

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	17343015	姓名	陈伟松
电话	15113757645	Email	982899377@qq.com
开始日期	2019-10-20	完成日期	2019-12-12

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。

二、方案设计

存储设计、数据流图、核心功能介绍(文字+代码)

存储设计：

通过在合约中存储变量以达到数据上链

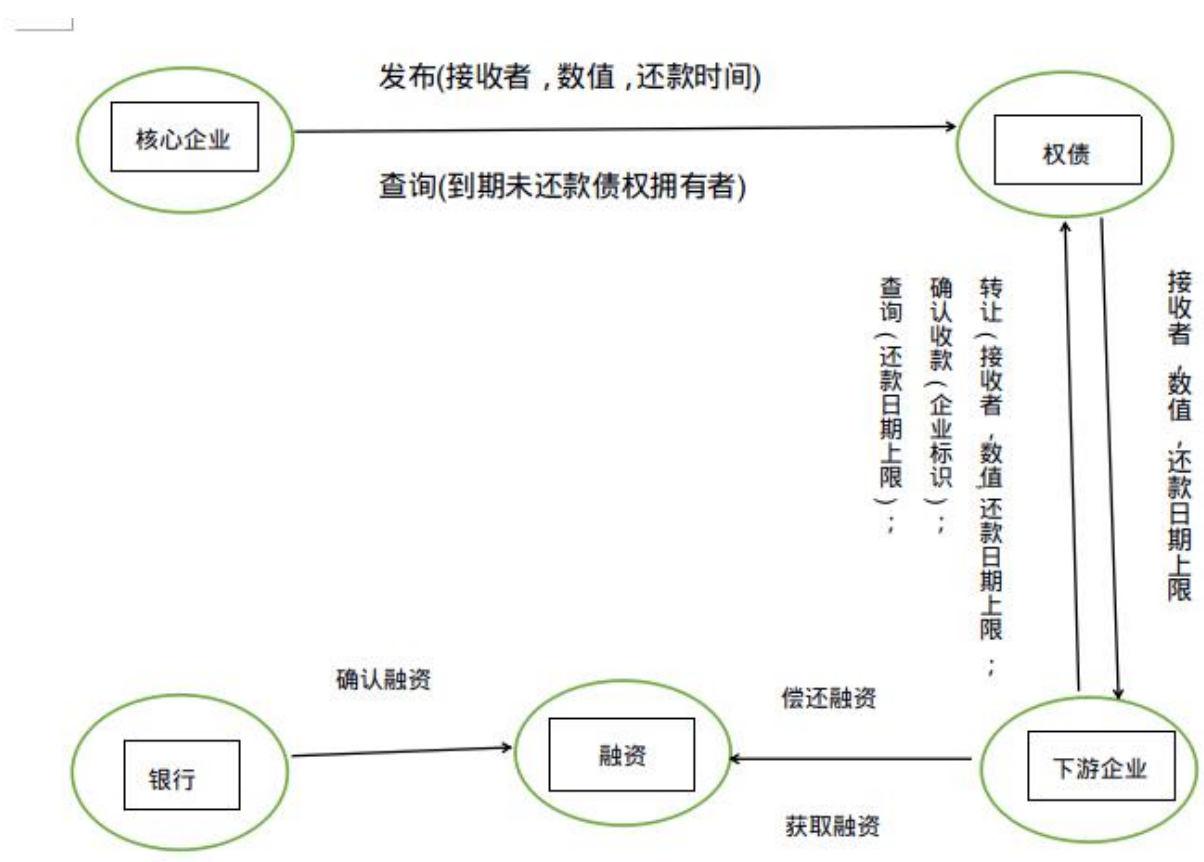
在 Link.sol 中定义两个结构体：

1. receipt：定义核心企业向下游企业签发的应收账款单据,其中包括下游企业的地址，欠款多少以及还款到期时间以及 used, payed 作为功能实现的控制参数
2. downStreamCompany：定义下游企业结构，主要包括企业 id（自增唯一），企业名称，持有核心企业的债券的下标（对应所有 receipt 构成的数组，这是以还款时间从早到晚排序好的），已经融资的数量，以及融资的控制参数

定义变量：

1. bank(address): 唯一的一个银行
2. coreCompany(address): 唯一的核心企业
3. totReceipts(receipt[]): 所有的账款单据集合
4. inDebt(uint): 当前核心企业发出单据总额
5. downStreamCompanies(mapping(uint=>downStreamCompany)): 一个关于下游企业 id 以及 downStreamCompany 结构的映射
6. unpaid(bool[100]): 核心企业是否欠某个 id 债的标志，每次查看前由 CheckUnpaid 函数更新

数据流图



核心功能介绍与实现：

- 实现功能流程：
- 1. 实现采购商品—签发应收账款 交易上链。核心企业在链下向下游企业采购商品后在链上添加储存着包括接收者的目的地址，签发金额，以及还款时间的单据。
 - 2. 实现应收账款的转让上链。下游企业可以发起对自己的债权的转让，在转让时可以要求只能转让在某还款时间之前的单据，同时转让方也会尽可能把具有较晚还款时间的单据转让。
 - 3. 利用应收账款向银行融资上链。一个下游企业记录着一条关于自己的已融资总金额信息，在融资前，银行可以查询下游企业的已融资金额以及它拥有核心企业某还款时间之前的应收账款来判断是否给它融资。决定允许融资后，在链下交易，然后由企业增加融资总金额，相应的偿还融资也是如此，但是这里企业每次修改融资总金额后都需要一次来自银行的确认才能获取，否则无法继续执行融资操作。
 - 4. 应收账款支付结算上链。核心企业获取它在当时对某下游企业有欠款后查看当前对它的欠款总额，在链下偿还对某下游企业到当日为止的所有到期权债后，下游企业相应地在链上去除它所持有的所有已经支付了的账单。

定义函数：
内部函数：

1. addReciept：将账单添加到 totReciepts 中
 2. pushIdxWFTDebtTime：将相应账单下标加到下游企业的结构体的账单数组中
 3. popIdxWFTDebtTime: 将相应账单下标从下游企业的结构体的账单数组中去除
- 外部函数：
1. AddDownStreamCompany: 加入一个下游企业
 2. GetCompanyName：根据 id 获取企业名称
 3. SignAndIssue：定义核心企业向下游企业签发应收账款函数
 4. GetRight：获取某个下游企业所拥有的在多长时间以内就能获得付款的债权
 5. TransferRight：定义转让债权函数, debtTime 可以设置要求获得指定还款时间上限的债权；失败返回当前 from 所拥有的债权，成功返回 0
 6. GetFinance：获取下游企业已获融资总额
 7. BankCheckFinance：银行确认企业的融资总额更改有效
 8. CompanyAddFinance：企业获得融资更改融资总额
 9. CompanyPayFinance：企业支付融资更改融资总额
 10. ConfirmPaied：应收账款支付结算函数
 11. CheckUnpaied：更新核心企业到期未偿还标志的函数

执行流程：

修改/model/python_sdk/interateWithChain.py 的第一行为主机 python 安装位置，在/执行 python3 model/python_sdk/console.py deploy Link fe53c8c63fefce538070fc7498f3883f05626f94 dceddee3792f46e33aed9dd906a57bef66d89108 部署合约，其中 Link 是合约名，后两个分别已经生成的 test,test1 用户的地址,在这里分别代表核心企业和银行，在/执行 go run main.go，打开 localhost:8080 即可开始操作

后端与链端交互：

使用 fisco bcos 提供的 python-sdk 并参考自带的 console.py，编写 /model/python_sdk/interateWithChain.py，interateWithChain.py 实现生成新账户和发送对应于部署合约 Link 的事务的接口函数，然后将事务结果按要求输出到 stdout,每个函数的调用通过命令参数选择，后端用 exec 指令执行 interateWithChain.py，用命令参数指定选择的操作以及相应接口函数的参数，然后使用管道读取命令执行的标准输出，解析后就可以做为后端的输出，要是能提供更加细致，更友好的 json rpc 就不用这么绕弯子了，我觉得与其支持多样的 sdk，定义出更友好的交互格式或许会更方便。

三、 功能测试

测试分链端测试和后端+前端测试

链端测试

部署上链





启动 webase，创建四个用户，对应核心企业，银行，下游企业 1，下游企业 2，记下它们地址以便后续使用

```
weltloose@Linux2:~/下载/fisco/webase-deploy$ python3 deploy.py startNode
===== FISCO-BCOS start... =====
try to start node0
try to start node1
node0 start successfully
node1 start successfully
===== FISCO-BCOS end... =====
weltloose@Linux2:~/下载/fisco/webase-deploy$ python3 deploy.py startWeb
===== WeBASE-Web start... =====
[sudo] weltloose 的密码:
===== WeBASE-Web start success! =====
===== WeBASE-Web end... =====
weltloose@Linux2:~/下载/fisco/webase-deploy$ python3 deploy.py startManage
===== WeBASE-Node-Manager start... =====
===== WeBASE-Node-Manager start success! =====
===== WeBASE-Node-Manager end... =====
weltloose@Linux2:~/下载/fisco/webase-deploy$ python3 deploy.py startFront
===== WeBASE-Front start... =====
===== WeBASE-Front start success! =====
===== WeBASE-Front end... =====
```


用户名称	用户ID	用户描述	用户公钥地址信息	用户状态	操作
dsCmp2	 700005		 0xe0b8c14a1e06fec9...	正常	修改
dsCmp1	 700004		 0xbccbbbff6559b673e...	正常	修改
bank	 700003		 0xd08a15d8e551601...	正常	修改
coreCompany	 700002		 0xfe1ab9ba9b5e1a07...	正常	修改

编译合约


Supply.sol


```
1 pragma solidity ^0.4.4;
2
3 contract Supply {
4     // 定义核心企业向下游企业签发的应收账款 单据
5     struct receipt {
6         address to; // 欠谁
7         bool used; // 判断这个元素是否被占用
8         uint amount; // 这张单据代表欠多少
9         uint endTime; // 还款到期时间
10        bool payed; // 标记该账单是否已经被偿还
11    }
12
13    enum FinanceCondition {
```

contractName  Supply

abi

 [{"constant":false,"inputs":[{"name":"from","type":"address"}, {"name":"amount","type":"uint256"}],"name":"CompanyPayFinance","outputs":[{"payable":false,"stateMutability":"nonpayable","type":"function"}],["constant":false,"inputs":[{"name":"owner","type":"address"}, {"name":"debtTime","type":"uint256"}],"name":"GetRight","outputs":

bytecodeBin

 608060405234801561001057600080fd5b506040516040806123e68339810180604052810190808051906020019092919080519060200190929190505050600060048190555081600160006101000a81548173fffffffffffffffffffffffffffffffff021916908373ffffffffff160217905550806000806101000a81548173fffffffffffffffffffffffffffffffff021916908373ffffffffff160217905550505061230f806100d76000396000f306080604052600436106100a4576000357c0100000000

部署,部署需要初始参数,coreCompany 和 bank,找到对应地址添上,用户则无所谓,不在本项目定义范围内

选择用户

×

用户:

dsC2

▼

参数:

_coreCompany

2c12ebe87e

_bank

22ee07d973fd469e0e4

❗ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：array1,array2。string等其他类型也不用加上引号。

取消

确定

两个下游企业用户每个创建一个下游企业对象

发送交易

×

合约名

称: Supply

合约地

址: 0x01542a1bef773a0b7b3dbd6ea61c2

❗

用户:

dsC1

▼

方法:

function

▼

AddDownStre

▼

参数:

name

a

❗ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：array1,array2。string等其他类型也不用加上引号。

取消

确定

功能一测试

核心企业调用 SignAndIssue 函数,指定对象,数值,以及欠债时间(只能由核心企业调用)给 dsCmpy2 发送 100 限定到期时间从当前开始 200 秒

发送交易

×

合约名

称: Supply

合约地

址: 0x01542a1bef773a0b7b3dbd6ea61c2

❗

用户:

coreCompany

▼

方法:

function

▼

SignAndIssue

▼

参数:

to

2

amount

100

debtTime

200

❗ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：array1,array2。string等其他类型也不用加上引号。

取消

Typora

再给 dsCmpy2 发送 100 限定到期时间 1000 秒

发还义务

合约名

称: Supply

合约地

址: 0x01542a1bef773a0b7b3dbd6ea61c2

用户: coreCompany

方法: function SignAndIssue

参数:

to	2
amount	100
debtTime	1000

如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如：arry1,arry2。string等其他类型也不用加上引号。

取消 Typora

查询 dsCmpy2 500 秒内能收到款

发送交易

合约名

称: Supply

合约地

址: 0x01542a1bef773a0b7b3dbd6ea61c2

用户: dsC2

方法: function GetRight

参数:

owner	2
debtTime	500

如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如：arry1,arry2。string等其他类型也不用加上引号。

取消 确定

output: function: GetRight(uint256 owner,uint256 debtTime)

data:

name	type	data
	uint256	100

还原

查询 1000 秒后

×

logs: Π

发送交易

×

合约名

称: Supply

合约地

址: 0x01542a1bef773a0b7b3dbd6ea61c2 ⓘ

用户: dsC2 ▾

方法: function ▾ TransferRight ▾

参数:

from	2
to	1
amount	100
debtTime	500

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：arry1,arry2。string等其他类型也不用加上引号。

取消

确定

查看双方的持有 500 秒内债权

发送交易

×

合约名

称: Supply

合约地

址: 0xacabf4770436bcd2073804b59ddb8 ⓘ

id=1:

用户: dsC2 ▾

方法: function ▾ GetRight ▾

参数:

owner	1
debtTime	500

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：arry1,arry2。string等其他类型也不用加上引号。

取消

确定

X

id=2:

×

id=2 1000 秒内债权

data:	name	type	data
		uint256	 10

当然这里可以让一次转让由多次凭证组合而成
核心企业向 1 签发金额 200 到期时间 500 秒的凭证,然后 1 向 2 转让 600 秒内 230 金额,查看 1 跟 2 在 600 秒内的金额

×

确定

发送交易

×

合约名

称: Supply

合约地

址: 0xacabf4770436bcd2073804b59ddb8 ⓘ

用户: dsC1 ▾

方法: function ▾ TransferRight ▾

参数:

from	1
to	2
amount	230
debtTime	600

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如：arry1,arry2。string等其他类型也不用加上引号。

取消

确定

结果:

发送交易

×

合约名

称: Supply

合约地

址: 0xacabf4770436bcd2073804b59ddb8 ⓘ

用户: dsC2 ▾

方法: function ▾ GetRight ▾

参数:

owner	1
debtTime	600

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如：arry1,arry2。string等其他类型也不用加上引号。

取消

确定

×

还原

✕

还原

先给企业增加融资

发送交易

×

合约名

称: Supply3

合约地

址: 0x3015e40eabe29db679df05b271ec4 ⓘ

用户: dsC2 ▾

方法: function ▾ CompanyAddFi ▾

参数: from 2
amount 100

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如: arry1,arry2。string等其他类型也不用加上引号。

取消

确定

银行确认

发送交易

^

合约名

称: Supply3

合约地

址: 0x3015e40eabe29db679df05b271ec4 ⓘ

用户: dsC2 ▾

方法: function ▾ BankCheckFir ▾

参数: from 2

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如: arry1,arry2。string等其他类型也不用加上引号。

取消

确定

获得当前融资总额

发送交易

×

合约名

称: Supply3

合约地

址: 0x3015e40eabe29db679df05b271ec4 ⓘ

用户: dsC2 ▾

方法: function ▾ GetFinance ▾

参数: from 2

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如: arry1,arry2。string等其他类型也不用加上引号。

取消

确定

- 14 -

交易内容

```
{  
  transactionHash: "0xe90503a01db8292cf74887b9c367f42e72f5ec7b5600be9c3a176b9ed  
    b600aec"  
  transactionIndex: 0  
  blockHash: "0x0ea971153e914d6615eda07fbf609289f83430d28b204dd399323d8f13d21f4  
    a"  
  blockNumber: 47  
  gasUsed: 29959  
  contractAddress: "0x00000000000000000000000000000000000000000000000000000000"  
  root: null  
  status: "0x0"  
  from: "0x49b98a4ed6ddc1dd4b393098b321c523789f069d"  
  to: "0x3015e40eabe29db679df05b271ec4e5f1e41231c"  
  input: "0x4c3f674400000000000000000000000000000000000000000000000000000000  
    00002000000000000000000000000000000000000000000000000000000000000000"  
  output: function: GetRight(uint256 owner,uint256 debtTime)  
    data:  
      name          type          data  
  
      uint256       330
```

[还原](#)

也可以由核心企业主动更新欠债人员,然后以此查询

- 15 -

发送交易

×

合约名

称: Supply3

合约地

址: 0x3015e40eabe29db679df05b271ec4 ⓘ

方法: function ▼ unpaied ▼

参数: 1

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如: arry1,arry2。string等其他类型也不用加上引号。

取消

确定

交易内容

▶ [true]

copy

其中 unpaied 参数是下游企业 id,返回内容是有无到期权债
双方在链下完成还款后由权债拥有者备注账单已偿还,这时再获取当前过期权债则返回 0

发送交易

➤

合约名

称: Supply3

合约地

址: 0x3015e40eabe29db679df05b271ec4 ⓘ

用户: dsC2 ▼

方法: function ▼ ConfirmPaiec ▼

参数: to 2

ⓘ 如果参数类型是数组，请用逗号分隔，不需要加上引号，
例如: arry1,arry2。string等其他类型也不用加上引号。

取消

确定

×

称: Supply3

址: 0x3015e40eabe29db679df05b271ec4 ⓘ

用户: dsC2

方法: function GetRight

参数:	owner	2
-----	-------	---

debtTime	0
----------	---

❶ 如果参数类型是数组，请用逗号分隔，不需要加上引号，例如：arry1,arry2。string等其他类型也不用加上引号。

取消

[illegible]

data:	name	type	data
		uint256	0

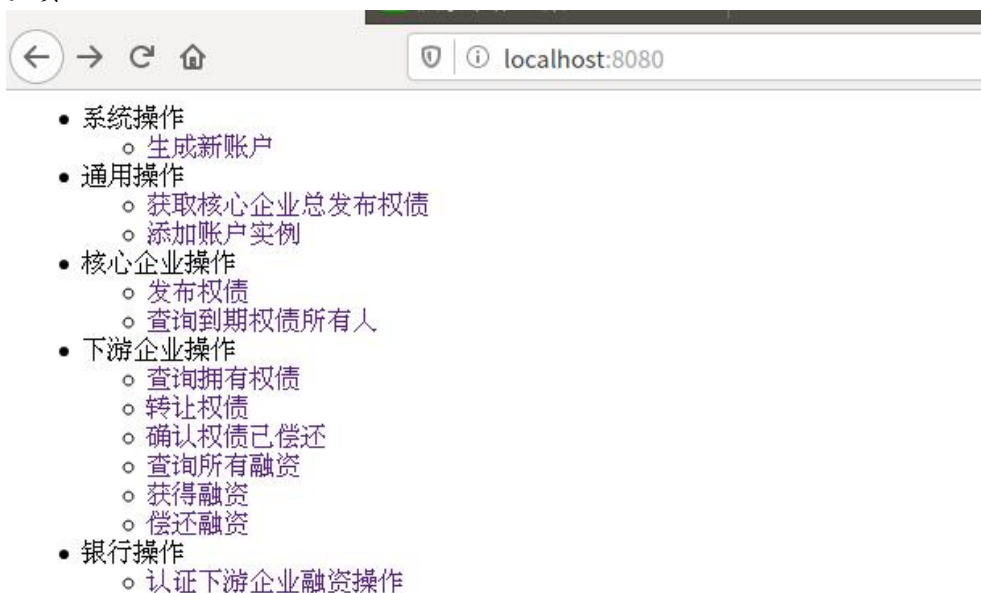
后端+前端测试

```
weltloose@Linux2:~/blockChain$ go run main.go
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery
middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in prod
ction.
- using env: export GIN_MODE=release
- using code: gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /public/*filepath      --> github.com/gin-gonic/gin.(*Rou
erGroup).createStaticHandler.func1 (3 handlers)
[GIN-debug] HEAD   /public/*filepath      --> github.com/gin-gonic/gin.(*Rou
erGroup).createStaticHandler.func1 (3 handlers)
[GIN-debug] GET    /                       --> github.com/gin-gonic/gin.(*Rou
erGroup).StaticFile.func1 (3 handlers)
[GIN-debug] HEAD   /                       --> github.com/gin-gonic/gin.(*Rou
erGroup).StaticFile.func1 (3 handlers)
[GIN-debug] POST   /api/GenerateAccount    --> github.com/Weltloose/blockCha
/controller.GenerateAccount (3 handlers)
[GIN-debug] POST   /api/InDebt             --> github.com/Weltloose/blockCha
/controller.InDebt (3 handlers)
[GIN-debug] POST   /api/AddDownStreamComp --> github.com/Weltloose/blockCha
```

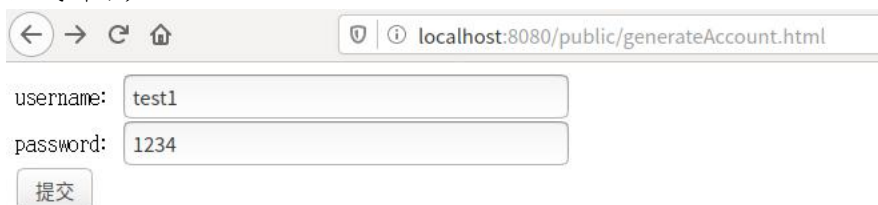
主页面



← → ↺ 🏠 | 📄 ⓘ localhost:8080

- 系统操作
 - 生成新账户
- 通用操作
 - 获取核心企业总发布权债
 - 添加账户实例
- 核心企业操作
 - 发布权债
 - 查询到期权债所有人
- 下游企业操作
 - 查询拥有权债
 - 转让权债
 - 确认权债已偿还
 - 查询所有融资
 - 获得融资
 - 偿还融资
- 银行操作
 - 认证下游企业融资操作

生成新账户

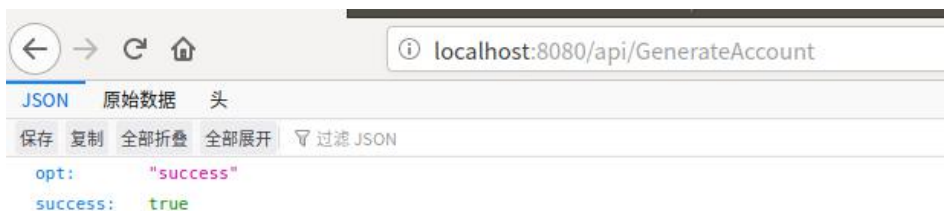


← → ↺ 🏠 | 📄 ⓘ localhost:8080/public/generateAccount.html

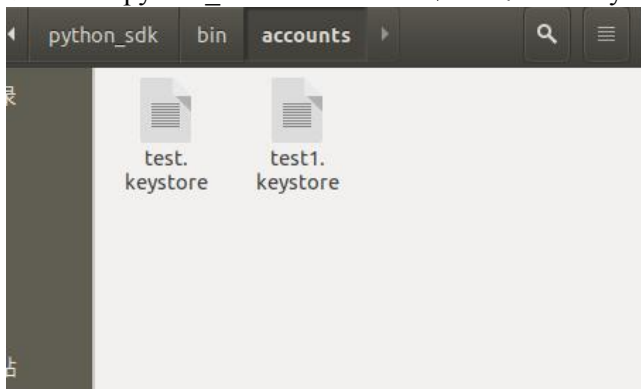
username:

password:

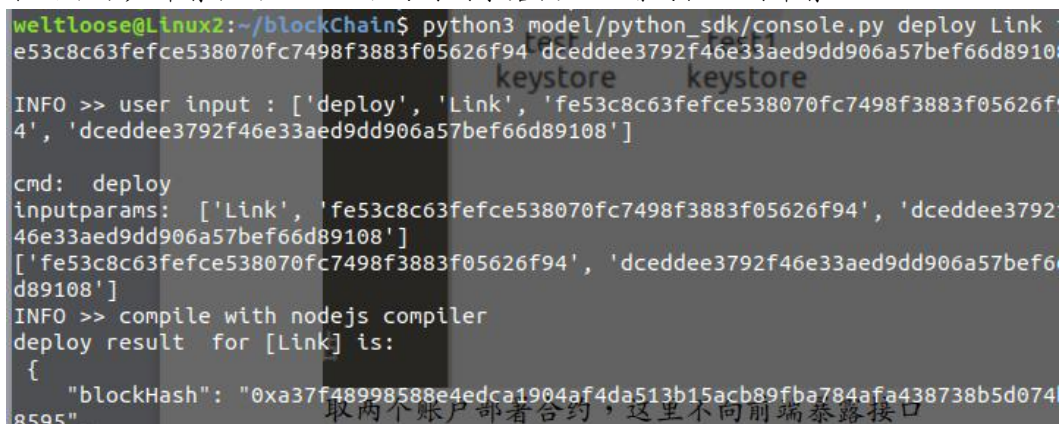
结果：页面返回成功



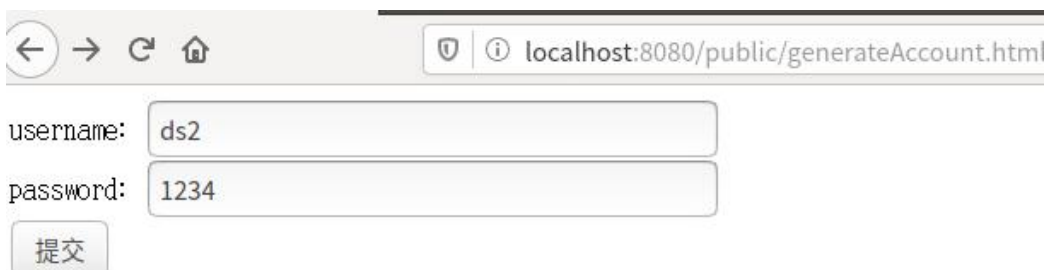
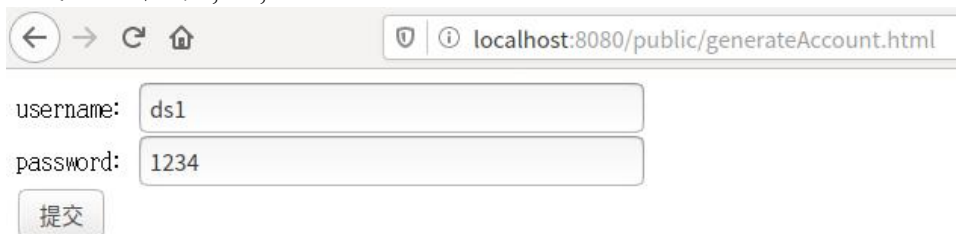
在 model/python_sdk/bin/accounts 中生成 test1.keystore



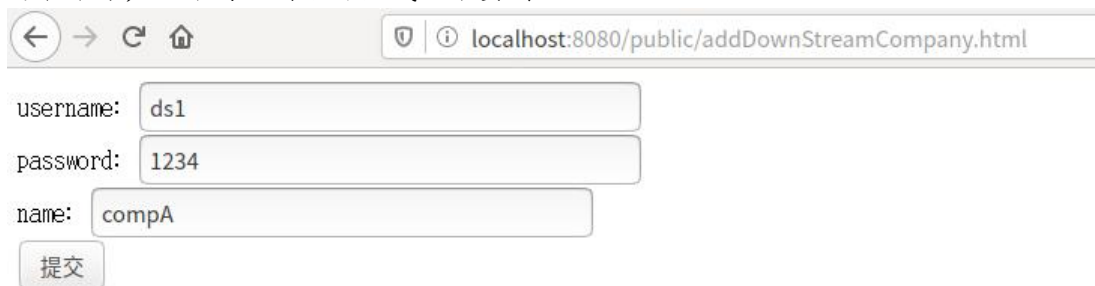
取两个账户部署合约，这里不向前端暴露接口，需要在后端部署



生成两个新账户,ds1,ds2，



为各个账户以下游公司名称生成一个实例



localhost:8080/public/addDownStreamCompany.html

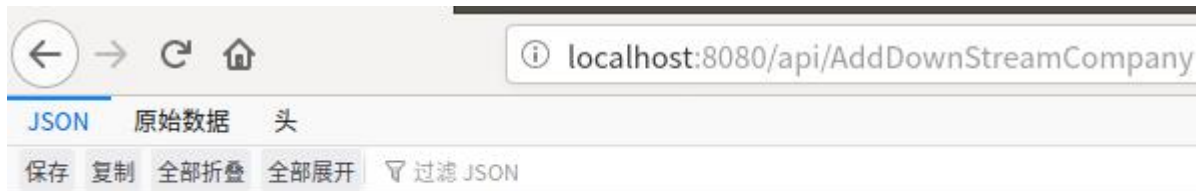
username: ds1

password: 1234

name: compA

提交

返回的是该公司在链上的唯一标识符



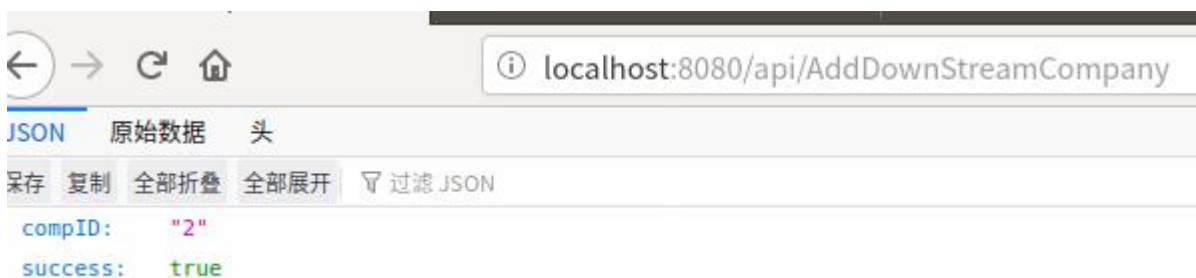
localhost:8080/api/AddDownStreamCompany

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

compID: "1"

success: true



localhost:8080/api/AddDownStreamCompany

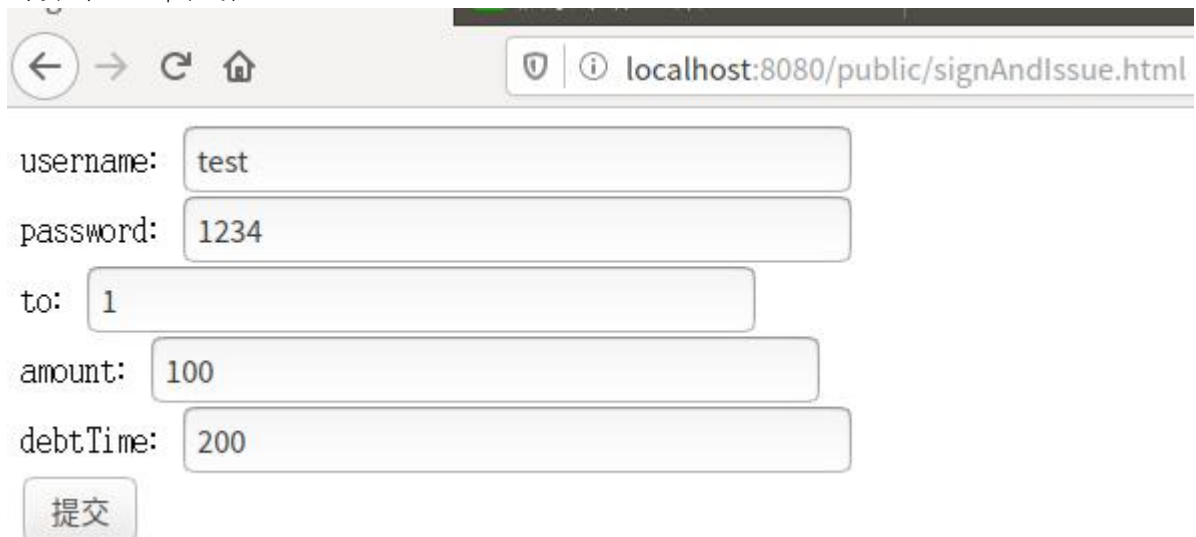
JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

compID: "2"

success: true

向实例 1 发布权债



localhost:8080/public/signAndIssue.html

username: test

password: 1234

to: 1

amount: 100

debtTime: 200

提交

其中，username,password 的内容要求是核心企业的账户，to 是前面提到的公司的唯一标识符，amount 是数值，debtTime 是债权到期时间（单位为 s）



查询总发布债权，使用任一有效账户均可

localhost:8080/public/inDebt.html

username: ds1

password: 1234

提交



查询实例 1 拥有债权

localhost:8080/public/getRight.html

username: ds1

password: 1234

owner: 1

debtTime: 100

提交

这里 debtTime 是指的从当前开始 debtTime 时间内会到期的该实例所拥有权债总额



转让权债

localhost:8080/public/transferRight.html

username: ds1

password: 1234

from: 1

to: 2

amount: 50

debtTime: 100

提交

实例 1 向实例 2 转让 100s 内到期的权债，其中 username,password 是实例 1 所有者的账户信息

localhost:8080/api/TransferRight

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

success: true

实例 1 获取融资

localhost:8080/public/companyAddFinance.html

username: ds1

password: 1234

from: 1

amount: 100

提交

localhost:8080/api/CompanyAddFinance

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

success: true

银行确认

localhost:8080/public/bankCheckFinance.html

username: test1

password: 1234

from: 1

提交

localhost:8080/api/BankCheckFinance

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

success: true

实例 1 偿还融资

localhost:8080/public/companyPayFinance.html

username: ds1

password: 1234

from: 1

amount: 20

提交

localhost:8080/api/CompanyPayFinance

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

success: true

再次银行确认

查询实例 1 当前融资额

← → ↻ 🏠 localhost:8080/public/getFinance.html

username:

password:

from:

← → ↻ 🏠 localhost:8080/api/GetFinance

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

finance: "80"

success: true

标记确认偿还

← → ↻ 🏠 localhost:8080/public/confirmPaied.html

username:

password:

to:

Username,password 是债权所有者的账户信息，to 是相应实例，通过链下交易偿还后将当前所有到期权债标记偿还

← → ↻ 🏠 localhost:8080/api/ConfirmPaied

JSON 原始数据 头

保存 复制 全部折叠 全部展开 过滤 JSON

success: true

查询到期权债所有人

localhost:8080/public/getUnpaied.html

username: test

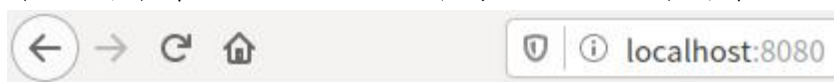
password: 1234

提交

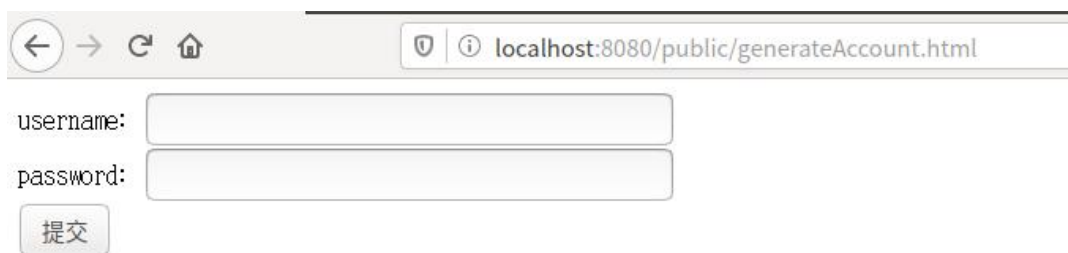


四、 界面展示

这里设计得简陋主要是觉得在生产环境中大部分隔一段时间才会执行其中的某以项操作，就很适合这种简洁明了的前端界面



- 系统操作
 - 生成新账户
- 通用操作
 - 获取核心企业总发布权债
 - 添加账户实例
- 核心企业操作
 - 发布权债
 - 查询到期权债所有人
- 下游企业操作
 - 查询拥有权债
 - 转让权债
 - 确认权债已偿还
 - 查询所有融资
 - 获得融资
 - 偿还融资
- 银行操作
 - 认证下游企业融资操作



A screenshot of a web browser window. The address bar shows 'localhost:8080/public/generateAccount.html'. Below the address bar, there are two input fields: 'username:' and 'password:'. Below the password field is a button labeled '提交' (Submit).

五、 心得体会

通过本次实验进一步了解了区块链技术在联盟链中的应用，学习区块链的基本部署，以及如何进行 fisco bcos 链端与后端的交互,并且独立实现了一个简单的区块链应用，算是令我对这门热门新奇的技术不再陌生。