中山大学数据科学与计算机学院本科生实验报告

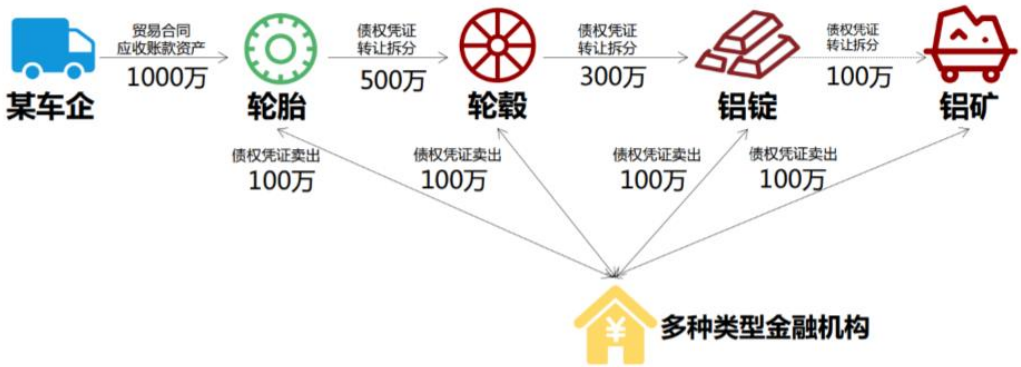
(2019 年秋季学期)

课程名称：区块链原理与技术　　任课教师：郑子彬

年级	17	专业（方向）	软件工程
学号	17343120	姓名	吴涛
电话		Email	a529995805@qq.com
开始日期	2019.12.1	完成日期	2019.12.13

一、 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。

二、 方案设计

智能合约的设计依据官方网站上的构建第一个区块链应用的智能合约设计，在一些方面进行了修改。

存储设计：

```
function createTable() private {
    TableFactory tf = TableFactory(0x1001);

    tf.createTable("t_asset", "account", "asset_value");
}
```

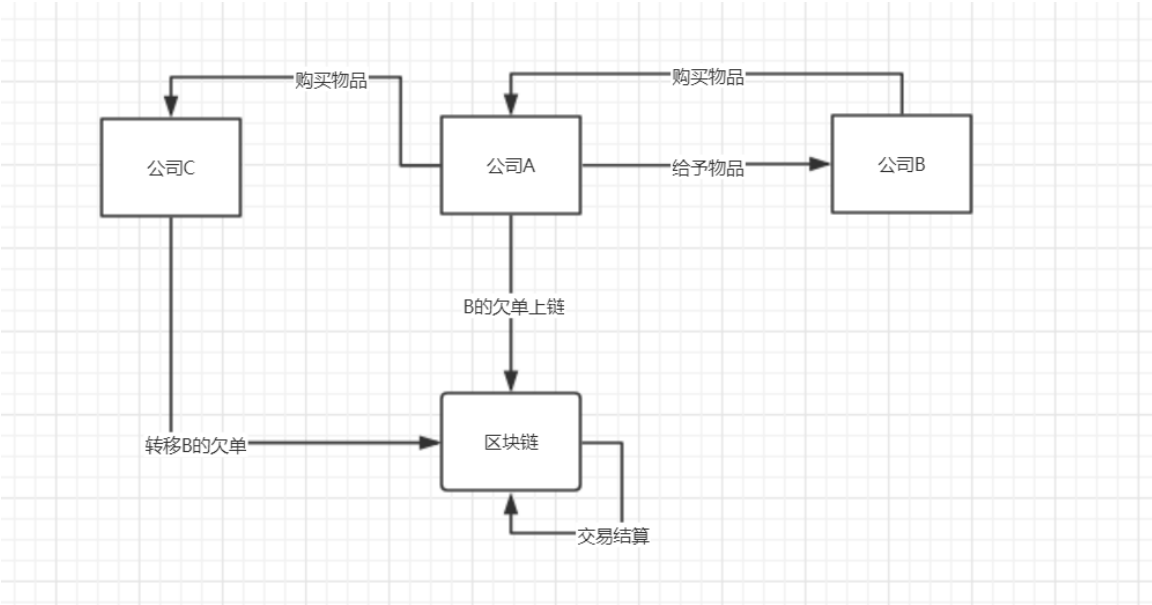
使用表来进行数据存储，每个账户对应有其欠单的值，欠单为正，结算时用户的余额会增加，欠单为负，结算时用户的余额会减少

```
address constant master = 0xbAF9163Da71F944b2B26a8CBe6484563cbE3EB47;

mapping(string=>address) str_addr;
mapping(address=>string) addr_str;
string[] accountList;
```

设置为管理员地址，给予高的权限，并且通过 mapping 实现用户到地址的映射，使用 accountList 链来存储依据注册的用户

数据流图：



核心功能介绍（文字+代码）:

1.用户注册

需要查询用户是否存在，存在则无法创建，并判断是否对表进行数据插入是否成功，失败创建也不能成功。

```
function register(string account, uint256 asset_value) public returns(int256){
    int256 ret_code = 0;
    int256 ret= 0;
    uint256 temp_asset_value = 0;
    // 查询账户是否存在
    (ret, temp_asset_value) = select(account);
    if(ret != 0) {
        Table table = openTable();

        Entry entry = table.newEntry();
        entry.set("account", account);
        if(msg.sender == master){
            entry.set("asset_value", int256(asset_value));
        }else{
            entry.set("asset_value", int256(0));
        }
        // 插入
        int count = table.insert(account, entry);
        if (count == 1) {
            // 成功
            ret_code = 0;
            str_addr[account] = msg.sender;
            addr_str[msg.sender] = account;
            accountList.push(account);
        } else {
            // 失败? 无权限或者其他错误
            ret_code = -2;
        }
    } else {
        // 账户已存在
        ret_code = -1;
    }

    emit RegisterEvent(ret_code, account, asset_value);

    return ret_code;
```

只有在链上注册了的用户才能进行交易，并且每个刚注册的用户的欠单为 0，只有管理者能够注册欠单数目为正的用户。

2.查询欠单

```
function select(string account) public constant returns(int256, uint256) {
    // 打开表
    Table table = openTable();
    // 查询
    Entries entries = table.select(account, table.newCondition());
    uint256 asset_value = 0;
    if (0 == uint256(entries.size())) {
        return (-1, asset_value);
    } else {
        Entry entry = entries.get(0);
        return (0, uint256(entry.getInt("asset_value")));
    }
}
```

打开表，获取对应账户的数据，如果找不到则返回-1。

3.转移欠单

```
Table table = openTable();

Entry entry0 = table.newEntry();
entry0.set("account", from_account);
entry0.set("asset_value", int256(from_asset_value - amount));

int count = table.update(from_account, entry0, table.newCondition());
if(count != 1) {
    ret_code = -5;

    emit TransferEvent(ret_code, from_account, to_account, amount);
    return ret_code;
}

Entry entry1 = table.newEntry();
entry1.set("account", to_account);
entry1.set("asset_value", int256(to_asset_value + amount));

table.update(to_account, entry1, table.newCondition());
```

打开表，利用新的 Entry 进行数据的更新，实现两个账户之前的欠单转移

4.欠单结算

```
function Settlement() public returns(int256){
    Table table = openTable();

    for(uint i = 0; i < accountList.length; i++){

        Entry entry = table.newEntry();
        entry.set("account", accountList[i]);
        entry.set("asset_value", int256(0));

        table.update(accountList[i], entry, table.newCondition());
    }

    emit SettlementEvent(0);
    return 0;
}
```

将各个用户的余额按照欠单进行变化后，将各个用户的欠单都重新设置为 0。

三、 功能测试

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh deploy
 deploy Asset success, contract address is 0x777efe21930e5693a22dfd64ca43a8e2ad8
043d9
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ ^C
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh
 Usage :
   bash asset_run.sh deploy
   bash asset_run.sh query    asset_account
   bash asset_run.sh register asset_account asset_amount
   bash asset_run.sh transfer from_asset_account to_asset_account amount
   bash asset_run.sh settlement


examples :
   bash asset_run.sh deploy
   bash asset_run.sh register  Asset0  10000000
   bash asset_run.sh register  Asset1  10000000
   bash asset_run.sh transfer  Asset0  Asset1 11111
   bash asset_run.sh query Asset0
   bash asset_run.sh query Asset1
   bash asset_run.sh settlement
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh register  As
set0  10000000
```

四、 界面展示

Welcome to FISCO BCOS

account asset_value Register

account select

from_account to_account amount Transfer

Settlement

五、 心得体会

本次区块链实验对我来说是极为艰难的，因为从前并没有尝试写过后端的代码，所以对此有些无从下手，虽然在官方的网站上有 asset-app 项目，但是还是没有能够完成，因为并不会使用 java 来构建相应的服务器，来与前端进行沟通，不过前端倒是由于在服务计算课程中有所了解，使用 vue 实现了一个简单的界面。

总体来说，在本次大作业中还是学会了许多的东西，如如何在利用命令行编译和部署智能合约，完成链端的实现部分。我对区块链也有了更深的了解，虽然最后没能够成功的实现完整的应用十分另人遗憾，不过以后还是会尝试去完成它的。