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：区块链原理技术

任课教师：郑子彬

年级	17	专业（方向）	软件工程
学号	17343065	姓名	李秀祥
电话	15178187219	Email	1034186061@qq.com
开始日期	12月1日	完成日期	12月12日

一、项目背景

1、传统供应链金融 传统中，只有与获得银行信用认可的企业直接关联的企业，才可以使用被认可企业的的欠条进行和其他企业的发展交易等操作。但是无疑不利于中小型企业的发展，因为只要不是直接与被认证企业有关系，其他企业都是很难相信这个企业的。 2、区块链+传统供应链金融 引入第三方可信机构，来证实交易和单据的真实性，行为使没有信用认可的公司之间能否顺利的进行交易，让核心企业的应用可以传递到供应链的下游企业

二、方案设计

这个项目中，最需要解决的其实应该是信用问题，因为没有信用导致了后续操作的失败，那么解决的方法可以有两种：1、使用自己信用 2、借用他人信用。接下来的任务其实就是围绕这个这两个目的或者其中一个就可以了。 我们首先可以引入一个第三方绝对可信机构（仲裁机构）。这个块就叫做银行吧。银行的职能有以下几点

1. 根据企业的资产，来判断企业可以“打白条”的额度，银行并按照额度的等级给企业不同的信用额度。
2. 银行会定期的查看企业的资产详情，但是银行查看的标准仅限于该企业现有的信用等级，如果企业仍然可以拿出来和他信用等级等同的资产，那么银行会保留这个等级，如果不能拿出来相应的资产，银行会根据他目前的资产而修改他的信用等级。但是企业无需拿出来所有的资产，这个取决于企业自己的意思（这一个功能可能在后续操作中不会体现，但是如果当在真实的系统中，必须要有）
3. 银行根据企业的要求可以响应企业的借款请求
4. 银行作证交易的产生，并且在收款、借款纪录上留下自己的印记，这个印记只能有银行可以拥有

对于企业来说，它的功能有下：

1. 向银行申请信用凭证，这个需要拿相应的资金去让银行评价
2. 可以根据自己的信用凭证，向产品相关企业赊账，并且会产生一个赊账纪录，这个就叫他“白条”吧。假设A向B赊账1000，那么A的信用额度减少1000，而B的信用额度加上1000。 白条必须经过银行才能产生，当然喽如果两家企业关系特别好就可以不通过银行，但是这样的白条不能向其他不接受的企业使用。白条里面记录的信息会很多，这个在后面再说。这一个过程其实很复杂，需要

考虑的东西很多：如果现在白条有300元，但是我要借400元，而且我的信用凭证可借200，这个时候怎么办，这涉及到交叉部分，这种情况我会在后续考虑。

3. 企业可以根据自己的收据票单，向银行融资
4. 还款，这个功能让人头大

接下来说一下整个的流程 对于银行和公司，我们可以使用表存放，我这里建立两个表格：

```
function createTable() private {
    TableFactory tf = TableFactory(0x1001);
    // 资产管理表, key : account, field : asset_value
    // |  资产账户(主键)      |     资产金额       |
    // |--------------------- |--------------------|
    // |        account       |    asset_value     |
    // |----------------------|--------------------|
    //
    // 创建表
    tf.createTable("t_asset2", "account", "asset_value");
     // 资产管理表, key : account, field : asset_value
    // |  资产账户(主键)      |     信用额度       |
    // |--------------------- |--------------------|
    // |        account       |    credit     |
    // |----------------------|--------------------|
    //
    tf.createTable("t_credit2", "account", "credit");
}
```

然后就是注册用户，对于每一个用户我给了他两个初始值：现有资产、信用额度。分别存放到两个表里面。对于银行来说，初始化时直接就按照最大限度的值进行初始化即可。

```
if(ret != 0) {
    Table table = openTable();

    Entry entry = table.newEntry();
    entry.set("account", account);
    entry.set("asset_value", int256(asset_value));
    // 插入
    int count = table.insert(account, entry);

    table = openTable1();
    entry = table.newEntry();
    entry.set("account",account);
    entry.set("credit", int256(credit));
    count = table.insert(account, entry);
    if (count == 1) {
        // 成功
        ret_code = 0;
    } else {
        // 失败? 无权限或者其他错误
        ret_code = -2;
    }
} else {
    // 账户已存在
    ret_code = -1;
}
```

同时还需要可以查询用户的信息，传入参数：用户名，返回参数：现有资产、信用额度。

```
} else {
    Entry entry = entries.get(0);
    uint256 res1 = uint256(entry.getInt("asset_value"));

    table = openTable1();
    entries = table.select(account, table.newCondition());
    entry = entries.get(0);
    uint256 res2 = uint256(entry.getInt("credit"));
    return (0, res1, res2);
}
```

接下来就是进行交易，交易传递参数：交易双方的账户名，交易金额。首先呢需要判断交易双方账户是否存在，这个需要对表格进行查询：

```
// 转移账户是否存在?
(ret, from_asset_value, from_credit) = select(from_account);
if(ret != 0) {
    ret_code = -1;
    // 转移账户不存在
    emit TransferEvent(ret_code, from_account, to_account, amount);
    return ret_code;

}

// 接受账户是否存在?
(ret, to_asset_value, to_credit) = select(to_account);
if(ret != 0) {
    ret_code = -2;
    // 接收资产的账户不存在
    emit TransferEvent(ret_code, from_account, to_account, amount);
    return ret_code;
}
```

然后还需要判断交易的金额是否对账户溢出：

```
if(from_credit < amount) {
    ret_code = -3;
    // 转移资产的账户金额不足
    emit TransferEvent(ret_code, from_account, to_account, amount);
    return ret_code;
}

if (to_credit + amount < to_credit) {
    ret_code = -4;
    // 接收账户信用
    emit TransferEvent(ret_code, from_account, to_account, amount);
    return ret_code;
}
```

接下来就可以将交易双方的信用额度进行加减然后在表格中进行更新了：

```
    Table table = openTable1();

    Entry entry0 = table.newEntry();
    entry0.set("account", from_account);
    entry0.set("credit", int256(from_credit - amount));
    // 更新转账账户
    int count = table.update(from_account, entry0, table.newCondition());
    if(count != 1) {
        ret_code = -5;
        // 失败? 无权限或者其他错误?
        emit TransferEvent(ret_code, from_account, to_account, amount);
        return ret_code;
    }

    Entry entry1 = table.newEntry();
    entry1.set("account", to_account);
    entry1.set("credit", int256(to_credit + amount));
    // 更新接收账户
    table.update(to_account, entry1, table.newCondition());

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

    return ret_code;
```

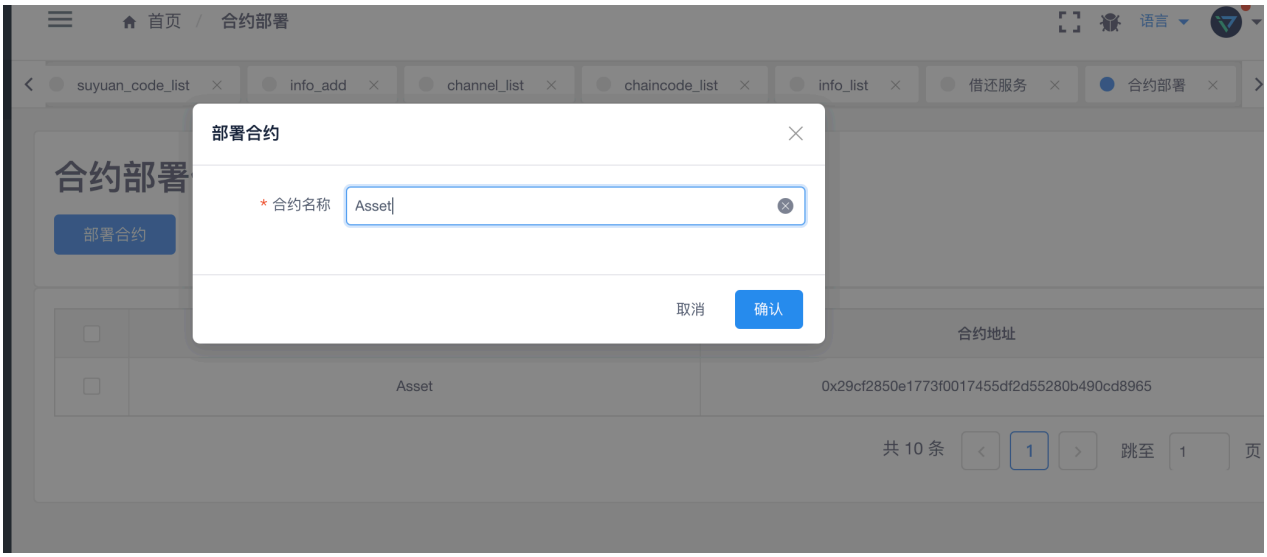
接下来还有一个功能就是将信用额度转化为资产，也就是提现的过程。这个过程为了防止企业取巧，所以提现完后的信用额度不能小于初始的额度。这样就十分巧妙的解决了链上“扒帐”后不好处理的情况。

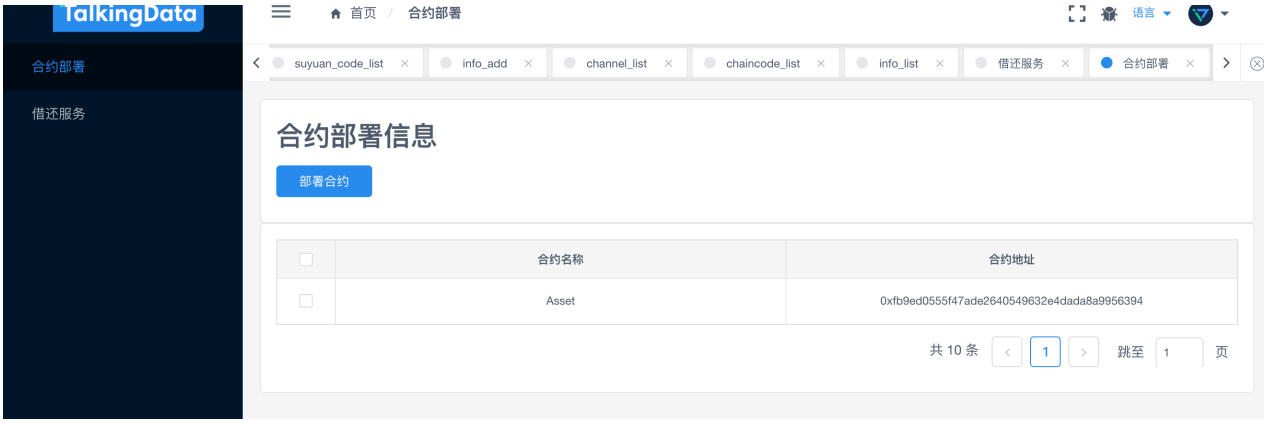
```
    Table table = openTable1();

    Entry entry0 = table.newEntry();
    entry0.set("account", from_account);
    entry0.set("credit", int256(from_credit - amount));
    // 更新转账账户
    int count = table.update(from_account, entry0, table.newCondition());
    if(count != 1) {
        ret_code = -3;
        // 失败? 无权限或者其他错误?
        return ret_code;
    }
    table = openTable();
    Entry entry1 = table.newEntry();
    entry1.set("account", from_account);
    entry1.set("asset_value", int256(from_asset_value + amount));
    // 更新接收账户
    table.update(from_account, entry1, table.newCondition());
    return ret_code;
}
```

三、功能测试

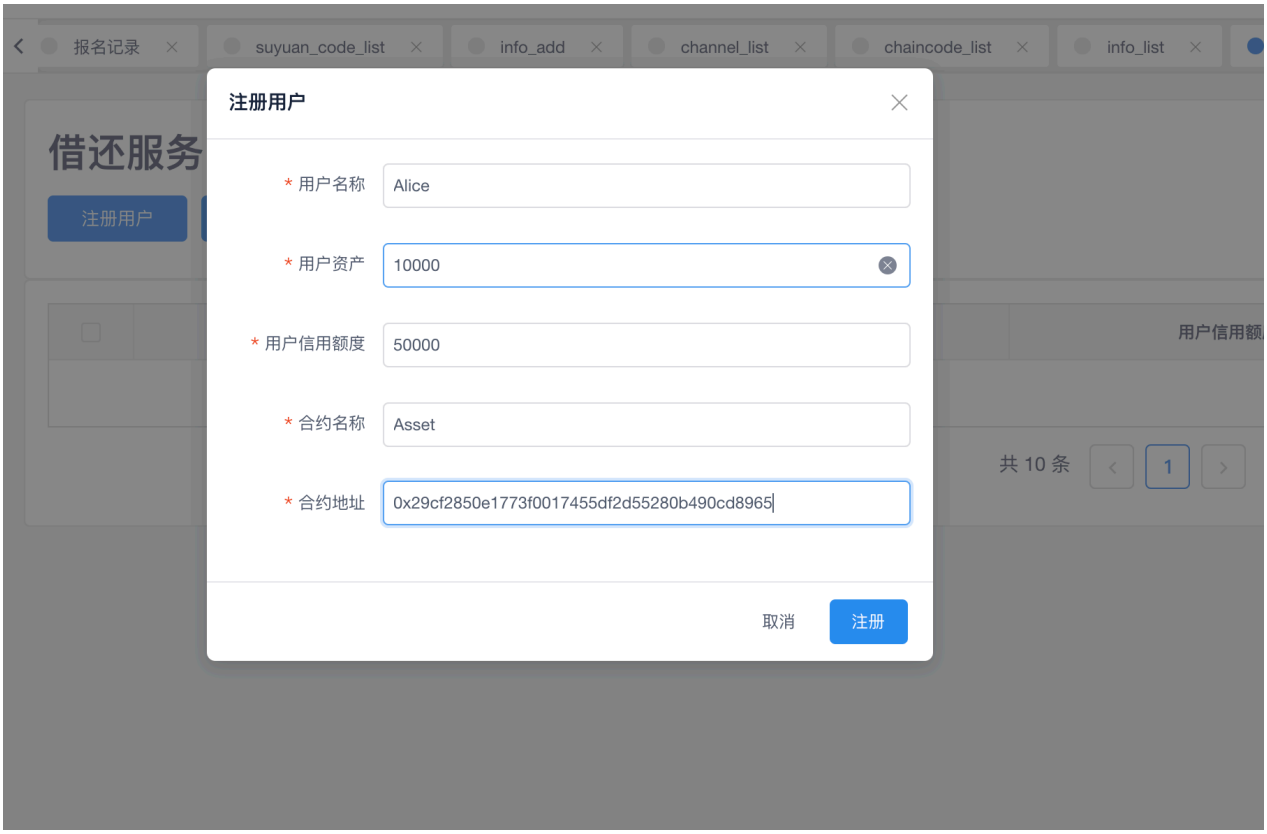
1、部署合约





2、注册用户

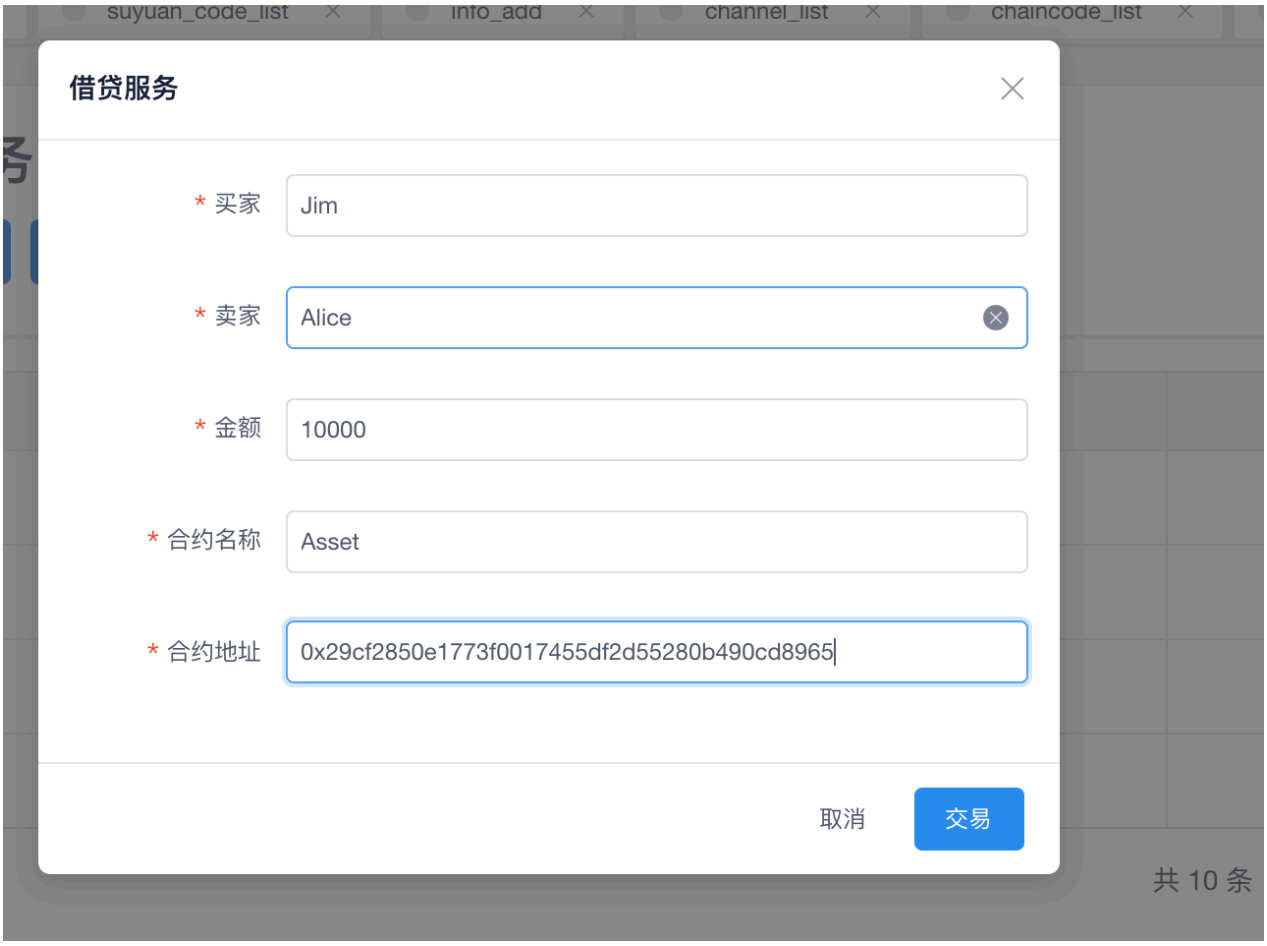
创建`Alice`为例



在创建两个用户名字为：Bob、Jim。同时我初始化了一个银行名为：bank，他的资产和信用都设置为最大。

3、交易操作

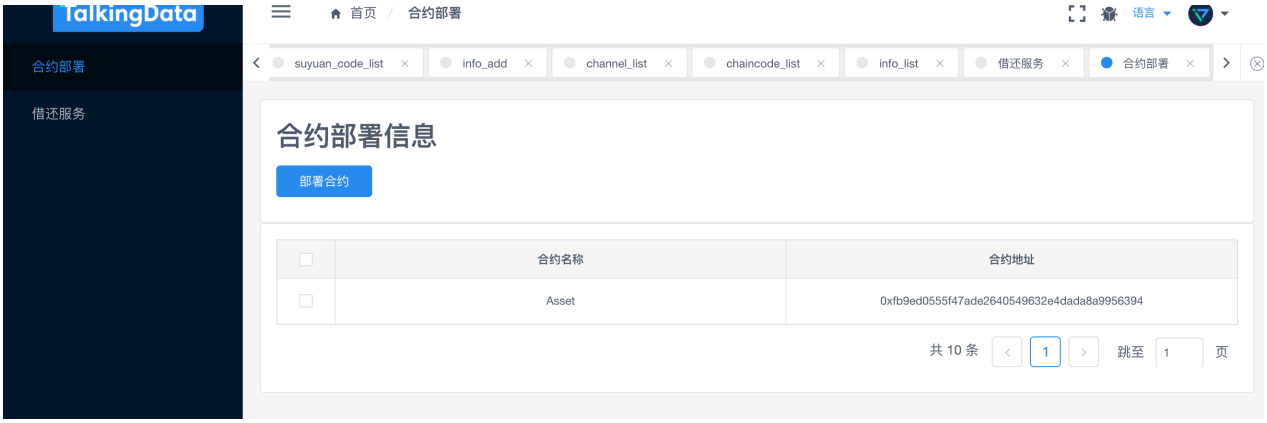
为了方便区分交易的双方，这里使用卖家和买家来区分。Jim向Alice使用信用买了10000东西。那么Jim的信用将减少10000，信用变成78888。



结果正确。

四、界面展示

前端：



后端：

```
temp Asset
temp Asset
temp { account: 'Alice',
  property: '10000',
  credit: '50000',
  contract: 'Asset',
  addr: '0x29cf2850e1773f0017455df2d55280b490cd8965',
  func: 'register' }
temp { account: 'bank',
  contract: 'Asset',
  addr: '0x29cf2850e1773f0017455df2d55280b490cd8965',
  func: 'select' }
temp { account: 'Bob',
  property: '5000',
  credit: '50000',
  contract: 'Asset',
  addr: '0x29cf2850e1773f0017455df2d55280b490cd8965',
  func: 'register' }
temp { account: 'Jim',
  property: '6666',
  credit: '88888',
  contract: 'Asset',
  addr: '0x29cf2850e1773f0017455df2d55280b490cd8965',
  func: 'register' }
temp { fromAccount: 'Jim',
  toAccount: 'Alice',
  amount: '10000',
  contract: 'Asset',
  addr: '0x29cf2850e1773f0017455df2d55280b490cd8965',
  func: 'transfer' }
temp Asset
temp { account: 'Jim',
  contract: 'Asset',
  addr: '0x42fcf5fc9fa440ce0d0311d53dda8bd0d6500ca2',
  func: 'select' }
```

五、心得体会

这一次作业其实我是很迷糊的。第一开始走了错的方向，去研究了fisco官网上的浏览器部分，然后后来才发现，自己研究错了，应该先研究如何将SOL代码转化为后端API，然后就开始看nodejs sdk部分，看如何将链端代码转化为后端API，第一开始看很迷糊，因为cli.js给出来的API，我调用后没办法得到返回值，他自己输出了出来，所以当我发现这种情况，心里暗道：“糟了”。然后我开始转战java sdk，嗯，不出意外，看一会还是觉得nodejs sdk适合我。然后就下定决心使用nodejs写前端，我开始一点一点“溯源”cli的API调用的位置，然后将其提取出来放到我自己的API里面，然后就一点一点地使用postman对接口进行调试。但是当我开始写前端，发现传到后端的数据还需要进行特殊的处理，因为传进来的数据是按照键值的字典，还被封装成了string，所以想了很久才将需要的内容分离出来。最后做出来了这个东西，感觉还可以做到更好，因为调用时，我为了省事，合约的名字和地址都没有省略，所以这会给用户带来很大的不便。

最后，感觉这个大作业很有趣，确实很锻炼人的意志，因为我有好几次要放弃。最后还是坚持了下来，同时也懂了很多nodejs的内容，感觉最后收获还是很大的。