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 级	专业（方向）	软件工程
学号	17343126	姓名	肖靖烨
电话	15626225129	Email	673189914@qq.com
开始日期	2019/11/13	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。

在 fisco-bcos 中，利用区块链我们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

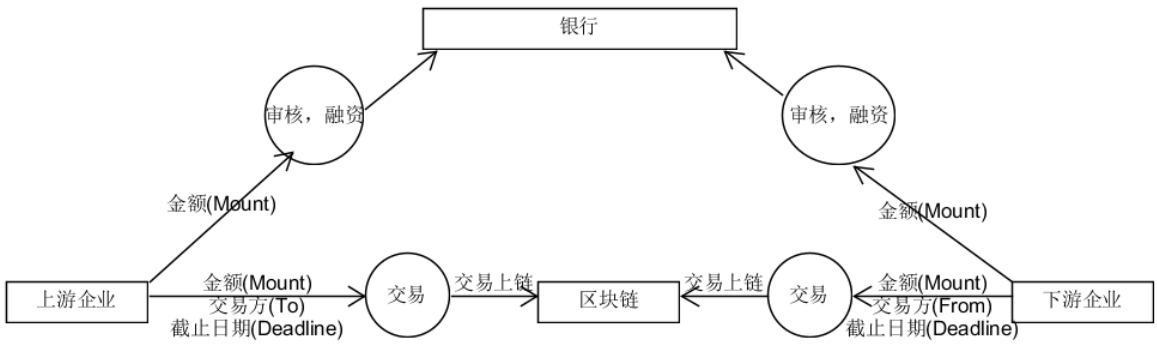
在供应链金融车企这个场景中，我们需要完成以下功能，

1.实现采购商品—签发应收账款 交易上链。签订应收账款单据。
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，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。
为此，我设计了如下功能接口

```
1.  function Grade(address To,uint mount,string From_To,int Date,uint num,bool status )
2.  function GradeWithBank()
3.  function Repay(int Date)
```

Grade 实现了**功能 1** 和**功能 2**
GradeWithBank 实现了**功能 3**
Repay 实现了**功能 4**

数据流图



在 receipt 中，其储存结构为
Num（单据编号）
From（欠款人）
To（收款人）
Mount（金额）
State（状态（是否被银行确认））
Deadline（还款截止日期）

（1）Grade（交易）：

首先判断 balance 里的钱是否够付款，如果够的话就用 balance 里的钱付款，不够则打欠条。

```
1.   balances[msg.sender]-=mount;
```

接下来，便是如何打欠条，如果 B 给 C 打欠条，但 B 那里有 A 的一张欠条，看 A 给 B 的欠条是否够 B 给 C，足够的话，将 A 给的欠条分一部分给 C

```
if(receipts[i].mount>mount)
                {
                    receipts[num].come=receipts[i].come;
                    receipts[num].to=To;
                    receipts[num].mount=mount;
                    receipts[num].number=num;
                    receipts[num].deadline=Date;
                    receipts[num].status=status;

                    receipts[i].mount=receipts[i].mount-mount;

                }
```

不够的话，将 A 的欠条全部给 C，并让 B 在给 C 一张欠条，这里单据编号为当前单号加 10

```
 else
                {


                    receipts[num].come=receipts[i].come;
                    receipts[num].to=To;
                    receipts[num].mount=receipts[i].mount;
                    receipts[num].number=num;
                    receipts[num].deadline=Date;
                    receipts[num].status=status;

                    receipts[num+10].come=msg.sender;
                    receipts[num+10].to=To;
                    receipts[num+10].mount=mount-receipts[i].mount;
                    receipts[num+10].number=num+10;
                    receipts[num+10].deadline=Date;
                    receipts[num+10].status=status;

                     receipts[i].mount=0;
                }
```

但以上条件均不满足时，重新创建一张欠条。

```
 receipts[num].come=msg.sender;
                receipts[num].to=To;
                 receipts[num].mount=mount;
                receipts[num].number=num;
                receipts[num].deadline=Date;
                receipts[num].status=status;
```

（2）GradeWithBank（融资）

在 receipt 中遍历查找别人给它的欠条，也就是查找 To，查找到时将 receipt 的 to 变为银行，将企业的 balances 加上银行融的资。

```
uint sum=0;
        for(uint i=1;i<30;i++)
       {
           if(receipts[i].to==msg.sender)
           {
               balances[msg.sender] += (receipts[i].mount);
               sum+=receipts[i].mount;
               receipts[i].to=Manager;
```

```
            }
        }
```

（3）Repay（还债）：

遍历 receipt，知道来源于自己的欠条，也就是 come==自己，如果超出还款时间，则不能还款，当然这里可以设计成超出时间后按利息算钱。将 receipt 的金额置为 0，并改变其状态，表示还款成功。

```
for(uint i=1;i<30;i++)
        {
            if(receipts[i].come==msg.sender)
            {
                 if(Date>receipts[i].deadline)
        return ("Fail long time");

                else
                {
                    sum+=receipts[i].mount;
                receipts[i].mount=0;
             receipts[i].deadline=99999999;
                }
            }
        }
```

在后端 nodejs 中利用 get 处理不同事件

```
if(From==null)//银行融资
         {
             console.log("rongzhi");
             var content = fs.readFileSync('public/res2.html');
             var s=[];
             s.push(Num);
             s.push(To);
             var promise2 = new Promise(function(resolve, reject) {
                 resolve(we.sendRawTransaction("0xe4a741c3b8209e5e2fc443ace071fd950f875128","GradeWithBank(string,string)",s));
             });

             promise2.then(function(value) {
                    console.log(value);
                 });

         }
```

三、 功能测试

首先我们在虚拟机上配置好 webase-front 环境，配置完成后会出现以下界面

我们首先创建一个管理员用户 Bank，它负责部署合约

Bank	700005	0xf1e423f310f...	正常	修改

接着创建三个普通用户 A，B，C 充当企业。

C	700012	0xd51a0e60a...	正常	修改
B	700011	0xfb286113fb...	正常	修改
A	700010	0xf46c174293...	正常	修改

他们的地址分别为

A: 0xf46c1742933c0f854049691657ae7ffcda0c614b

B: 0xfb286113fb04ad1df509753ebd90c68a9119a5de

C: 0xd51a0e60aae7735135c16d64fec1bf549d1a60c7

现在我们开始测试合约，首先将合约部署到 Bank 上

我们测试交易 Grade 接口：

企业 A 向企业 B 购买了价值 200 的物品，向 B 写了一张 200 的收据。

其参数含义如下：

To：为接收收据的企业地址，这里为 B 的地址

Mount：收据金额

From_To:从 A 到 B 的收据

Date:还款的最后期限

Num：为单据编号，

Status:银行是否确认了交易

交易成功后我们根据编号查询此收据

查看 receipt1[1]里面的内容

根据地址显示出我们的收据正确

我们继续调用 Grade 接口，测试从 B 到 C 实现应收账款的转让上链。这里设置为 B 应当给 C 100 的收据。

查看 receipt[1]里面的内容，说明 A 给 B 的收据中的 100 以被转让给下家 C 了。

Receipt[2],从地址可以判断出这是来自于 A 的欠条。

接下来测试 GradewithBank，让 C 找银行融资，它拿着 A 的欠条可以融资到 100

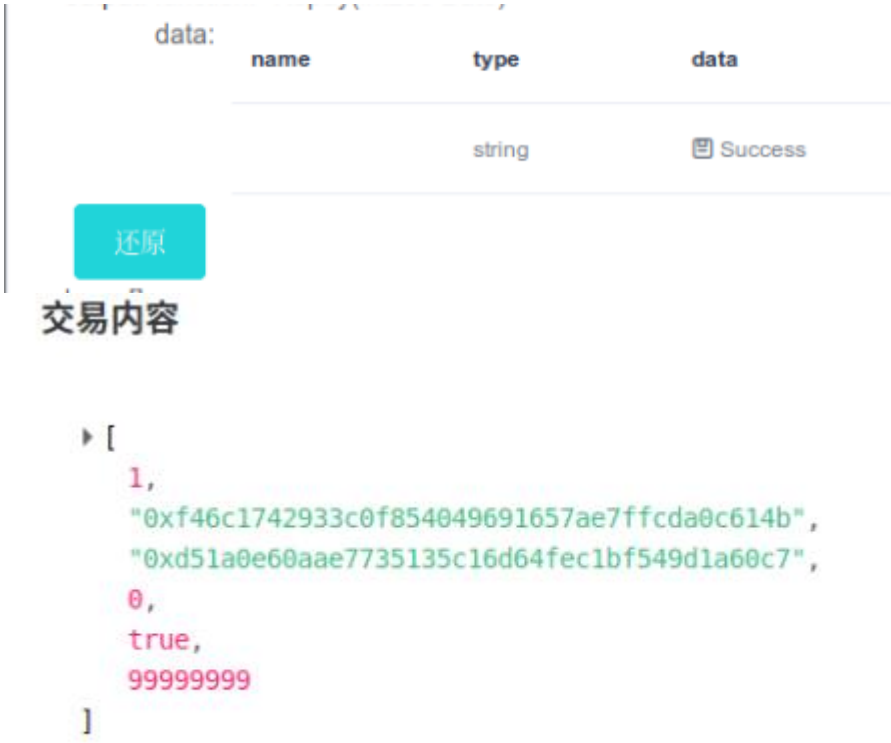
查询 receipt[2]可以看到 To 的地址已经变成了 Bank 的地址。

查询 C 的 balabce 里面已经有银行融资的 100

最后测试还款

输入 date 表示还款期限

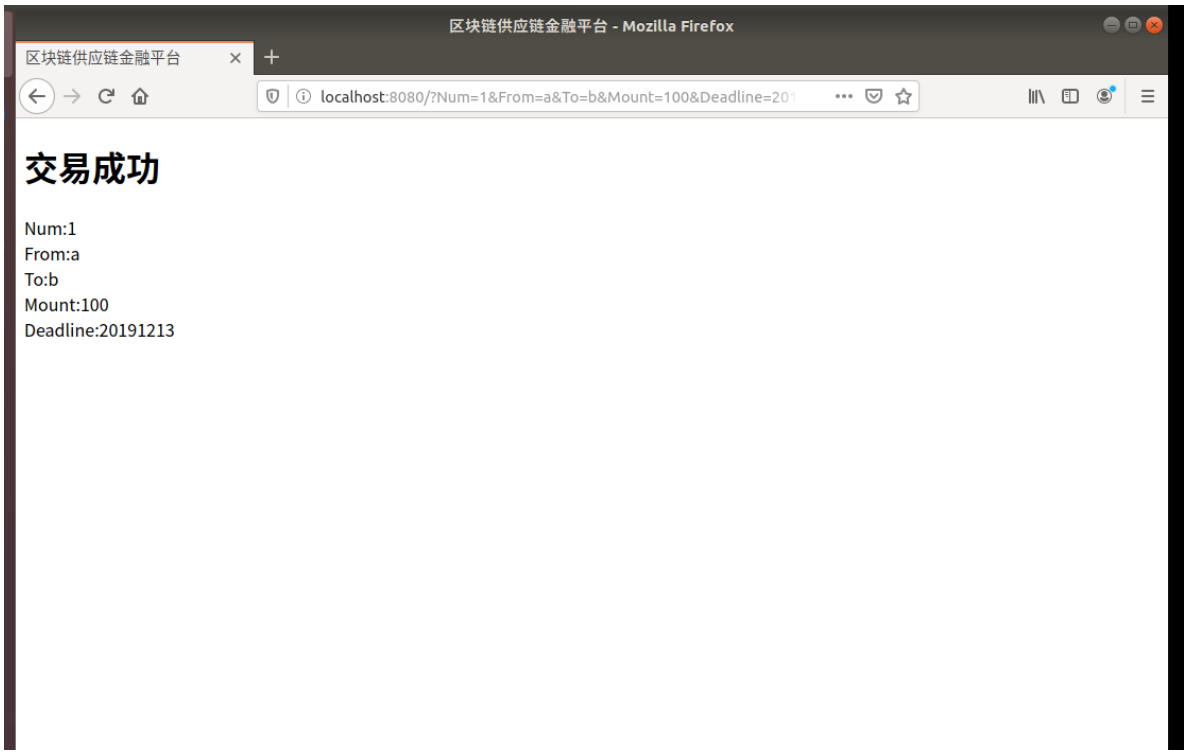
还款成功，come 来自 A 的账单成功还请。



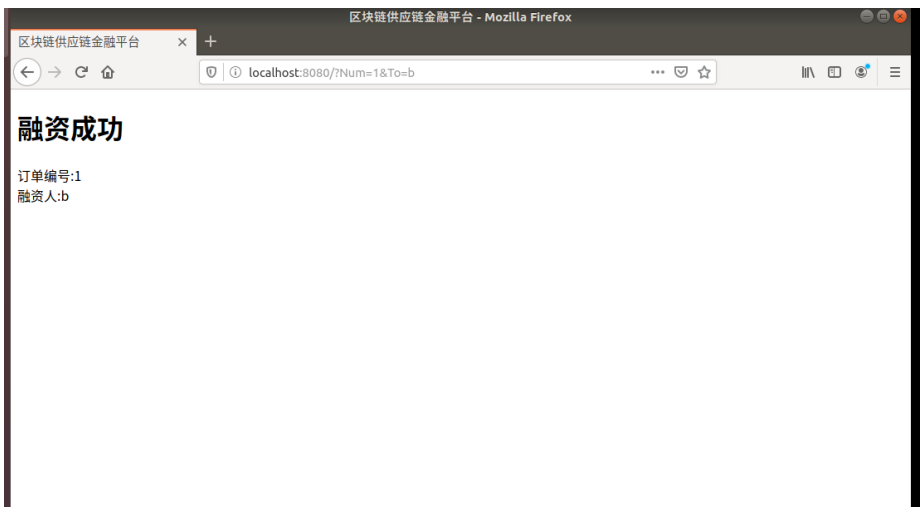
四、 界面展示

主界面

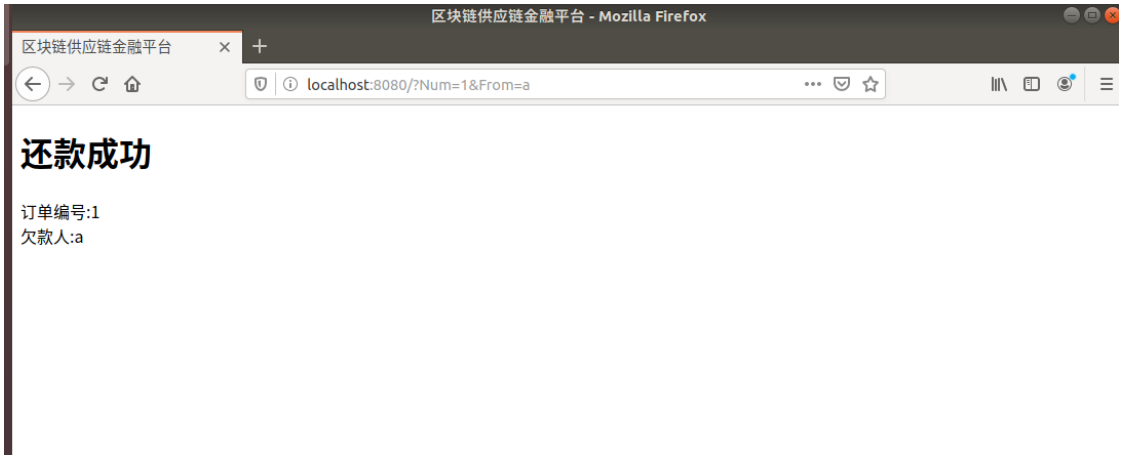
交易成功页面



融资成功界面



还款成功界面



实现效果视频：

请见视频

五、 心得体会

本次大作业由于合约实现比较简单，其实大部分时间是用在写前端与后端交互上面，由于不熟悉 node-js jdk，写这部分的时候挺费劲的，特别是还要与 fisco 的链进行交流，很多 api 都不怎么会用，而且 github 上关于 fiscobcos 的 nodejsjdk 的描述挺少的，需要自己阅读源码分析并使用，花费了大量时间。fiscobcos 是一个区块链底层平台，这次的大作业，让我更加深刻意识到区块链是一个分布式的账本，去中心化，改变了生产关系同时也是一种自动执行的共识，是信任的机器。

我们这次写的合约类似于一个分布式账本，链上的用户可以清楚的找到每一笔交易，降低了传统交易中由于信用不透明带来的经济成本，区块链应用的分布式账本技术带来的影响可能将与互联网革命一样巨大，我们有理由相信，未来区块链一定会大放异彩，在我们的生活中掀起一股改革的巨浪。