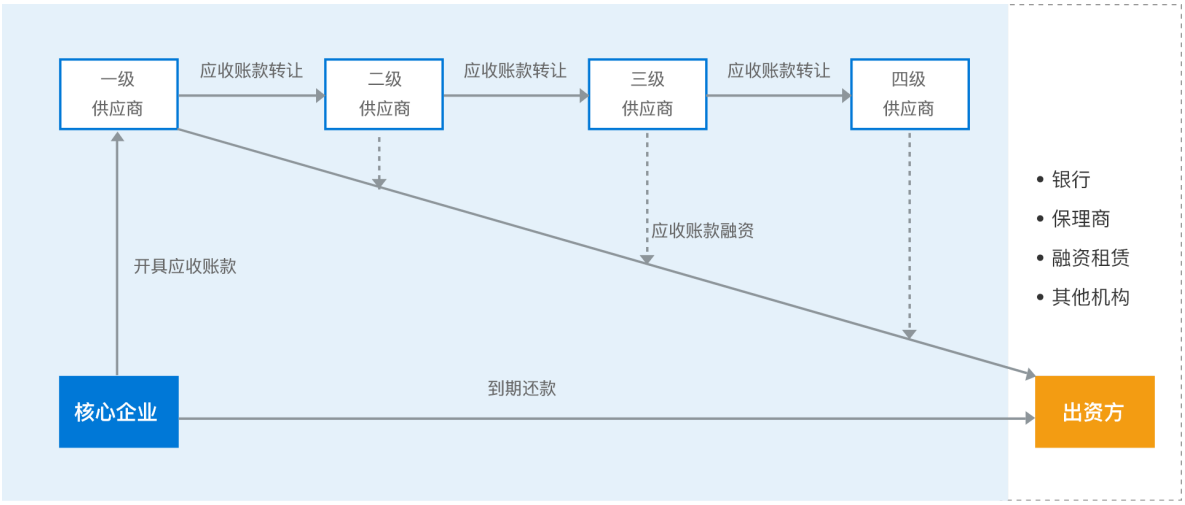
中山大学数据科学与计算机学院本科生实验报告
(2019年秋季学期)

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	17343036	姓名	郭章旭
电话	15692416866	Email	691215689@qq.com
开始日期	2019-10-23	完成日期	2019-12-08

一、项目背景

传统供应链由于交易信息的不透明化，信任关系不会逐级往下传递，从而导致了下游公司融资难的问题，同时也增加了金融机构进行评估的成本。

如果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。



根据上图，我对于区块链在供应链中的应用理解如下：首先该智能合约是要由核心企业来部署的，只有核心企业才可以开具应收账款、而一级供应商可以根据自己的情况去拆分核心企业所开具的应收账款，也可以拿去银行融资，由于核心企业开具应收账款这一个操作是由智能合约调用的，因此记录在了链上，二级供应商或银行都可以在链上查看这份应收账款是否真实有效，确保了可信性。二级供应商到三级供应商或二级供应商去融资也是同样的道理。核心企业到期还款的时候，因为每次拆分或融资都会记录在链上，所以核心企业只需要查看链上的所有应收账款的收款人地址和金额即可完成还款，而无需再花时间去验证这些子应收账款是否可信。

二、方案设计

存储设计

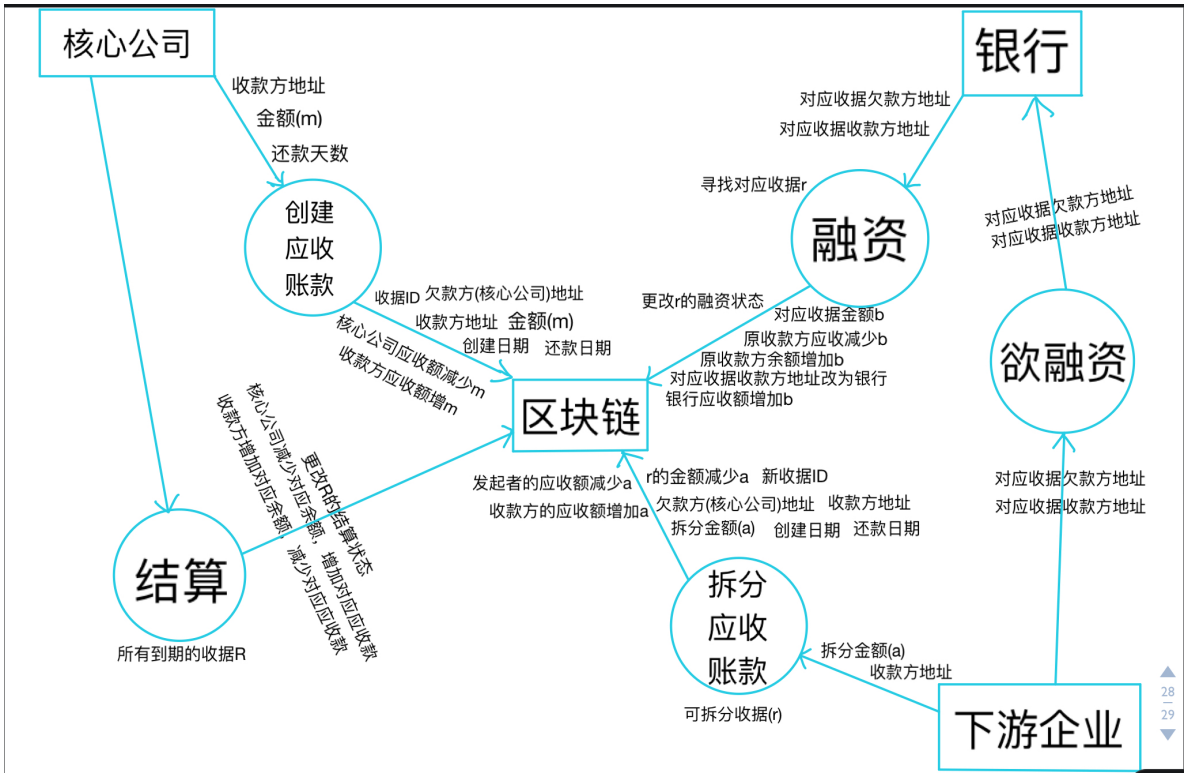
在智能合约中，最主要的是Receipt这个结构体，这个结构体保存了应收账款的欠款方、收款方，金额、还款日期等重要信息，无论是拆分账款、融资、还是结算，都得用到。

```
struct Receipt {
    uint id;                    //收据编号
    address from;               //欠款人
    address to;                 //收款人
    uint mount;                 //金额
    uint createDate;            //创建日期
    uint deadLine;              //还钱日期
    bool isRepay;               //是否已经还钱了
    bool isFinancing;           //是否已经向银行融资
}
```

然后还有就是地址与名字、余额、应收额的映射，这里主要用于显示调用是否正确，在现实中，余额通过实际的现金支付来变化。

```
mapping(address => string)public names;      //公司名
mapping(address => uint)public balances;     //公司的现金
mapping(address => int32)public credit;      //公司的信用
```

数据流图



核心功能介绍

智能合约

构造函数

```
constructor() public {
    //核心公司赋值为合约部署者
    coreCompany = msg.sender;
    //指定银行账户
    bankCompany = 0x3B8c80Ae60f5D6699B8E6fe90B63525905F7B0A3;
    //初始化银行和核心公司余额
    balances[bankCompany] = 2147483647;
    balances[coreCompany] = 200000000;
    names[coreCompany] = "核心公司";
    names[bankCompany] = "银行";
    //初始化应收账款数
    index = 0;
}
```

这里除了指定核心公司之外，还得指定银行的公钥地址，这是因为融资的时候，只有银行验证真实有效之后才可以将现金发放给想融资的企业。

创建应收账款

```
function createContract(address receiver, uint amount, uint timeToRepay)public {
    //验证是否是核心企业创建应收账款
    if(msg.sender == coreCompany){
        //计算还款时间
        uint timeTemp = now + timeToRepay * 1 days;
        //新建一份应收账款

        receipts.push(Receipt({
            id: index,
            from: msg.sender,
            to: receiver,
            mount: amount,
            createDate: now,
            deadLine: timeTemp,
            isRepay:false,
            isFinancing:false
        }));

        index ++;
        //balances[msg.sender] -= amount;
        credit[msg.sender] -= int32(amount);
        credit[receiver] += int32(amount);
        emit Createcontract("合约创建成功！ ", index, msg.sender, receiver);
    }
    //不是核心企业
    else{
        revert("Sorry! You have no right to create a receipt!");
    }
}
```

首先执行该函数的时候先验证调用的用户是否为核心企业，若是才能正常开具应收账款，否则没有权利去开具。然后根据企业之间给定的还款期限，去计算还款日期，然后新建一份应收账款并加入到存储所有应收账款的数组中。credit是用来记录用户的欠款和应收款的总额。

拆分应收账款

```
function splitReceipt(address receiver, uint amount)public{
    if(receiver == msg.sender){
        revert("Sorry! You can not fill in your own address as receiver!");
    }
    uint tempIndex = 0;
    //遍历找到属于自己要收钱的应收账款
    for(uint i = 0; i < receipts.length; i ++){
        //此应收账款收到的钱要大于等于准备拆分的钱，且没结算，没融资
        if(receipts[i].to == msg.sender && receipts[i].mount >= amount && receipts[i].isRepay == false && receipts[i].isFinancing == false){
            //更改原本的应收账款
            tempIndex = i;
            receipts[i].mount -= amount;
            credit[msg.sender] -= int32(amount);
            //创建新的应收账款
            receipts.push(Receipt({
                id: index,
                from: receipts[i].from,
                to: receiver,
                mount: amount,
                createDate: now,
                deadLine: receipts[i].deadLine,
                isRepay:false,
                isFinancing:false
            }));
            emit Split(tempIndex, "原本收据拆分成功", "新收据创建成功", index, receipts[i].from, receiver);
            index ++;
            credit[receiver] += int32(amount);
        }
    }
}
```

首先是拆分的发起方找到属于自己收款的应收账款，找到后判断一下该份应收账款的金额是否大于等于拆分金额，如果是则进行拆分，将原本应收账款的金额减去拆分金额，新建的应收账款的信息除了id、收款人、金额、创建日期不同以外，其他与原本应收账款一样。为了防止企业把结算了的或者融资了的应收账款再进行拆分，拆分前再添加了两个条件：该应收账款没有被结算或者没有被融资。

融资应收账款

```
function financing(address from, address to)public{
    //只能由银行调用
    if(msg.sender != bankCompany){
        revert("Sorry! You have no right to call financing function!");
    }
    else{
        //遍历所有的应收账款
        for(uint i = 0; i < receipts.length; i ++){
            if(receipts[i].from == from && receipts[i].to == to && receipts[i].isRepay == false && receipts[i].isFinancing == false){
                receipts[i].to = bankCompany;
                receipts[i].isFinancing == true;
                credit[to] -= int32(receipts[i].mount);
                credit[bankCompany] += int32(receipts[i].mount);
                balances[to] += receipts[i].mount;
                balances[bankCompany] -= receipts[i].mount;
                emit Financing("融资成功", bankCompany, msg.sender, receipts[i].mount);
            }
        }
    }
}
```

思路很简单，就是把要融资的应收账款的收款人改成银行的地址，然后融资的企业现金金额增加就好了。调用这个函数必须由银行来操作，因为现金只能由银行来发行。想融资的企业，只需将应收账款中的欠款方地址和收款方地址告诉银行即可。

结算

```
function repay()public{
    //只能由核心公司调用
    if(msg.sender == coreCompany){
        for(uint i = 0; i < receipts.length; i ++){
            //当前日期大于还款日期，执行还款
            //if(now >= receipts[i].deadLine){
                //判断是否已经还款
                if(receipts[i].isRepay == false){
                    balances[receipts[i].to] += receipts[i].mount;
                    credit[receipts[i].to] -= int32(receipts[i].mount);
                    balances[receipts[i].from] -= receipts[i].mount;
                    credit[receipts[i].from] += int32(receipts[i].mount);
                    receipts[i].isRepay = true;
                    emit Repay( "还款成功", receipts[i].id);
                }
            //}
        }
    }
    else{
        revert("Sorry! You have no right to call repay function!");
    }
}
```

结算函数只能由核心公司调用，原本的思路是遍历存放所有应收账款的数组，把到期还款的而还没结算的应收账款给结算掉，核心公司金额减少，对于收款人金额增加。在测试的时候，为了方便测试，我把判断当前日期是否大于还款日期的语句注释掉了，只要核心公司执行结算函数，就会把所有没结算的应收账款给结算掉。

辅助函数

```
function getCredit(address addr) public view returns(int) {
    return credit[addr];
}

function getBalance(address addr) public view returns(uint) {
    return balances[addr];
}

function setName(string memory name2) public{
    if(msg.sender == bankCompany){
        revert("Sorry! The bank can't change name!");
    }
    names[msg.sender] = name2;
}

function getName(address addr) public view returns(string memory) {
    return names[addr];
}
```

用于给前端显示和设置公司名字。

利用Truffle框架部署合约

使用turffle命令初始化好项目后，只需要将智能合约放入contracts，然后根据自己的合约修改index.html和index.js即可获得一个网页版的区块链应用。

修改index.js使其可以调用合约的函数。

例如创建应收账款的函数：

```
createContract: async function() {
    if(this.account != company){
        this.setStatus("Sorry! You have no right to create a receipt!",
"status1");
    }
    else{
        const receiver = document.getElementById("create-receiver").value;
        if(receiver == this.account){
            this.setStatus("Sorry! You can not fill in your own address as
receiver!", "status1");
        }
        else{
            const amount = parseInt(document.getElementById("create-
amount").value);
            const time = parseInt(document.getElementById("create-
repay").value);
            //console.log(amount);
            this.setStatus("Initiating receipt... (please wait)", "status1");

            const { createContract } = this.meta.methods;
            await createContract(receiver, amount, time).send({ from:
this.account, gas: 6721975 });
            //await createContract(receiver, amount, time).estimateGas({from:
this.account});
            this.setStatus("Create receipt completed!", "status1");
            this.refreshBalance();
            this.refreshCredit();
            //console.log('Create receipt complete!');
        }
    }
},
```

首先是通过网页的输入获得调用合约函数的必要参数，然后通过`const { createContract } = this.meta.methods`来找到智能合约中对应的函数，最后将参数传过去即可调用。由于在js中已经检查了当前账户是否为核心公司，所以智能合约该函数里检查调用者是否为核心公司并不会用到，如果在js中没有检查，则可以打开控制台来查看错误信息。其他函数的调用实现类似。

三、功能测试

测试环境

Windows 10 1909

nodejs

npm

ganache

truffle

测试说明

配置好上述所需要的环境后，终端输入命令`ganache-cli`，执行结果如下：

```
C:\Users\Kwok Cheung Yuk>ganache-cli
Ganache CLI v6.7.0 (ganache-core: 2.8.0)

Available Accounts
==================
(0) 0x27c8baFA36F948C8B51C7D639E3f00E44aF6C70f (100 ETH)
(1) 0xeDFE521051Ba99eDD5eF6EeF83EeaFf9deF7fC03 (100 ETH)
(2) 0x0bb5379f43C4395DB26c7D495Bf84aFD09932a7a (100 ETH)
(3) 0x2d0A491BC89D173D24097963E271D9F609FF6f27 (100 ETH)
(4) 0xA46481035e5c030e7760a7f37966e5ab7c6AEF94 (100 ETH)
(5) 0xc96552F686Cc85c96e1bbf5966Af9F82568FC14C (100 ETH)
(6) 0x2E5E6BD705A6291Dcc83A7211878A02A5349b76e (100 ETH)
(7) 0x3f1D557E87271ddBAE92333449dBBA64B2699363 (100 ETH)
(8) 0xCF2771e8A547D697C3EB877e9827060D43Eb3fDF (100 ETH)
(9) 0x5a9D1614c293F2EA1750216E12abcF9e61b83B63 (100 ETH)

Private Keys
==================
(0) 0x5c768511c3829db58be4311701d020982d3140892f6640625d91a0622f7e2ce6
(1) 0xc1b1a3dc4c7c99430e8a2a0fa029018fd051422a6020528b603eaf35ca003d42
(2) 0x733b97e33d307bab6b43d819618b66a2aba8d193fe401f72f30daf6629abc492
(3) 0xe49140b9f94f0552287b85faddd0c93429d1f0d2151f75445ee524f9d1824eb6
(4) 0x7ae80dc9587d73740d9da45f4a518b2d3e91a473e4f2c587f0b199e1d4c21b22
(5) 0xa2750334a78fc60a552bdc0f346536964255120a684912ef43f0e48a15ea49eb
(6) 0x473fdc5267fb2aadd6f13cb5a4ebc9cfd8b4800cc51584774113981c9fa7943f
(7) 0x519d30d5eca7079a73bfe66595e4c9d53c69f820903cb87e7e5d76f5ef04b6dd
```

然后修改SupplyChainApp/contracts/SupplyChain.sol中的银行地址，ganache会给出10个地址，自行选一个即可：

```
constructor() public {
    //核心公司赋值为合约部署者
    coreCompany = msg.sender;
    //指定银行账户
    bankCompany = 0x3B8c80Ae60f5D6699B8E6fe90B63525905F7B0A3;
    //初始化银行和核心公司余额
    balances[bankCompany] = 2147483647;
    balances[coreCompany] = 200000000;
    names[coreCompany] = "核心公司";
    names[bankCompany] = "银行";
    //初始化应收账款数
    index = 0;
}
```

新建一个终端，进入到SupplyChainApp/app，执行命令`truffle migrate`：

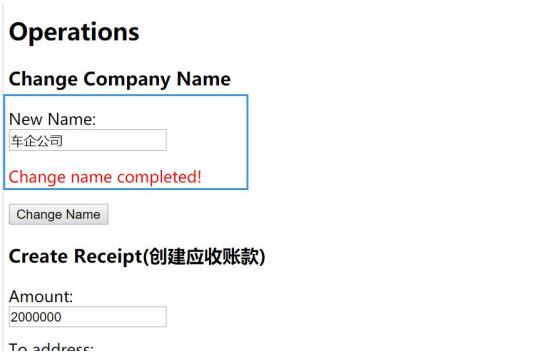
```
C:\Users\Kwok Cheung Yuk\Desktop\SupplyChainApp\app>truffle migrate

Compiling your contracts...
===========================
> Compiling .\contracts\SupplyChain.sol

    > compilation warnings encountered:

/C/Users/Kwok Cheung Yuk/Desktop/SupplyChainApp/contracts/SupplyChain.sol:74:13: Warning: Unreachable code.
            revert("Sorry! You have no right to create a contract!");
            ^-----------------------------------------------------^
,/C/Users/Kwok Cheung Yuk/Desktop/SupplyChainApp/contracts/SupplyChain.sol:121:13: Warning: Unreachable code
            revert("Sorry! You have no right to call financing function!");
            ^------------------------------------------------------------^
,/C/Users/Kwok Cheung Yuk/Desktop/SupplyChainApp/contracts/SupplyChain.sol:162:13: Warning: Unreachable code
            revert("Sorry! You have no right to call repay function!");
            ^--------------------------------------------------------^
,/C/Users/Kwok Cheung Yuk/Desktop/SupplyChainApp/contracts/SupplyChain.sol:177:13: Warning: Unreachable code
            revert("Sorry! The bank can't change name!");
            ^-------------------------------------------^


> Artifacts written to C:\Users\Kwok Cheung Yuk\Desktop\SupplyChainApp\build\contracts
> Compiled successfully using:
```

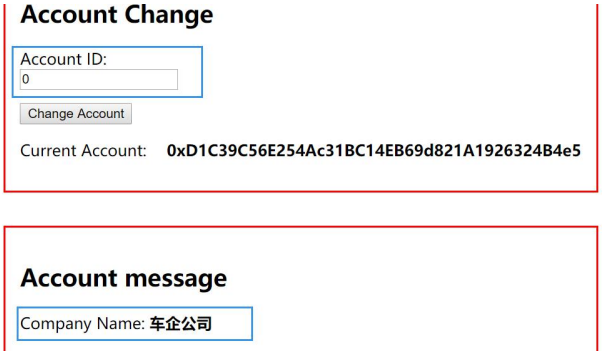
执行命令`npm run dev`运行服务器：

```
C:\Users\Kwok Cheung Yuk\Desktop\SupplyChainApp\app>npm run dev

> app@1.0.0 dev C:\Users\Kwok Cheung Yuk\Desktop\SupplyChainApp\app
> webpack-dev-server

[wds]: Project is running at http://localhost:8080/
[wds]: webpack output is served from /
[wds]: Content not from webpack is served from C:\Users\Kwok Cheung Yuk\Desktop\SupplyChainApp\app\dist
[wdm]: Hash: 2a67c95056cca9a02559
Version: webpack 4.41.2
Time: 1868ms
Built at: 2019-12-08 11:07:53
     Asset      Size  Chunks             Chunk Names
index.html  3.46 KiB          [emitted]
  index.js  2.93 MiB    main  [emitted]  main
Entrypoint main = index.js
[0] multi (webpack)-dev-server/client?http://localhost:8080 ./src/index.js 40 bytes {main} [built]
[../build/contracts/SupplyChain.json] 982 KiB {main} [built]
[./node_modules/ansi-html/index.js] 4.16 KiB {main} [built]
[./node_modules/html-entities/index.js] 231 bytes {main} [built]
[./node_modules/web3/src/index.js] 2.01 KiB {main} [built]
[./node_modules/webpack-dev-server/client/index.js?http://localhost:8080] (webpack)-dev-server/client?http://localho
080 4.29 KiB {main} [built]
[./node_modules/webpack-dev-server/client/overlay.js] (webpack)-dev-server/client/overlay.js 3.51 KiB {main} [built]
[./node_modules/webpack-dev-server/client/socket.js] (webpack)-dev-server/client/socket.js 1.53 KiB {main} [built]
[./node_modules/webpack-dev-server/client/utils/createSocketUrl.js] (webpack)-dev-server/client/utils/createSocketUr
 2.89 KiB {main} [built]
```

浏览器输入http://localhost:8080/进行访问。

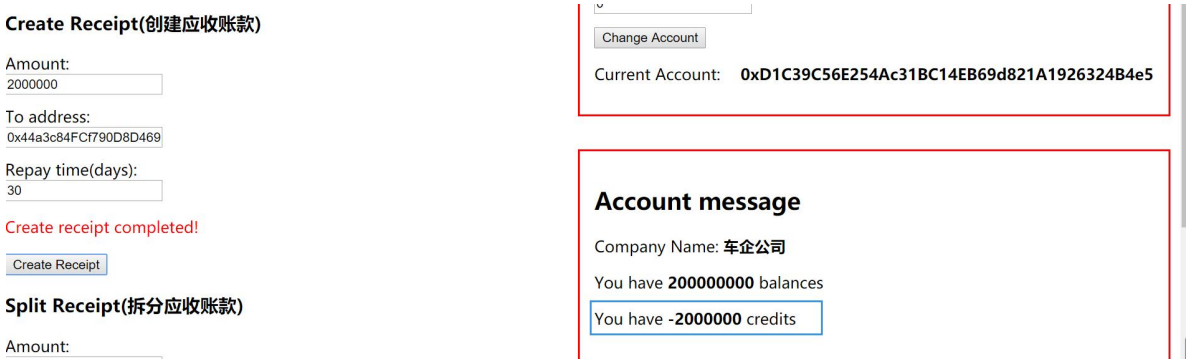
更改公司名字

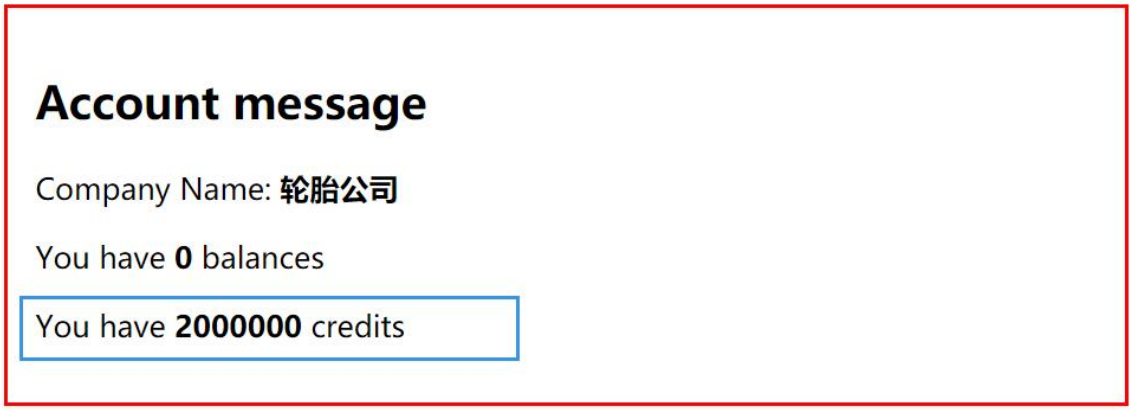




创建应收账款

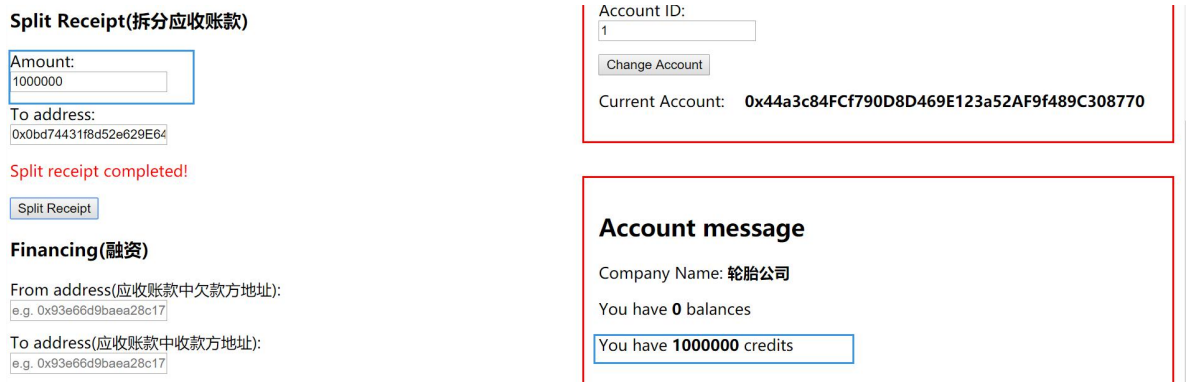
创建成功后，车企公司(核心公司)的应收账款总额减少，轮胎公司应收账款总额相对于增加：

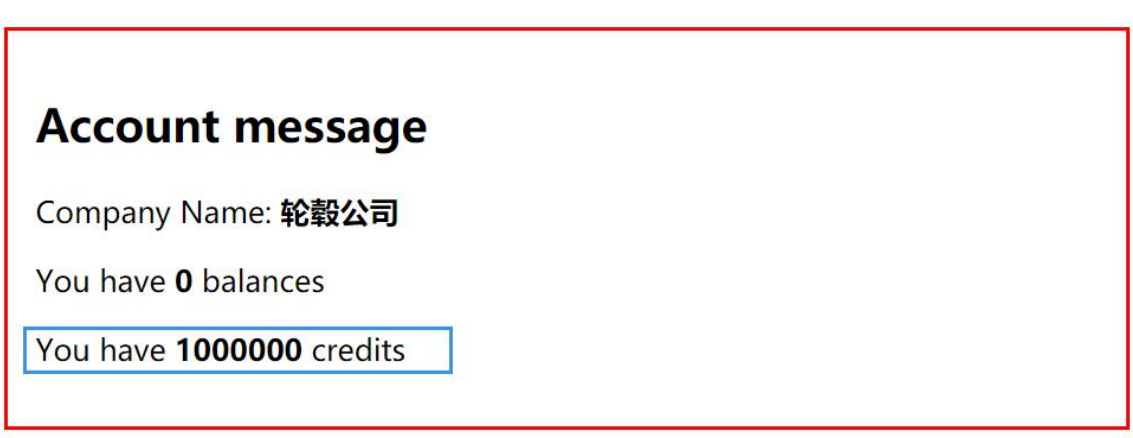




拆分应收账款

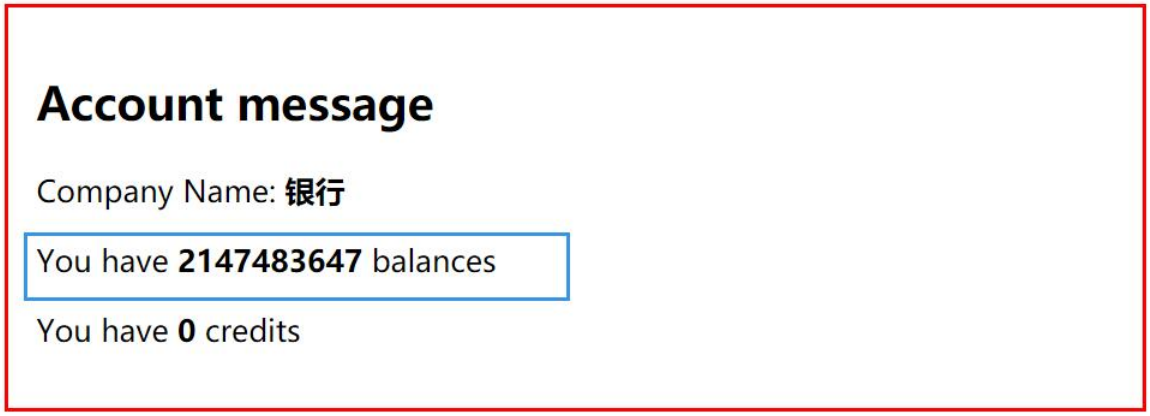
轮胎公司将应收账款拆分成功后，轮胎公司应收账款总额减少对于金额，轮毂公司应收账款总额对于增加：



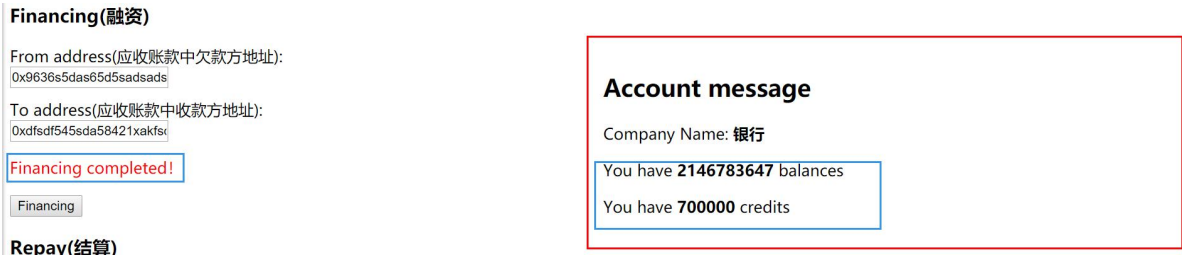


融资

融资前银行余额：

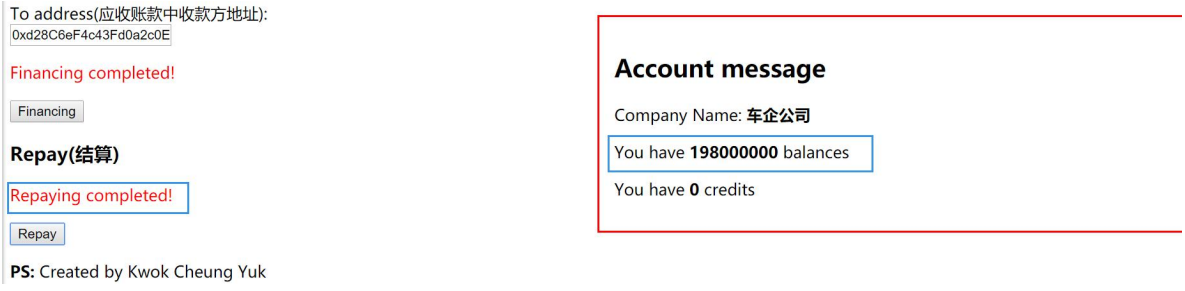


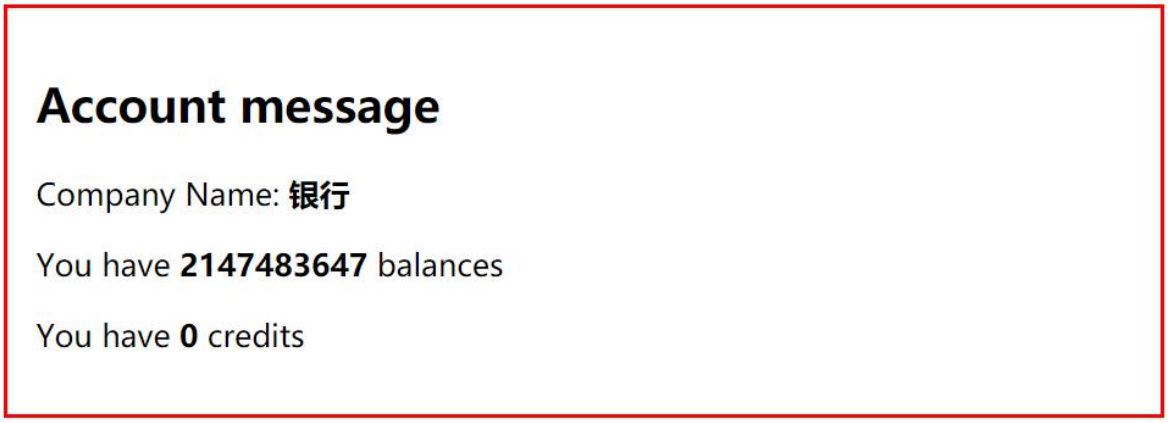
融资成功后银行和下游小企业的余额：



结算

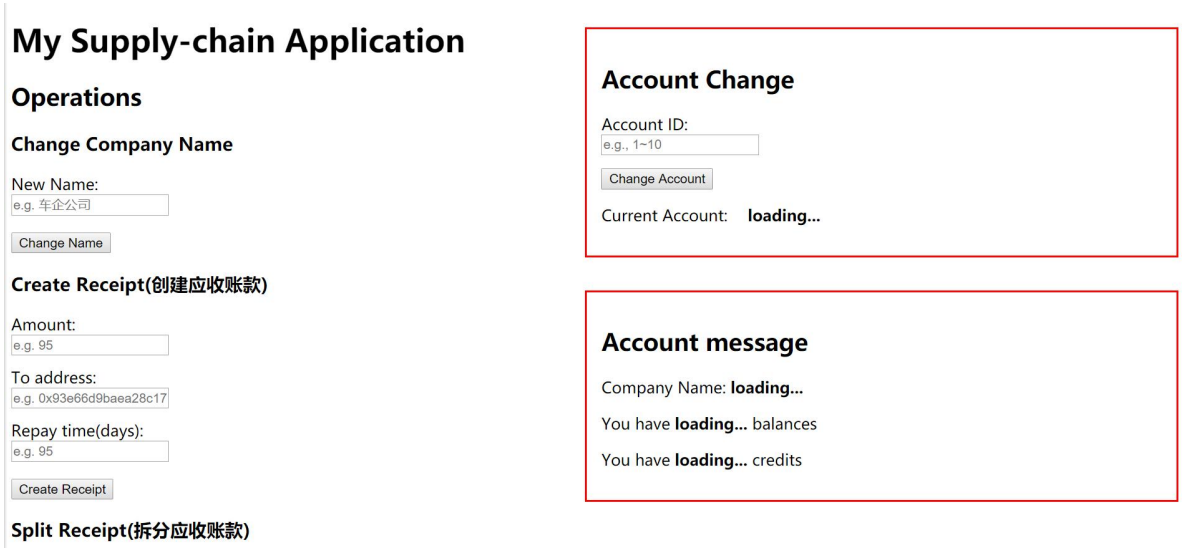
结算后各家公司的余额和营收额变化：





四、界面展示

界面是用truffle框架搭建的，比较简陋，左边是对应的一些操作，右上部分用于切换账户，右下部分用于显示账户信息，包括名字，余额，应收账款总额。



五、心得体会

从一开始以为区块链等价于比特币到现在自己可以完完整整写一个智能合约，做一个小应用，这个过程虽短，但是真的是挺不容易的。在这过程中，需要自己学的东西有很多，例如学习solidity的语法呀，如何编写一个合约、如何将合约部署到链上。微众银行的Webase让合约的部署，合约的测试变得简单多了。然后到后面链端前端后端相结合做成一个完整应用，一开始也是一脸懵逼的，不知道要如何做，后来在网上看到可以用truffle框架来构建，我就尝试了一下，慢慢磨，也做出来了，所幸自己之前学过一些HTML和JavaScript的语法，所以这一部分也不太难。而框架提供的demo中，也相当明白地表示了如何去调用合约中的函数，所以只要照葫芦画瓢，就可以做好了。当然我个人觉得，最令人费神的就是配置环境这一部分了，国内很多资源都墙了，还得想方设法"科学上网"，做完这个作业，我又一次感觉到国内网络对国内开发者的不友好了。。。。。。

《区块链原理与技术》这门课也逐渐接近了尾声，区块链技术相对来讲是比较新颖的，但是任课老师讲得很透彻，而且还找来了其他各领域厉害的老师来给我们讲相对应的知识点，更是找来了微众银行的专家来教我们如何完成大作业，这是很难得的一个机会。虽然课快上完了，但是对于区块链，我了解的只是很少很少，希望自己日后会有更多时间深入了解区块链这一个领域。

GitHub仓库