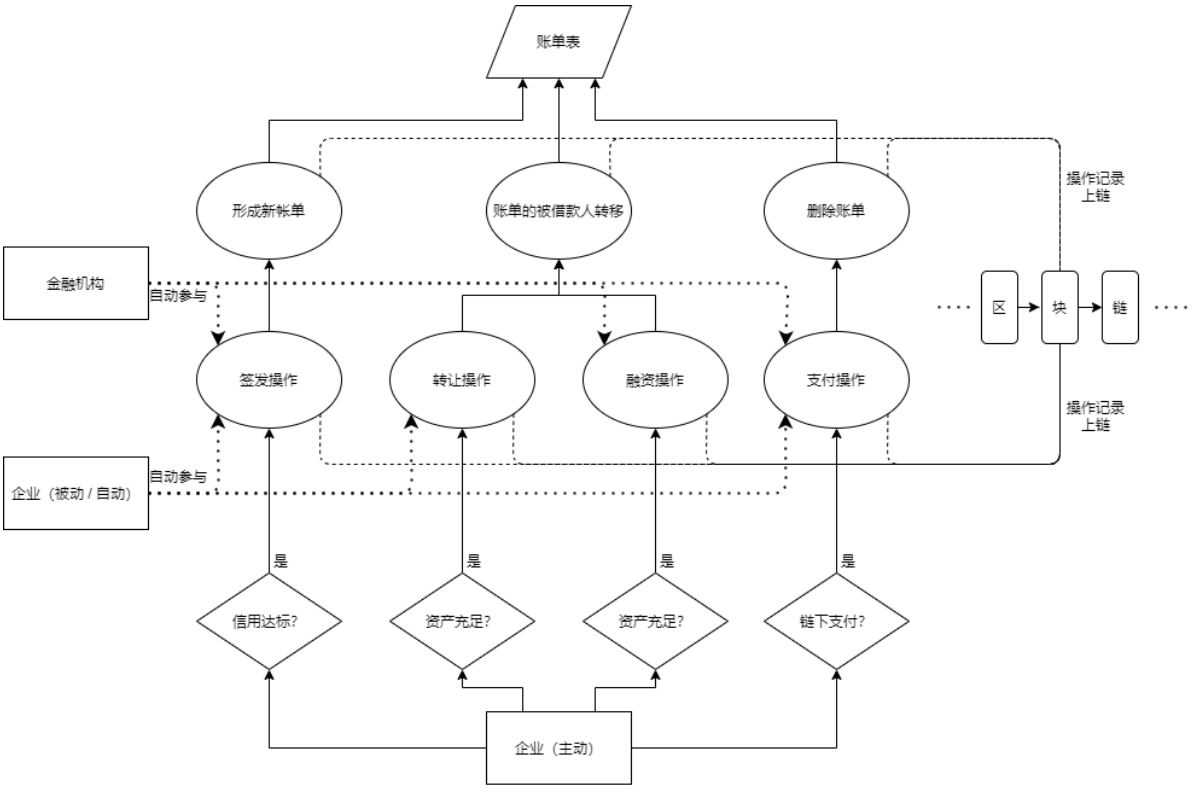
1. 由平台管理员搭建平台，亦即创建表 Account 和 Bill
2. 由平台管理员管理账户的加入与退出，但其无权影响各方的信用分评价
3. 只有信用分大于等于 60 的企业有资格向其他企业签发收款单据并得到金融机构的见证
4. 只有本企业的**资产（仅指平台上的账款单据的累加）**充足才能够发起账单的转让
5. 只有本企业的资产充足才有资格向金融机构融资
6. 企业链下进行账款的支付后才能够在链上删除相关账单

其中，本次 demo 设计尚未完全第 6 点。第 6 点的本意是指企业支付账款后，由其他待收账款的企业和金融机构确认后才能删除账单记录，但是在本次设计中，仅实现了删除账单记录，尚未实现其中的确认操作。

在编写智能合约时，通过一些自动的条件判断，来决定一些操作的发生与否，如

1. 企业请求签发账款单据操作，智能合约自动判断其信用达标与否。不达标，直接拒绝；否则相关企业或金融机构自动参与进来
2. 企业请求转让账款单据操作，智能合约自动判断其资产充足与否。不充足，直接拒绝；否则相关企业自动参与进来
3. 企业请求融资操作，智能合约自动判断其资产充足与否。不充足，直接拒绝；否则相关金融机构自动参与进来
4. 企业请求支付操作，智能合约让收取账款的账户判断是否已收到款。未收到，账单不予删除；否则，正常删除。如果企业是在期限内完成支付，则信用还会加分，否则，信用减分

其中，第 4 点与前述相同，尚未实现。具体的设计流程图如下



通过上述的流程，可以让所有的操作由一方主动发起后自动完成，不满足条件的操作均会被驳回。回到项目的背景可以发现，这其中的设计，为了解决高信用企业的信用在供应链中传递，采取的办法就是形成大家都有目共睹的有效账单，形成的规则由平台参与者制定，所有人在平台运行当中，自动地受智能合约约束。这样的有效账单像是企业之间的通行货币，并且还能有信用凭证的效力。而账单的这些效力之所以能够被所有参与者相信和认同，是因为大家的交易都是公开透明的，账单的来源是可溯源的，且是不可篡改的，而这，就是区块链所带的效果，也是这个问题得以解决的关键。如若不引入区块链，则账单可能仍有上述的效力，但账单的真假是有待考察的，这样的考察在现实生活中会带来不小的时间成本和金钱成本。但现在，区块链巧妙地解决了这个信任危机。

3. 实现的 demo 代码

首先引入合约的版本号，如下

```
pragma solidity ^0.4.25;
```

由于此设计使用 FISCO-BCOS 的 CRUD 接口，因此需要引入 CRUD 接口文件，如下

```
import "./Table.sol";
```

其次是状态变量，这里只设置了一个变量，用于记录部署智能合约的管理员的账户地址，如下

```
address private god;
```

再者是 DEBUG 使用的 event 事件，如下（由于在官方文档中找不到 2.7 版本控制台查看 event logs 的命令，因此此部分近乎于作废）

```
event signEvent(address borrower, address lender, address witness, int amount,
int timestamp, int deadline);
event transferEvent(address sender, address receiver, int amount);
event financingEvent(address enterprise, address institution, int amount);
event payEvent(address borrower, address lender, address witness, int amount,
int timestamp, int deadline);
```

紧接着是构建函数和修饰器，修饰器用来保证一些如表的创建和账户表的账户加入删除只能由管理员完成，如下

```
modifier onlyGod
{
    require(god == msg.sender, "Auth: only god is authorized.");
    _;
}

constructor () public
{
    god = msg.sender;
}
```

最后是 SupplyChain 智能合约的整体框架，包含一些辅助函数，对表 Account 的增删改查函数，对表 Bill 的增删改查函数，以及本设计的核心功能函数签发操作 sign、转让操作 transfer、融资操作 financing、支付操作 pay 等，如下

```
// 函数 辅助 字符串比较
function stringCompare(string a, string b) private pure returns(bool)

// 函数 辅助 编码
function encode(uint8 num) private pure returns(byte)

// 函数 辅助 地址转换为字符串
function addressToString(address addr) private pure returns(string)

// 函数 账户表 创建操作
function createAccountTable() public onlyGod returns(int);
```

```
// 函数 账户表 插入操作
function insertAccountTable(address id, string role, int credit) public onlyGod
returns(int);

// 函数 账户表 删除操作
function removeAccountTable(address id) public onlyGod returns(int);

// 函数 账户表 更新操作
function updateAccountTable(address id, int new_credit) private returns(int);

// 函数 账户表 查询操作
function selectAccountTable(address id) private returns(Entries);

// 函数 账单表 创建操作
function createBillTable() public onlyGod returns(int);

// 函数 账单表 插入操作 有重载
function insertBillTable(address lender, address borrower, address witness, int
amount, int timestamp, int deadline) private returns(int);

// 函数 账单表 插入操作 有重载
function insertBillTable(Entry entry, address lender, int amount) private
returns(int);

// 函数 账单表 删除操作 有重载
function removeBillTable(address lender, address borrower, address witness, int
timestamp, int deadline) private returns(int);

// 函数 账单表 删除操作 有重载
function removeBillTable(Entry entry) private returns(int);

// 函数 账单表 更新操作 有重载 后续代码废用了此功能
// function updateBillTable(address lender, address borrower, address witness,
int timestamp, int deadline, address new_lender, int new_amount) private
returns(int);

// 函数 账单表 更新操作 有重载 后续代码废用了此功能
// function updateBillTable(Entry entry, address new_lender, int new_amount)
private returns(int);

// 函数 账单表 查询操作 有重载
function selectBillTable(address lender) private returns(Entries);

// 函数 账单表 查询操作 有重载
function selectBillTable(address lender, address borrower, address witness, int
timestamp, int deadline) private returns(Entries);

// 函数 签发操作
function sign(address lender, address witness, int amount, int duration) public
returns(int);

// 函数 转让操作
function transfer(address receiver, int total_amount) public returns(int);

// 函数 融资操作
function financing(address institution, int total_amount) public returns(int);
```

```
// 函数 - 支付操作
function pay(address lender, address witness, int timestamp, int deadline)
public returns(int);
```

关键的实现逻辑在于最后核心的四个函数，即 sign、transfer、financing、pay。

sign 函数，即签发操作，当企业需要签发收款单据时调用，参数为 [被借款人账户地址，见证人账户地址，金额，借款时长]，在函数中，需要得到系统时间形成当前账单的时间戳，并计算得到最后的还款期限，写入 Bill 表中，具体代码如下

```
function sign(address lender, address witness, int amount, int duration) public
returns(int)
{
    // 考虑 borrower 和 lender 和 witness 的存在性
    // 考虑 borrower 和 lender 是否为企业，witness 是否为金融机构
    // 考虑 borrowe 信用分足够与否
    Entries entries_borrower = selectAccountTable(msg.sender);
    Entries entries_lender = selectAccountTable(lender);
    Entries entries_witness = selectAccountTable(witness);
    if(entries_borrower.size() == 0)
    {
        return -1; // borrower 不存在
    }
    if(entries_lender.size() == 0)
    {
        return -2; // lender 不存在
    }
    if(entries_witness.size() == 0)
    {
        return -3; // witness 不存在
    }
    if(!stringCompare(entries_borrower.get(0).getString("role"), "enterprise"))
    {
        return -4; // borrower 非企业
    }
    if(!stringCompare(entries_lender.get(0).getString("role"), "enterprise"))
    {
        return -5; // lender 非企业
    }
    if(!stringCompare(entries_witness.get(0).getString("role"), "institution"))
    {
        return -6; // witness 非金融机构
    }
    if(entries_borrower.get(0).getInt("credit") < 60)
    {
        return -7; // borower 信用分不足
    }

    //
    int timestamp = int(now);
    int deadline = timestamp + duration * 1 days;
    int value = insertBillTable(lender, msg.sender, witness, amount, timestamp,
deadline);
    emit signEvent(msg.sender, lender, witness, amount, timestamp, deadline);
    return 0;
}
```

其中函数的开头会判断各方参与者的有效性，以及发起者的信用是否达标。其中借款时长以日计，即 duration 的单位为天。

transfer 函数，即转让操作，当企业需要转让账款单据时调用，参数为 [接收者账户地址，金额]，也就是说，企业可以根据具体的金额来转让，一张账单可以变成两张，只要达到金额即可，无需硬性地被要求每次转让都要整张账单，具体的代码如下

```
function transfer(address receiver, int total_amount) public returns(int)
{
    // 考虑 sender 和 receiver 的存在性
    // 考虑 sender 和 receiver 是否为企业
    Entries entries_sender = selectAccountTable(msg.sender);
    Entries entries_receiver = selectAccountTable(receiver);
    if(entries_sender.size() == 0)
    {
        return -1; // sender 不存在
    }
    if(entries_receiver.size() == 0)
    {
        return -2; // receiver 不存在
    }
    if(!stringCompare(entries_sender.get(0).getString("role"), "enterprise"))
    {
        return -3; // sender 非企业
    }
    if(!stringCompare(entries_receiver.get(0).getString("role"), "enterprise"))
    {
        return -4; // receiver 非企业
    }

    //
    Entries entries = selectBillTable(msg.sender);
    int index = 0;
    int asset = 0;
    for(int i = 0; i < entries.size(); ++i)
    {
        asset += entries.get(i).getInt("amount");
        if(asset >= total_amount)
        {
            index = i + 1;
            break;
        }
    }
    if(asset < total_amount)
    {
        return -5; // 没有足够资产转让
    }
    int amount;
    for(i = 0; i < index - 1; ++i)
    {
        amount = entries.get(i).getInt("amount");
        removeBillTable(entries.get(i));
        insertBillTable(entries.get(i), receiver, amount);
    }
    amount = entries.get(index - 1).getInt("amount");
    removeBillTable(entries.get(index - 1));
    if(asset == total_amount)
```

```
    {
        insertBillTable(entries.get(index - 1), receiver, amount);
    }
    else
    {
        insertBillTable(entries.get(index - 1), receiver, amount - asset +
total_amount);
        insertBillTable(entries.get(index - 1), msg.sender, asset -
total_amount);
    }
    emit transferEvent(msg.sender, receiver, total_amount);
    return 0;
}
```

同样的，函数开头仍然需要判断参与双方的身份，而后对发起操作的企业核查资产，决定如何转让账单。

financing 函数，即融资操作，当企业需要融资时调用，参数为 [金融机构账户地址，金额]，这个函数实质上与 transfer 非常类似，只是账单的被借款人不同而已，因此不再赘述，代码如下

```
function financing(address institution, int total_amount) public returns(int)
{
    // 考虑 enterprise 和 institution 的存在性
    // 考虑 enterprise 是否为企业，institution 是否为金融机构
    Entries entries_enterprise = selectAccountTable(msg.sender);
    Entries entries_institution = selectAccountTable(institution);
    if(entries_enterprise.size() == 0)
    {
        return -1; // enterprise 不存在
    }
    if(entries_institution.size() == 0)
    {
        return -2; // institution 不存在
    }
    if(!stringCompare(entries_enterprise.get(0).getString("role"),
"enterprise"))
    {
        return -3; // enterprise 非企业
    }
    if(!stringCompare(entries_institution.get(0).getString("role"),
"institution"))
    {
        return -4; // institution 非金融机构
    }

    //
    Entries entries = selectBillTable(msg.sender);
    int index = 0;
    int asset = 0;
    for(int i = 0; i < entries.size(); ++i)
    {
        asset += entries.get(i).getInt("amount");
        if(asset >= total_amount)
        {
            index = i + 1;
            break;
        }
```

```
    }
    if(asset < total_amount)
    {
        return -5; // 没有足够资产融资
    }
    int amount;
    for(i = 0; i < index - 1; ++i)
    {
        amount = entries.get(i).getInt("amount");
        removeBillTable(entries.get(i));
        insertBillTable(entries.get(i), institution, amount);
    }
    amount = entries.get(index - 1).getInt("amount");
    removeBillTable(entries.get(index - 1));
    if(asset == total_amount)
    {
        insertBillTable(entries.get(index - 1), institution, amount);
    }
    else
    {
        insertBillTable(entries.get(index - 1), institution, amount - asset +
total_amount);
        insertBillTable(entries.get(index - 1), msg.sender, asset -
total_amount);
    }
    emit financingEvent(msg.sender, institution, total_amount);
    return 0;
}
```

pay 函数，即支付操作，当企业需要支付账款单据时调用，参数为账单的信息（用来确认企业想要支付哪一张账单），即 [被借款人账户地址，见证人账户地址，时间戳，期限]，依据这些信息，在数据库中查找账单后删除，具体代码如下

```
function pay(address lender, address witness, int timestamp, int deadline)
public returns(int)
{
    // 考虑账单的存在性
    Entries entries = selectBillTable(lender, msg.sender, witness, timestamp,
deadline);
    if(entries.size() == 0)
    {
        return -1; // 账单不存在
    }
    if(entries.size() >= 2)
    {
        return -2; // 账单不唯一，重大错误，有待解决
    }

    //
    Entries entries_borrower = selectAccountTable(msg.sender);
    int credit = entries_borrower.get(0).getInt("credit");
    uint time = now;
    if(time >= uint(deadline)) // 超时减信用分
    {
        updateAccountTable(msg.sender, credit - 1);
    }
    else // 守时加信用分
```

```
    {
        updateAccountTable(msg.sender, credit + 1);
    }
    int amount = entries.get(0).getInt("amount");
    int value = removeBillTable(entries.get(0));
    emit payEvent(msg.sender, lender, witness, amount, timestamp, deadline);
    return 0;
}
```

可以看到，与之前类似，这里函数开头首先会判断账单的存在与否在进行操作，同时判断还款时间是否在期限内，若在则还款人信用加分，否则减分。事实上，如上所述，此部分的实现是很粗糙的，区块链本身并不能判断企业是否真的已经还款，因此我们认为应当增加一个函数 confirm 来让收款的企业或金融机构来确认收款，只有在现实中收到款时方才可以删除账单记录。由于时间所限，此部分功能暂未实现，后续再给予补充。

二、功能测试文档

根据项目背景的描述，希望解决项目背景中所说的使轮毂公司得到银行的融资。此部分功能测试就基于此对本次设计的智能合约进行测试和展示。

基于热身实验搭建的有四个节点的私有链，部署智能合约 SupplyChain，同时使用 FISCO-BCOS 的工具，申请五个账户来测试，如下（企业金融机构表格）

账户地址	账户身份	信用评分
0x00cd1b8249c5e68b0347d18c618f748a79429303	管理员	nan
0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9	车企	80
0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad	轮胎公司	60
0x397b96c63f4b491db929bed23262aba2dd04d7eb	轮毂公司	50
0x575a276f322da6bb074e9bf0493e32a90a338e99	银行	nan

下面是测试步骤。

1. 使用工具编译智能合约 SupplyChain.sol

   进入 fisco/console 目录，运行命令 ./sol2java.sh com 即可，运行结果如下

   ```
   *** Compile solidity SupplyChain.sol***
   INFO: Compile for solidity SupplyChain.sol success.
   *** Convert solidity to java  for SupplyChain.sol success ***

   *** Compile solidity Table.sol***
   INFO: Compile for solidity Table.sol success.
   *** Convert solidity to java  for Table.sol success ***
   ```

2. 使用管理员账户进入控制台，开始部署智能合约

   进入控制台，运行命令 getCurrentAccount 查看当前账户，并运行命令 deploy SupplyChain 进行智能合约部署，运行结果如下

   ```
   [group:1]> getCurrentAccount
   0x00cd1b8249c5e68b0347d18c618f748a79429303

   [group:1]> deploy SupplyChain
   transaction hash: 0x146a6a417f860c24386c3f6d4bee202cc0ddf0104ff4ff8393af2276136ec084
   contract address: 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
   ```

3. 仍使用管理员，创建表 Account 和 Bill

正常情况下，只有管理员能够创建表（参考代码部分的 onlyGod 修饰器）。由于此前实验已经先行创建，因此此处会创建失败，在未曾有表 Account 和 Bill 的系统中，这里是能够正常运行的。运行命令 call SupplyChain latest createAccountTable 和 call SupplyChain latest createBillTable，运行结果如下

```
[group:1]> call SupplyChain latest createAccountTable
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0x4ab263f60dceb798b3a468987023350b8ef785f11562e031b8effa5be545571c
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: The table already exist
Return message: The table already exist
Return value: [-50001]
-------------------------------------------------------------------------------------------
Event logs
Event: {}

[group:1]> call SupplyChain latest createBillTable
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0x8976e7607ece8b9d04921656955381c156be2dbc8c819d025c9f1b1cc5ad546f
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: The table already exist
Return message: The table already exist
Return value: [-50001]
-------------------------------------------------------------------------------------------
Event logs
Event: {}
```

4. 仍使用管理员，将各方账号加入平台

这里按照最开始的表格来将车企、轮胎公司、轮毂公司和银行的地址和信用评分记录进表 Account 中，即分别运行如下命令

> call SupplyChain latest insertAccountTable 0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9 "enterprise" 80

> call SupplyChain latest insertAccountTable 0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad "enterprise" 60

> call SupplyChain latest insertAccountTable 0x397b96c63f4b491db929bed23262aba2dd04d7eb "enterprise" 50

> call SupplyChain latest insertAccountTable 0x575a276f322da6bb074e9bf0493e32a90a338e99 "institution" 100

运行结果如下

```
[group:1]> call SupplyChain latest insertAccountTable 0x94c3bf18f4ddbb822aa9e3181c68debfae1b6
6c9 "enterprise" 80
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0x6e28a6ea6b9d34209a3e86d09e9c25bb333bb2b02825230923c932a29d2141ec
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [1]
-------------------------------------------------------------------------------------------
Event logs
Event: {}
```

```
[group:1]> call SupplyChain latest insertAccountTable 0xf4eb32c4d1c46a89436e1c9810c284b16e556
0ad "enterprise" 50
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0x8efb073e3f36ad3687b5ea7dd46e5fc86c51486160617257744d0c96269636c6
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [1]
-------------------------------------------------------------------------------------------
Event logs
Event: []
```

```
[group:1]> call SupplyChain latest insertAccountTable 0x397b96c63f4b491db929bed23262aba2dd04d
7eb "enterprise" 50
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0xbefb413bf7b9e58c3a4aa7d837b9ed305aa7baa52a0c41c3092dd344978a21f7
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [1]
-------------------------------------------------------------------------------------------
Event logs
Event: []
```

```
[group:1]> call SupplyChain latest insertAccountTable 0x575a276f322da6bb074e9bf0493e32a90a338
e99 "institution" 100
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0xb2c73fa088c8ac859cb07d01aa67cd116bd4b93843f12080ff9533613e813e37
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [1]
-------------------------------------------------------------------------------------------
Event logs
Event: []
```

更直观地展示，使用 sql 语句查询以确保其已经全部在 Account 表中，如下

```
[group:1]> select * from Account where id=0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9
{id=0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9, role=enterprise, credit=80}
1 row in set.

[group:1]> select * from Account where id=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad
{id=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad, role=enterprise, credit=50}
1 row in set.

[group:1]> select * from Account where id=0x397b96c63f4b491db929bed23262aba2dd04d7eb
{id=0x397b96c63f4b491db929bed23262aba2dd04d7eb, role=enterprise, credit=50}
1 row in set.

[group:1]> select * from Account where id=0x575a276f322da6bb074e9bf0493e32a90a338e99
{id=0x575a276f322da6bb074e9bf0493e32a90a338e99, role=institution, credit=100}
1 row in set.
```

对照最开始列出的企业金融机构表格可知，从上至下分别是车企、轮胎公司、轮毂公司、银行，也就是说，测试的企业和金融机构成功加入平台了

5. 登录车企的账号，向轮胎公司签发 1000 万的收款单据

   控制台切换到车企账号，先运行命令 getCurrentAccount 查看当前账号，再运行命令

   > call SupplyChain latest sign 0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad 0x575a276f322da6bb074e9bf0493e32a90a338e99 1000 365

   即车企向轮胎公司签发 1000 万，在银行的见证下，从此刻起，借款 365 天，运行结果如下

```
[group:1]> getCurrentAccount
0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9

[group:1]> call SupplyChain latest sign 0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad 0x575a276f
322da6bb074e9bf0493e32a90a338e99 1000 365
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0xf0aad97bbec5185dc8f896c2e3178350b24a13975f8b878311bad77dc92f89a7
---------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
---------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
---------------------------------------------------------------------------------------
Event logs
Event: {"signEvent":[["0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9","0xf4eb32c4d1c46a89436e1c9
810c284b16e5560ad","0x575a276f322da6bb074e9bf0493e32a90a338e99",1000,1607869215859,1607900751
859]]}
```

使用 sql 语句直观查看如下

```
[group:1]> select * from Bill where lender=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad
{lender=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad, borrower=0x94c3bf18f4ddbb822aa9e3181c68de
bfae1b66c9, witness=0x575a276f322da6bb074e9bf0493e32a90a338e99, amount=1000, timestamp=160786
9215859, deadline=1607900751859}
1 row in set.
```

可以看到账单成功加入了 Bill 表，事务上表明车企签发账单给轮胎公司成功

6. 登录轮胎公司账号，进行融资操作

   控制台切换到轮胎公司账号，先运行命令 getCurrentAccount 账号，如下

   ```
   [group:1]> getCurrentAccount
   0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad
   ```

   如项目背景所述，轮胎公司资金短缺，假设其向银行融资 300 万，则运行命令

   > call SupplyChain latest financing 0x575a276f322da6bb074e9bf0493e32a90a338e99 300

   结果如下

   ```
   [group:1]> call SupplyChain latest financing 0x575a276f322da6bb074e9bf0493e32a90a338e99 300
   latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
   transaction hash: 0xb38b51f12a30909d4b4565b72f8e653e10d855a7d3db6f922e545ab867373548
   ---------------------------------------------------------------------------------------
   transaction status: 0x0
   description: transaction executed successfully
   ---------------------------------------------------------------------------------------
   Output
   Receipt message: Success
   Return message: Success
   Return value: [0]
   ---------------------------------------------------------------------------------------
   Event logs
   Event: {"financingEvent":[["0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad","0x575a276f322da6bb07
   4e9bf0493e32a90a338e99",300]]}
   ```

   直观的，现在使用 sql 语句分别查看轮胎公司的账单情况银行的账单情况，如下

   ```
   [group:1]> select * from Bill where lender=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad
   {lender=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad, borrower=0x94c3bf18f4ddbb822aa9e3181c68de
   bfae1b66c9, witness=0x575a276f322da6bb074e9bf0493e32a90a338e99, amount=700, timestamp=1607869
   215859, deadline=1607900751859}
   1 row in set.

   [group:1]> select * from Bill where lender=0x575a276f322da6bb074e9bf0493e32a90a338e99
   {lender=0x575a276f322da6bb074e9bf0493e32a90a338e99, borrower=0x94c3bf18f4ddbb822aa9e3181c68de
   bfae1b66c9, witness=0x575a276f322da6bb074e9bf0493e32a90a338e99, amount=300, timestamp=1607869
   215859, deadline=1607900751859}
   1 row in set.
   ```

   其中，上面的是轮胎公司的，可以看到 amount = 700 的字样，且其中的借款人、被借款人、见证人、时间戳、期限等正确无误。金额同样正确无误，即 1000 - 300 = 700，这意味着轮胎公司使用车企签发的账单成功融资 300 万；下面的是银行的，可以看到 amount = 300 的字样，其中借款人仍为车企，被借款人变成了银行，时间戳和期限不变，金额也是正确的，说明车企的一部分欠款转移到了银行上，事务上而言是正确的

7. 重新切回车企的账号，对银行那一部分进行支付还款

切回账号，使用 getCurrentAccount 查看账号如下

```
[group:1]> getCurrentAccount
0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9
```

假设现在车企已经在现实中就银行那一部分的欠款还款银行，则运行如下命令进行支付

> call SupplyChain latest pay 0x575a276f322da6bb074e9bf0493e32a90a338e99 0x575a276f322da6bb074e9bf0493e32a90a338e99 1607869215859 1607900751859

其中 pay 后面参数依次是被借款人（银行）账户地址、见证人（银行）账户地址，要还的账单的时间戳和期限，运行结果如下

```
[group:1]> call SupplyChain latest pay 0x575a276f322da6bb074e9bf0493e32a90a338e99 0x575a276f322da6bb074e9bf0493e32a90a338e99 1607869215859 1607900751859
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0xdb1929e7e6e8f118f233a3a8d061c73ddb7a7253ebeab1c4a6827fe090ce4da4
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
-------------------------------------------------------------------------------------------
Event logs
Event: {"payEvent":[["0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9","0x575a276f322da6bb074e9bf0493e32a90a338e99","0x575a276f322da6bb074e9bf0493e32a90a338e99",300,1607869215859,1607900751859]]}
```

使用 sql 语句只管查看银行方面的信息，有

```
[group:1]> select * from Bill where lender=0x575a276f322da6bb074e9bf0493e32a90a338e99
Empty set.
```

银行已不持有账单，说明车企还款成功，同时，车企是在期限内还款，信用加分，原本是 80 分，现在如下

```
[group:1]> select * from Account where id=0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9
{id=0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9, role=enterprise, credit=81}
1 row in set.
```

其中 credit = 81，正确无误

8. 切回轮胎公司账号，进行转让操作

使用 getCurrentAccount 查看当前账号如下

```
[group:1]> getCurrentAccount
0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad
```

假设轮胎公司向轮毂公司买轮毂，但资金不足，因此首先是打算签发 500 万账单，运行命令

> call SupplyChain latest sign 0x397b96c63f4b491db929bed23262aba2dd04d7eb 0x575a276f322da6bb074e9bf0493e32a90a338e99 500 150

其中参数不再做解释，最后两个参数就是金额和借款时长，运行结果如下

```
[group:1]> call SupplyChain latest sign 0x397b96c63f4b491db929bed23262aba2dd04d7eb 0x575a276f322da6bb074e9bf0493e32a90a338e99 500 150
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0x39f60f2e16c43e556e0b070bae6a6d4156e947ac6feeb2b712f7bcbe24f50e51
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [-7]
-------------------------------------------------------------------------------------------
Event logs
Event: []
```

可以看到返回值为 -7，参考源代码知道，这意味着信用不达标，只有大于等于 60 才是达标的，回看前面轮胎的信用初始分，可以看到其只有 50，因此银行不做见证导致签发账单失败，这也说明了智能合约发挥了作用。签发账单失败，因此轮胎公司转而使用转让操作，希望将自己手上源于车企的 700 万转让 500 万给轮毂公司，命令如下

> call SupplyChain latest transfer 0x397b96c63f4b491db929bed23262aba2dd04d7eb 500

结果如下

```
[group:1]> call SupplyChain latest transfer 0x397b96c63f4b491db929bed23262aba2dd04d7eb 500
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0x308e0f96989859deac73138e91846e3b5932b9b32d72808bfb914f0f5d36546b
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
-------------------------------------------------------------------------------------------
Event logs
Event: {"transferEvent":[["0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad","0x397b96c63f4b491db92
9bed23262aba2dd04d7eb",500]]}
```

转让成功。使用 sql 语句查看轮胎公司和轮毂公司各自的账单如下

```
[group:1]> select * from Bill where lender=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad
{lender=0xf4eb32c4d1c46a89436e1c9810c284b16e5560ad, borrower=0x94c3bf18f4ddbb822aa9e3181c68de
bfae1b66c9, witness=0x575a276f322da6bb074e9bf0493e32a90a338e99, amount=200, timestamp=1607869
215859, deadline=1607900751859}
1 row in set.

[group:1]> select * from Bill where lender=0x397b96c63f4b491db929bed23262aba2dd04d7eb
{lender=0x397b96c63f4b491db929bed23262aba2dd04d7eb, borrower=0x94c3bf18f4ddbb822aa9e3181c68de
bfae1b66c9, witness=0x575a276f322da6bb074e9bf0493e32a90a338e99, amount=500, timestamp=1607869
215859, deadline=1607900751859}
1 row in set.
```

上面的是轮胎公司的 amount = 200 正确，下面的是轮毂公司的 amount = 500 同样正确

9. 登录轮毂公司账号，向银行融资

仍查看当前帐号 getCurrentAccount，如下

```
[group:1]> getCurrentAccount
0x397b96c63f4b491db929bed23262aba2dd04d7eb
```

轮毂公司希望向银行融资 500 万，但是本身信用 50 分，银行不信任，但是轮毂公司手持 500 万的单据，其源自车企，因此，在此平台上，轮毂公司可以直接向银行融资获取 500 万，即运行命令

> call SupplyChain latest financing 0x575a276f322da6bb074e9bf0493e32a90a338e99 500

结果如下

```
[group:1]> call SupplyChain latest financing 0x575a276f322da6bb074e9bf0493e32a90a338e99 500
latest contract address for "SupplyChain" is 0xbfca903fa0ef4a47168a5bc06299082a3b856d79
transaction hash: 0x3239b87bd8d4f7307e755b10e3e53dec90dd9a80f4603bc761965c31bd548765
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
-------------------------------------------------------------------------------------------
Event logs
Event: {"financingEvent":[["0x397b96c63f4b491db929bed23262aba2dd04d7eb","0x575a276f322da6bb07
4e9bf0493e32a90a338e99",500]]}
```

融资成功！这对于信用分为 50 的轮毂公司而言，一个命令就融资了 500 万，同时银行也无需费时费钱费力地考察轮毂公司！使用 sql 语句查看轮毂公司持有的账单和银行持有的账单，如下

```
[group:1]> select * from Bill where lender=0x397b96c63f4b491db929bed23262aba2dd04d7eb
Empty set.

[group:1]> select * from Bill where lender=0x575a276f322da6bb074e9bf0493e32a90a338e99
{lender=0x575a276f322da6bb074e9bf0493e32a90a338e99, borrower=0x94c3bf18f4ddbb822aa9e3181c68debfae1b66c9, witness=0x575a276f322da6bb074e9bf0493e32a90a338e99, amount=500, timestamp=1607869215859, deadline=1607900751859}
1 row in set.
```

上面的是轮毂公司的，为空，而下面的是银行的，手持的账单的借款人是车企，而被借款人是银行，金额 500 万，核对时间戳和期限可以发现，轮毂公司的账单转移到了银行手里。这一张账单，最初是由车企签发出来的，经由轮胎公司和轮毂公司之手，最终由银行接手。项目背景中，银行十分信任该家车企，因此手持其账单也是放心的，同时轮胎公司和轮毂公司尽管信用不高，但是也都依赖平台解决了资金的燃眉之急，其中轮毂公司的融资得益于了车企信用的传递，也就是说，此次测试是成功的。

三、总结

上面测试尽管说明了一部分的成功，但是上面的设计本身还存在非常大的缺陷。其中之一是项目设计说明中提及的支付确认尚未实现，而且一个严重的问题是，表 Bill 设计不合理，主键难以选择，很有可能一个 Bill 中会出现两条一模一样的记录，即同一借款人、同一被借款人、同一见证人、同一金额、同一时间戳、同一期限，这会是一个非常大的严重隐患，源代码并未就此情况进行处理。事实上，这种情况极有可能出现，如轮胎公司收到车企签发的 1000 万账单，转让 500 万给轮毂公司，而后轮毂公司又转让回来。事实说明，本设计尚不足以解决此情况，希望后续继续改进。

此次任务分工如下

学号	姓名	工作分配
18340097	李岩	功能测试与项目设计，部分代码编写
18340100	李泽桐	项目设计与代码编写，部分报告编写
18340105	廖家源	代码编写与功能测试，部分项目设计