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　　任课教师： 郑子彬

年级	2017	专业（方向）	软件工程
学号	17343023	姓名	董宸宇
电话	15622126658	Email	815694609@qq.com
开始日期	2019.12.1	完成日期	2019.12.13

一.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，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

实现功能：

功能一：实现采购商品—签发应收账款易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。

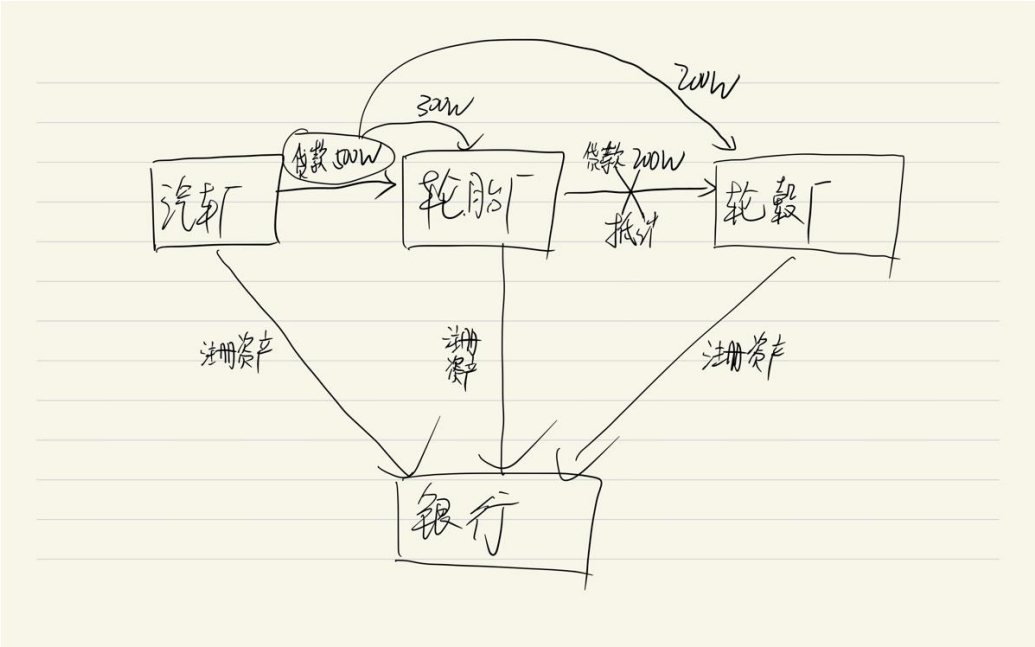
功能二：实现应收账款的转让上链，轮胎公司从轮毂公司购买一笔轮毂，便将于车企的应收账款单据部分转让给轮毂公司。轮毂公司可以利用这个新的单据去融资或者要求车企到期时归还钱款。

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。

二.方案设计

一个简单的数据流图如下(手绘版，画的比较草)



在阶段二中，我们已经编写了合约成功实现了本次作业所需要的功能，但由于一些原因，阶段二的合约不能够直接拿到这里面来用
其中有一个地方就是在阶段二中，我们可以直接在浏览器中进行用户的注册，但是在这里面不行，需要自己来搞用户的注册
在这里，我参照了 FISCO BCOS 官网教程里面的例子，他的例子里面存储结构使用了一个表，表的内容有两项，分别是账户名以及资产，(我原本想把债务加上去，但是在改变表的过程中会出现一些莫名其妙的问题，所以我就把债务按照阶段二的方法单拿出来了)
注册的过程其实就是往表里面添加一项
关键代码如下

```
Table table = openTable();
Entry entry = table.newEntry();
entry.set("account", account);
entry.set("asset", int256(asset_value));
```

Loan 函数遵循了阶段二中的思路，先检测 AB 两厂之间有没有借过钱，有的话就直接在旧的债务上添加金额就可以了，没有的话就新建债务账单
核心代码如下

```
for(uint i=0;i<len_rent;i++){
    if(rent[receiver][i].borrower==sender){
        rent[receiver][i].mount+=amount;
        rent_appear=true;
        break;
    }
}
if(rent_appear==false){
    receipt_rent memory rece=receipt_rent(sender,amount,reason,believe);
    receipt_borrow memory send=receipt_borrow(receiver,amount,reason,believe);
    rent[receiver].push(rece);
    borrow[sender].push(send);
}
```

Partsend 函数:这个函数可以说是本次作业中最复杂的一个函数了，涉及到部分合约转让，这里依旧延续了阶段二中相关合约的设计

(1)当前需要借钱的数目小于当前账单的额度

这种情况就需要在当前的账单中将相应的额度减掉，然后再给 receiver 以及账单的原债务人各添加一份相应的合约

```
int left=rent[sender][i].mount-needsend;
rent[sender][i].mount=left;
for(uint j=0;j<borrow[debt].length;j++){
    if(sender==borrow[debt][j].renter){
        if(borrow[debt][j].believe==false){
            continue;
        }
        else{
            borrow[debt][j].mount=left;
            receipt_rent memory rece=receipt_rent(debt,needsend,reason,true);
            rent[receiver].push(rece);
            receipt_borrow memory send=receipt_borrow(receiver,needsend,reason,true);
            borrow[debt].push(send);
        }
    }
}
```

当前需要借钱的数目大于当前账单的额度

这种情况跟刚才的情况是类似的，只不过将当前账单直接归 0，然后给 receiver 以及账单的原债务人各添加一份相应的合约，并继续遍历

```
rent[sender][i].mount=0;
for(uint j1=0;j1<borrow[debt].length;j1++){
    if(borrow[debt][j].believe==false){
            continue;
    }
    if(sender==borrow[debt][j1].renter){
        borrow[debt][j1].renter=receiver;
    }
}
```

最后是 pay 函数，这个函数的功能是偿还债务，这里也有些麻烦，因为不但要在债务数额上变化，在资产上也要有相应的变化，资产方面也要有相应的更改，在资产方面我遵循了网站上样例的设计，在表中检索到合约之后对资产进行增加/减少，而在合约方面我遵照了阶段二的思路：总体思路就是先通过遍历找到相应的合约，如果 money 大于账单上面的数额，则返回错误，小于等于的话就在相应账单中减去相应数额

```
for(uint i=0;i<pos;i++){
    if(pay_fac==renter[i].borrower&&get_fac==renter[i].borrower){
        rent[i].mount-=money;
        transfer(pay_fac,get_fac,money);
        uint len1=rent[get_fac].length;
        uint len2=borrow[pay_fac].length;
        for(uint j=0;j<len1;j++){
            if(rent[get_fac][j].borrower==pay_fac){
            int now_money=rent[get_fac][j].mount;
            if(money>now_money){
                return "too many money";
            }
            else{
                rent[get_fac][j].mount-=money;
                for(uint k=0;k<len2;k++){
                    if(borrow[pay_fac][k].renter==get_fac){
                        borrow[pay_fac][k].mount-=money;
                        return "pay successful";
                    }
                }
            }
        }
    }
    emit PayEvent(pay_fac, get_fac, money);
    return "pay successful";
```

三.功能测试

首先对合约进行编译和部署

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app$ ./gradlew build

BUILD SUCCESSFUL in 3s
4 actionable tasks: 2 executed, 2 up-to-date
fisco-bcos@fiscobcos-VirtualBox:~/asset-app$ cd dist
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh deploy
 deploy success, contract address is 0x5320b2d289bfecc64425e147025a47ed6573525f
```

然后进行资产的注册

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh register BMW
 10000000
 register asset account success => asset: BMW, value: 10000000
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh register Tir
es 10000000
 register asset account success => asset: Tires, value: 10000000
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh register Whe
el 10000000
 register asset account success => asset: Wheel, value: 10000000
```

可见注册资产成功，汽车厂，轮胎厂，轮毂厂都注册了 1000W

loan 函数:宝马厂购买轮胎，向轮胎厂贷款 500W

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh loan Tires B
MW 5000000
 loan success => from_asset: Tires, to_asset: BMW, amount: 5000000
```

queryloan 函数:用来查询债务，比如前面汽车厂向轮胎厂借了 500W

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh queryloan BM
W Tires
sum debt => BMW to Tires 5000000
```

然后还需要进行的是债务的转让

轮胎厂向轮毂厂借了 200W
当然轮胎厂手里还有汽车厂 500W 的账，所以它大可以拿着这 500W 的账来支付这 200W

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh partsend Tir
es Wheel BMW 2000000
send the debt successfully
```

这样就完成了债务的转让，那么这时候应该是汽车厂有两笔债务，一共 500W，轮胎厂 300W，轮毂厂 200W

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh queryloan BM
W Tires
sum debt => BMW to Tires 3000000
```

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh queryloan BM
W Wheel
sum debt => BMW to Wheel 2000000
```

接下来测试偿还函数
现在汽车厂欠轮胎厂 300W，在这里先偿还 200W

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh pay BMW Tire
s 2000000
 pay success => from_asset: BMW, to_asset: Tires, amount: 2000000
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh query BMW
 asset account BMW, value 8000000
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh query Tires
 asset account Tires, value 12000000
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh queryloan BM
W Tires
sum debt => BMW to Tires 1000000
```

还了钱以后，两边资产数量都有变化，汽车厂资产为 800W，轮胎厂资产为 1200W，查询一下两厂之间的债务发现只剩下了 100W

四.界面设计

UI 是自己糊的，没用模板，所以特别丑，仅仅是能用
登录界面

登录之后切到这个界面来

这就是 4 个功能，贷款，转让合约，查询，付款
贷款界面

转让合约

支付

五.心得体会

对我来说这次大作业难度还是挺大的，因为这次作业是要求做全栈，方方面面都要考虑到，网上的参考资料也不多，所以有些时候会遇到很多困难，这里面涉及的绝大多数内容都得从头学起，之前从来都没有接触过，从 FISCO-BCOS 的环境，再到用 solidity 语言编写合约，把合约编译成 JAVA，再到第三阶段用 JDK 将链端和后端连起来

在区块链这门课程的前几节课中，学习到了一些跟区块链相关的基础知识，但是很多东西还是很懵，但是在这次大实验中，我对很多关于区块链相关的知识有了更深一步的理解，在大作业的第一阶段中，我们使用 FISCO-BCOS 来进行了私有链的搭建以及节点的加入，并将合约成功部署到了私有链上，并成功的调用了合约，然后还知道如何使用命令来查看区块，以及各个字段的含义

在熟悉了一些相关的知识之后，开始对我们目前所处的情境进行分析，了解清楚我们当前的情况之后，根据当前情境的需求，编写相应的合约，这是阶段二的内容

第三阶段的难度比我想象的大很多，我最开始以为只需要一个前端，然后在前端中调用链端就可以完成，但是在我仔细了解了一下这个情况之后，我发现事实并不如此，在这里前端和链端中间还需要一个后端作为中间层，这个中间层负责调用链端，同时前端调用后端，完成整个作业，而且我在阶段二中所写的链端也因为一些原因不能直接在这里面用，需要作以改进，负责连接后端和链端 JAVA SDK 我在之前也完全没有接触过，所以也要从头学起，在这里我参照了官网上给的例子，并沿用了部分例子中的结构，算是找到了”抓手”，避免像无头苍蝇一样四处乱撞，同时还要对合约进行修改，使其能够完全契合后端，可能是我对相关知识的理解还不够深入，在这里遇到了很多莫名其妙的 Bug，经常因为一个 Bug 卡住好久，最后，后端和链端算是勉强调通了，但是前端和后端方面我却一直没有弄好，所以我的前端页面暂时连不上后端，这也是在本次作业中做的不好的地方

在这次大作业中，我的收获还是很大的，通过这次作业的操作，我对很多区块链的知识有了更深一步的了解，包括链，节点以及合约等等，在这次作业中也遇到了很多很多的问题，在遇到问题的时候也曾烦躁过(虽然有些问题还是没有解决)，但是在解决问题的时候一样有很大的收获