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	17343088	姓名	莫晓权
电话	15603018126	Email	1804851568@qq.com
开始日期	2019.12.01	完成日期	2019.12.13

项目背景

基于已有的开源区 块 链 系 统 FISCO-BCOS

（https://github.com/FISCO-BCOS/FISCO-BCOS），以联盟链为主，开发基于区块链或区块链智能合约的供应链金融平台，实现供应链应收账款资产的溯源、流转。

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。

实现功能：

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。

方案设计

存储设计、数据流图、核心功能介绍（文字+代码）

前端： 负责界面的展示，用户交互

后端： 使用 nodejs 实现后端，主要调用提供的 nodejs sdk，在链上创建表，每个用户注册的时候，都会将账户上链到表中，而交易，转账，借款等，都通过表的数据变化实现，因为表是在链端，所以每次交易都会被记录。

链端： 使用命令创建链，使用 nodejs sdk 中的 deploy 部署表到链端

后端的实现：

首先是注册登陆功能，用户名被作为 key 存储在链中，必须要有唯一性。默认的余额是 500，并且密码也需要存储在链端，但是与名字一起经过加密。

此为对前端传来的账户密码信息进行报错

```
if (url_info.pathname == '/insert' && req.method == "POST"){

    var str = "";//接收数据用
    var str2 = ""
    var str3 = ""
    var value_str
    req.on('data',function(data){
        var temp = decodeURIComponent(data);
        let datatemp2 = temp.split('&');
        str += datatemp2[0].split('=')[1];
        str2 += datatemp2[1].split('=')[1];
        str3 = "500";
        console.log(str);
        console.log(str2);
    });
```

插入前必须要判断账户的唯一性：

```
req.on('end', () => {

    crudService.desc("t_assets").then(tableInfo => {
        let table = new Table(tableInfo.tableName, str, tableInfo.valueFields, tableInfo.optional);

        crudService.select(table, ret).then(value=>{
        // console.log("aaaaaaaaaaaaaaaaaa")
        // res.write('<!DOCTYPE html><html><head><meta charset="utf-8"><title>select</title></head><body><form
        value_str = JSON.stringify(value);
        if(typeof value_str != "undefined" && value_str != null && value_str != "" && value_str != "[]"){
            console.log(str + "has exited");
            // console.log(value_str);
        }
```

插入上链：

```
let entry = new Entry();
for (let index in fieldNames) {
    entry.put(fieldNames[index], fieldValues[index]);
}

crudService.insert(table, entry).then(value=>{
    console.log(value)
    if(value == "1"){
        console.log("insert successfuuly")
        res.end()
    }
});
```

至此，注册的用户已经插入了链端。

转账功能，调用 sdk 中的 update 函数，对表中数据的金钱字段进行更改：
注意仅当金钱足够的时候才进行转账：

```
    if(parseInt(value["money"]) >= parseInt(trade_number)){
        fieldValues[1] = value['credit'];
        fieldValues[0] = (parseInt(value["money"]) - parseInt(trade_number)).toString(
        console.log("my_credit: " + value['credit'])
        console.log("my_mon: " + fieldValues[1])
        let entry = new Entry();


        for (let index in fieldNames) {
            entry.put(fieldNames[index], fieldValues[index]);
        }

        crudService.update(table, entry, ret).then(value=>{
            console.log(value)
            if(value != "0"){
                console.log("update successfuuly")
            }
        });
    }
 })
```

实现逻辑就是，发出账户金钱减少，对方账户金钱增加：

```
fieldValues_trade[0] = (parseInt(value["money"]) + parseInt(trade_number)).toString();
```

```
    fieldValues[0] = (parseInt(value["money"]) - parseInt(trade_number)).toString();
```

查询余额功能，主要调用 sdk 的 select 方法进行实现，通过对本账户的查询，返回账户的信息：

```
req.on('end', () => {
    // var obj = JSON.parse(str)
    let tableName = "t_assets";
    // let key = obj["account"];
    // let condition = obj["condition"];
    crudService.desc(tableName).then(tableInfo => {
        let table_com = new Table(tableName, tableInfo.key , tableInfo.valueFields, tableInfo.optional);
        console.log(table_com);
    console.log(tableInfo)

        let table = new Table("t_assets", str, tableInfo.valueFields, tableInfo.optional);
        var outValue = crudService.select(table, ret).then(value=>{
```

另外，还有另一张表是用来存储账单信息的，正常交易不会存储，但是当出现“借钱行为”的时候，则会记录，并且可以查看。

借出钱：

与支付的实现是类似的，但是有新的一点就是，需要对借出，借入进行记账：

```
crudService.desc("bill").then( tableInfo => {
    let table = new Table(tableInfo.tableName, your_account, tableInfo.valueFields, tableInfo.optional);
    let table2 = new Table(tableInfo.tableName, trade_account, tableInfo.valueFields, tableInfo.optional);

    let fieldNames = tableInfo.valueFields.split(',');
    let fieldValues = new Array("string", "string","string");
    let fieldValues_trade = new Array("string", "string","string");
    fieldValues[0] = trade_account
    fieldValues[1] = "lend"
    fieldValues[2] = trade_number;

    fieldValues_trade[0] = your_account
    fieldValues_trade[1] = "borrow"
    fieldValues_trade[2] = trade_number;

    let entry = new Entry();
    for (let index in fieldNames) {
        entry.put(fieldNames[index], fieldValues[index]);
    }
    crudService.insert(table, entry).then(value=>{
        console.log(value)
        if(value == "1"){
            console.log("insert successfuuly")
            res.end()
        }
```

查看借出账单，当银行判断你的融资可能的时候，可以查看你的借出账单，从而实现快速决定：

```
req.on('end', () => {
    // var obj = JSON.parse(str)
    let tableName = "bill";
    // let key = obj["account"];
    // let condition = obj["condition"];
    crudService.desc(tableName).then(tableInfo => {
        let table_com = new Table(tableName, tableInfo.key , tableInfo.valueFields, tableInfo.optional);
        console.log(table_com);
    console.log(tableInfo)

        let table = new Table("bill", str, tableInfo.valueFields, tableInfo.optional);
        var outValue = crudService.select(table, ret).then(value=>{
        res.writeHead(200,{'Content-Type':'html'});
        res.write('<!DOCTYPE html><html><head><meta charset="utf-8"><title>select</title></head><body><for
        res.end(JSON.stringify(value));
```

前端：

前端主要是写 html 网页，包含输入框以及提交按钮，提交到后端实现。见 table.htm

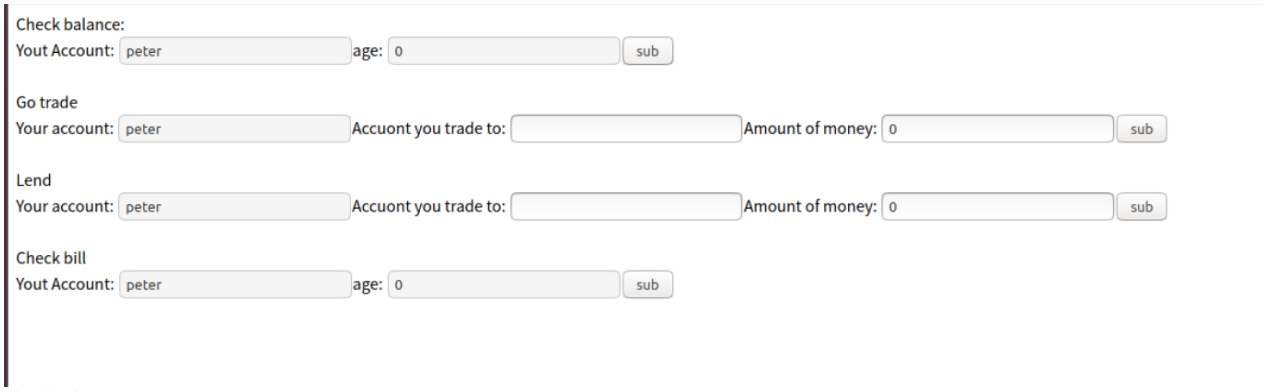
链端：

链端主要是依靠 nodejs sdk 调用 deploy 函数进行部署，请见 deploy.js，部署内容为在链上创建两个表，一个用来记录账户信息，一个用来记录交易账单。

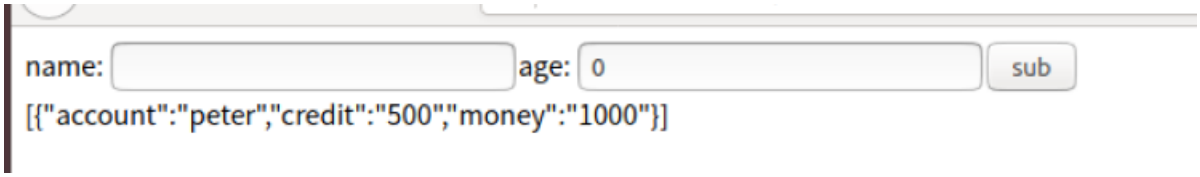
功能测试

创建账户：

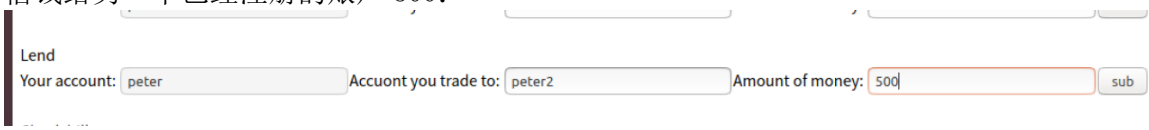
登进去后的功能界面：



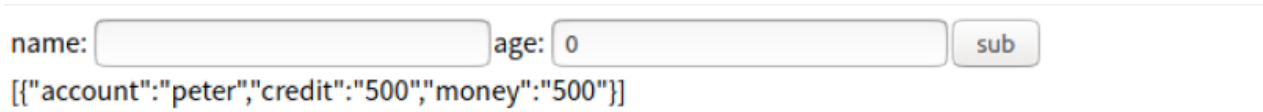
查询余额：



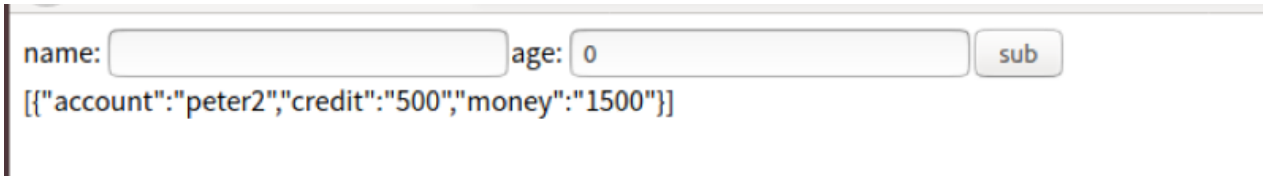
借钱给另一个已经注册的账户 500：



再次查看余额：

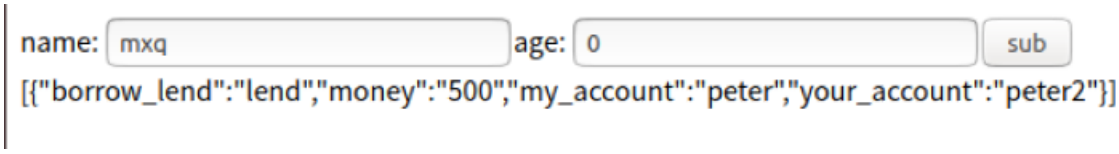


看另一位用户的余额，增加了 1500：



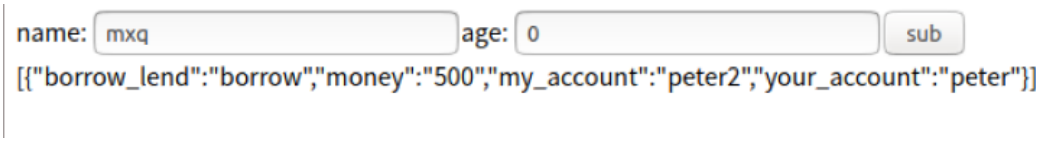
产看交易凭据：

对 peter：



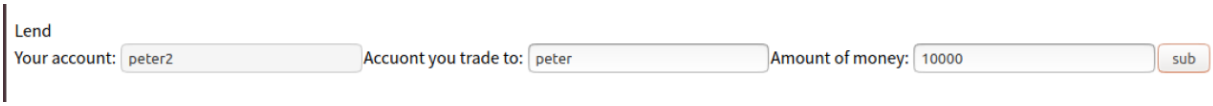
显示借出 peter2 500

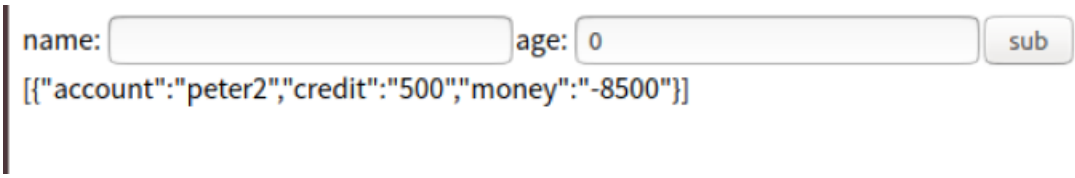
对 peter2：

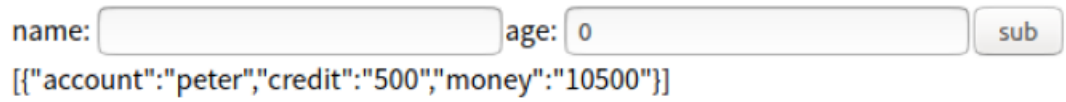


显示借入 peter 500；

而借钱可以变负债：







至此，基本的交易功能，记账功能，借账功能都已经实现。

实验心得：

因为之前对区块链的概念不是很清晰，所以在项目刚开始的时候走了许多弯路，一开始从智能合约的编写，到最后使用 nodejs sdk 实现，之间参考了许多 sdk 的源码，观察它的使用方法。最后终于是把记账，交易，融资结账的功能实现了。在这个过程中也学会了许多，对个人的自学能力的提升也很有益处，总而言之，在这次实验中我受益匪浅，既了解了区块链的运作，也学会了搭建，实现基本功能，从而实现更复杂的功能。