

中山大学数据科学与计算机学院本科生实验报告

课程名称	区块链原理与技术	任课老师	郑子彬
年级	2017	专业（方向）	软件工程
学号	16305204	姓名	郑佳豪
电话	13144951811	Email	zhengjh35@mail2.sysu.edu.cn
开始日期	2019-12-10	完成日期	2019-12-13

项目地址: github.com/Jiahonzheng/Blockchain/tree/master/Supplier-Financing

项目背景

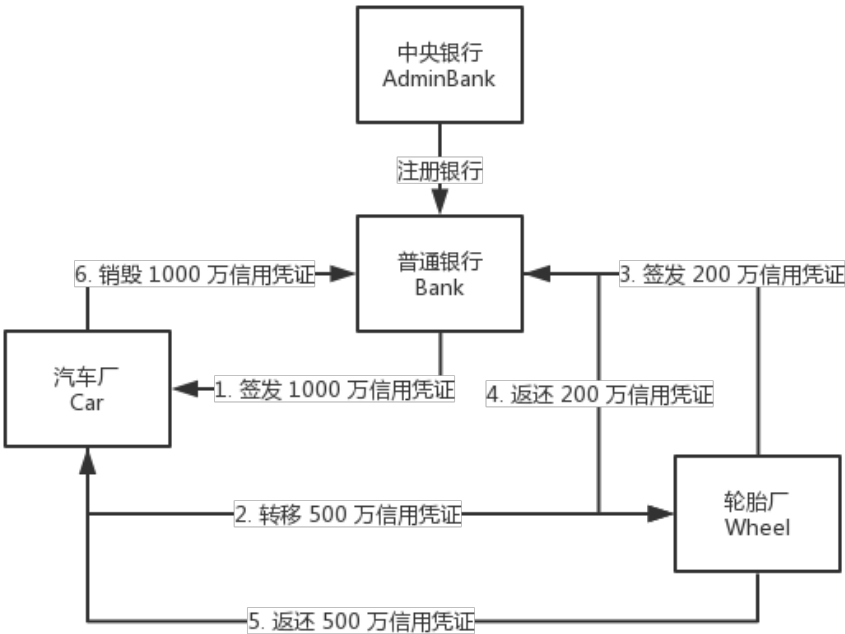
我们需要基于已有的开源区块链系统 FISCO-BCOS，以联盟链为主，开发基于区块链或区块链智能合约的供应链金融平台，实现供应链应收账款资产的溯源、流转。

根据实验要求，我们需要完成以下四项基本功能。

- 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

数据流图



存储设计

在合约设计中，我们使用 FISCO BCOS 的 [CRUD](#) 服务构建智能合约的存储，实现数据和合约的分离。在实现中，我们有两种类型的表：**账户表**和**交易表**。

账户表

我们使用 `t_company` 表作为智能合约的账户表，维护账户的公钥地址、统一社会信用代码、借入凭据总金额和借出凭证总金额，具体数据表定义如下。

名称	类型	定义
addr	address、主键	区块链公钥地址
uscc	string	18 位统一社会信用代码
inReceipts	int	借入凭据总金额
outReceipts	int	借出凭据总金额
type	int	账户类型

在实现中，我们将用户划分为 3 种：

- 中央银行：全局唯一，在部署合约时传入至合约中，可实现普通银行的注册。
- 普通银行：由中央银行注册，其可实现公司的注册。
- 公司

交易表

由于 FISCO BCOS 提供的 CRUD 服务是 K-V 数据库的封装，因此我们需要创建两个表（`t_in_receipt` 和 `t_out_receipt`），描述信用凭证的双向关系。`t_in_receipt` 和 `t_out_receipt` 的不同在于主键，前者的主键是 `debtee` 字段，后者是 `debtor` 字段。

名称	类型	定义
debtee	address	债权人公钥地址
debtor	address	债务人公钥地址
receiptId	int	全局信用凭证标识符
amount	int	凭据金额，以分为单位
deadline	int	截止日期，Unix 时间戳，以秒为单位

核心功能

智能合约代码地址：github.com/jiahonzheng/Blockchain/tree/master/Supplier-Financing/contracts

合约创建

在合约的构造函数中，我们实现了上述三个数据表的创建，同时插入中央银行的相关数据。

```
constructor(address _adminAddr, string uscc) {
    adminAddr = _adminAddr;
    // 创建表。
    TableFactory tf = TableFactory(0x1001);
    tf.createTable("t_company", "addr", "uscc,type,in_receipts,out_receipts");
    tf.createTable("t_in_receipt", "debtee",
        "debtor,debtee,receiptId,amount,deadline");
    tf.createTable("t_out_receipt", "debtor",
        "debtor,debtee,receiptId,amount,deadline");
    // 插入央行数据。
    insertCompany(adminAddr, uscc, 1, 1000000000000000000, 0);
}
```

银行注册

在设计和实现中，我们约定只有中央银行才可注册普通银行账户。

```
// 注册普通银行。
function registerBank(address addr, string uscc) public {
    // 鉴权：只有中央银行才可注册普通银行。
    require(msg.sender == adminAddr, "only the Admin Bank is allowed to register banks");
    insertCompany(addr, uscc, 2, 10000000000000, 0);
    emit RegistrationEvent(2, addr, uscc);
}
```

公司注册

在设计和实现中，我们约定只有普通银行才可注册公司账户。

```
// 注册公司。
function registerCompany(address addr, string uscc) public {
    string memory r1; int r2; int r3; int r4;
    (r1, r2, r3, r4) = selectCompany(msg.sender);
    // 鉴权：只有普通银行才可注册公司。
    require(r2 == 2, "only banks are allowed to register companies");
    insertCompany(addr, uscc, 3, 0, 0);
    emit RegistrationEvent(3, addr, uscc);
}
```

转移凭证

信用凭证，即企业间签发的应收账款单据，由债务人向债权人签发，其流动方向与法币方向相反：A 向 B 申请贷款，则 A 签发信用凭证给 B；当债务人 A 向债权人 B 支付法币欠款，B 需向 A 返还相应的信用凭证。

在 `transferReceipt` 函数中，我们实现了**转移凭证**的功能，其对应了三种场景：

- debtor 银行向 debtee 公司提供信用凭证，例如 Bank 给 Car 提供了 1000 万的信用凭证。
- debtor 公司向 debtee 公司转移信用凭证，例如 Car 转移 500 万信用凭证给 Wheel。
- debtor 公司向 debtee 银行借款，例如 Car 转移 200 万信用凭证给 Bank，表示申请获得 200 万的融资贷款。

在该函数中，我们更新了交易双方的**借入凭据总金额**和**借出凭据总金额**，同时通过向 `t_in_receipt` 和 `t_out_receipt` 插入数据的方式，实现信用凭证的溯源。

```
// 转移信用凭证。
// msg.sender = debtor
// 使用场景：
// 1. debtor 银行向 debtee 公司提供信用凭证。
// 2. debtor 公司向 debtee 公司转移信用凭证。
// 3. debtor 公司向 debtee 银行借款（转移信用至银行，表示融资）。
function transferReceipt(address debtee, int256 amount, int deadline) public {
    // 获取债务人的票据总量。
    int debtorIn; int debtorOut;
    (debtorIn, debtorOut) = selectCompanyReceipts(msg.sender);
```

```

// 获取债权人的票据总量。
int debteeIn; int debteeOut;
(debteeIn, debteeOut) = selectCompanyReceipts(debtee);
// 判断转移金额是否合法。
require(amount > 0, "amount must be greater than zero");
require(debtorIn - debtorOut >= amount, "debtor does not have enough
balance");
// 更新双方的票据总量。
debtorOut += amount;
debteeIn += amount;
updateCompanyReceipts(msg.sender, debtorIn, debtorOut);
updateCompanyReceipts(debtee, debteeIn, debteeOut);
// 插入新的票据信息。
nextReceiptId++;
insertReceipt("t_in_receipt", toString(debtee), msg.sender, debtee,
nextReceiptId, amount, deadline);
insertReceipt("t_out_receipt", toString(msg.sender), msg.sender, debtee,
nextReceiptId, amount, deadline);

emit TransactionEvent(msg.sender, debtee, amount);
}

```

归还凭证

在 `returnReceipt` 函数中，我们实现了归还凭证的功能，其对应了三种场景：

- debtee 公司向 debtor 公司返还信用凭证，例如当 Car 向 Wheel 支付 500 万的货款时，Wheel 需要向 Car 返还 500 万信用凭证。
- debtee 银行向 debtor 公司返还信用凭证，例如当 Wheel 偿还 Bank 的 200 万贷款时，Bank 需要向 Wheel 返还 200 万信用凭证。
- debtee 公司向 debtor 银行返还信用凭证，例如当 Car 完成所有的交易时，需要向 Bank 偿还 1000 万，同时 Bank 需要返回 1000 万信用凭证给 Car。

```

// 返还信用凭证。
// msg.sender = debtee
// 使用场景：
// 1. debtee 公司向 debtor 公司返还信用凭证（表示 debtor 公司向 debtee 公司支付货款）。
// 2. debtee 公司向 debtor 银行返还信用凭证（表示期限前撤销信用凭证）。
// 3. debtee 银行向 debtor 公司返还信用凭证（表示公司完成融资的还款）。
function returnReceipt(int receiptId, int amount) public {
    address r1; address r2; int r3; int r4; int r5;
    (r1, r2, r3, r4, r5) = selectReceipt("t_in_receipt", toString(msg.sender),
receiptId);
// 获取债务人的票据总量。
int debtorIn; int debtorOut;
(debtorIn, debtorOut) = selectCompanyReceipts(r2);
// 获取债权人的票据总量。
int debteeIn; int debteeOut;
(debteeIn, debteeOut) = selectCompanyReceipts(r1);

```

```

// 判断返还金额是否合法。
require(amount > 0, "amount must be greater than zero");
require(r4 >= amount, "returning credit more than amount transfered at first
is not allowed");
// 更新双方的票据记录。
updateReceipts("t_in_receipt", toString(r1), receiptId, r4 - amount);
updateReceipts("t_out_receipt", toString(r2), receiptId, r4 - amount);
// 更新双方的票据总量。
debtorOut -= amount;
debteeIn -= amount;
updateCompanyReceipts(r2, debtorIn, debtorOut);
updateCompanyReceipts(r1, debteeIn, debteeOut);

emit ReturnEvent(r2, r1, amount);
}

```

服务端

项目地址: github.com/jiahonzheng/Blockchain/tree/master/Supplier-Financing/Server

在服务端实现中, 我使用了 [NestJS](#) 框架构建 API 服务, 这是一款使用 TypeScript 开发的 Node 应用框架。我选择此框架的原因是由于技术合伙的关系, 我曾使用此框架开发线上服务的服务端项目, 在开发过程中, 也对框架进行了很对功能封装和扩充: 日志中间件、数据库中间件、鉴权中间件、数据检验中间件等。

数据表定义

由于平台需要支持用户使用**统一社会信用代码** (USCC) 和**密码**进行登录, 以及我们需要维护 USCC 和区块链公钥地址间的关系, 因此我们需要使用数据库来存储账户的账户信息。在实现中, 我使用了 MySQL 数据库存储账户信息, 使用 Redis 维护登录状态信息 (Session)。以下是 MySQL 表的定义。

```

CREATE TABLE `user` (
  `uid` int(11) NOT NULL AUTO_INCREMENT COMMENT '用户 UID',
  `uscc` varchar(18) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '统一社会信用代码',
  `address` varchar(42) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '公钥地址',
  `password` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '用户密码',
  `type` int(11) NOT NULL DEFAULT '0' COMMENT '公司类型',
  PRIMARY KEY (`uid`),
  UNIQUE KEY `uscc` (`uscc`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;

```

对接区块链

我们使用 FISCO BCOS 官方提供的 [nodejs-sdk](#) 实现服务端与区块链的交互，需要解决的核心问题是如何实现智能合约中的 `msg.sender` 的修改。在参照 SDK 源码后，我在 `packages/api/web3j/web3jService.js` 文件中扩展了以下[方法](#)，从而实现了使用自定义传入的公钥和私钥发起交易的功能。

函数 `sendRawTransactionUsingCustomCredentials` 基本是仿照[函数](#) `sendRawTransaction` 实现的，主要更改是取消 `sendRawTransaction` 函数中使用 `this.config.account` 和 `this.config.privateKey` 构建交易数据的逻辑，使之支持自定义公钥和私钥的传入。

```
/**
 * 用自定义的公钥和私钥发送交易。
 */
async sendRawTransactionUsingCustomCredentials(account, privateKey, ...args) {
  let node = utils.selectNode(this.config.nodes);
  if (args.length !== 3) {
    let requestData = {
      jsonrpc: '2.0',
      method: 'sendRawTransaction',
      params: [this.config.groupID, args[0]],
      id: 1,
    };
    return channelPromise(
      node,
      this.config.authentication,
      requestData,
      this.config.timeout,
    );
  } else {
    let to = args[0];
    let func = args[1];
    let params = args[2];
    let blockNumberResult = await this.getBlockNumber();
    let blockNumber = parseInt(blockNumberResult.result, '16');
    let signTx = await this.rawTransactionUsingCustomCredentials(
      account,
      privateKey,
      to,
      func,
      params,
      blockNumber + 500,
    );
    return this.sendRawTransactionUsingCustomCredentials(
      account,
      privateKey,
      signTx,
    );
  }
}
```

```

}

/**
 * 使用自定义公钥和私钥组装交易数据。
 */
async rawTransactionUsingCustomCredentials(
  account,
  privateKey,
  to,
  func,
  params,
  blockLimit,
) {
  if (!isArray(params)) {
    params = params ? [params] : [];
  }
  let signTx = web3Sync.getSignTx(
    this.config.groupID,
    account,
    privateKey,
    to,
    func,
    params,
    blockLimit,
  );
  return signTx;
}

```

PEM 解析

为实现对 PEM 私钥文本进行解析，我实现了 `getPrivateKey` 函数，具体函数代码如下。

```

/**
 * 根据 PEM 文本内容计算私钥。
 * @param raw PEM 文件文本内容
 */
export function getPrivateKey(raw: string): string {
  const encodedPem = Buffer.from(raw);
  const decodedPem = (PEM.decode(encodedPem) as Buffer).toString('hex');
  return decodePem(decodedPem);
}

// 前缀: -----BEGIN EC PRIVATE KEY-----。
const EC_PRIVATE_KEY_PREFIX = '30740201010420';
// 前缀: -----BEGIN PRIVATE KEY-----
const PRIVATE_KEY_PREFIX =
  '308184020100301006072a8648ce3d020106052b8104000a046d306b0201010420';

/**

```



```

* 从 PEM 证书中解析出私钥字符串内容。
* @param pem 证书
*/
function decodePem(pem: string): string {
  let privateKey = '';
  if (pem.startsWith(EC_PRIVATE_KEY_PREFIX)) {
    // -----BEGIN EC PRIVATE KEY-----
    privateKey = pem.substring(
      EC_PRIVATE_KEY_PREFIX.length,
      EC_PRIVATE_KEY_PREFIX.length + 64,
    );
  } else if (pem.startsWith(PRIVATE_KEY_PREFIX)) {
    // -----BEGIN PRIVATE KEY-----
    privateKey = pem.substring(
      PRIVATE_KEY_PREFIX.length,
      PRIVATE_KEY_PREFIX.length + 64,
    );
  } else {
    throw new Error('expected `EC PRIVATE KEY` or `PRIVATE KEY`');
  }
  return privateKey;
}

```

客户端

项目地址: github.com/jiahonzheng/Blockchain/tree/master/Supplier-Financing/Web

在客户端实现中, 我们使用 [UmijS](#) 框架构建 React 网页应用, 这同样也是一款生产可用的 React 开发框架。与选择 NestJS 框架的理由一样, 选择 UmijS 是因为在技术合伙的过程中, 曾使用该框架构建线上应用, 对框架进行了很多功能扩充。

页面鉴权

在 UmijS 的基础上, 我实现了页面路由的鉴权功能, 通过在路由表设置 `authority` 属性, 即可设置对应页面的权限状态。

```

// 银行和公司注册页面
{
  path: '/registration',
  name: 'registration',
  icon: 'idcard',
  // 公司无法访问注册页面。
  authority: ['admin_bank', 'bank'],
  routes: [
    {
      path: '/registration/bank',
      name: 'bank',
      icon: 'bank',
      // 只有中央银行账户才可注册银行。

```

```

    authority: ['admin_bank'],
    component: './Registration/Bank',
  },
  {
    path: '/registration/company',
    name: 'company',
    icon: 'team',
    // 只有普通银行账户才可注册公司。
    authority: ['bank'],
    component: './Registration/Company',
  },
],
}

```

例如，在上述路由注册代码中，我们设置了以下的页面权限：

- 只有中央银行和普通银行才可访问 `/registration` 路由。
- 只有中央银行才可访问 `/registration/bank` 银行注册页面。
- 只有普通银行才可访问 `/registration/company` 公司注册页面。

i18n

在实现中，我还实现了文本显示的国际化配置，在需要显示文本的地方，我们调用 `formatMessage` 函数，传入文本对应的字段名称，通过运行时解析不同的国际化配置文件，即可实现文本的国际化显示。

例如在以下统一社会信用代码的表单项代码中，我们使用 `formatMessage({ id: 'account.uscc' })` 代替原本的显示内容。

```

<Form.Item label={formatMessage({ id: 'account.uscc' })}>
  {getFieldDecorator('uscc', {
    initialValue: currentUser.uscc,
  })(<Input disabled={true} style={disabledStyle} />)}
</Form.Item>

```

在运行时，通过根据配置（开发者配置、浏览器配置），我们读取不同的国际化配置，即可显示对应语言的文本显示。

```

export default {
  'account.uid': '用户ID',
  'account.uscc': '统一社会信用代码',
  'account.name': '公司名称',
  'account.address': '公钥地址',
  'account.type': '账户类型',
  'account.password': '用户密码',
  'account.type.admin_bank': '中央银行',
  'account.type.bank': '银行',
  'account.type.company': '公司',
  'account.inReceipts': '借入凭证总金额（元）',
  'account.outReceipts': '借出凭证总金额（元）',
}

```

```
};
```

功能测试

在完成上述的功能设计后，我们需要对各部分进行测试。

账户生成

在 FISCO BCOS 中，我们使用**账户**来标识和区分每一个独立的用户。区块链系统使用**公私钥密码体系**，每个一个账户对应着一对**公钥**和**私钥**。根据[官方文档](#)，我们可使用 `get_account.sh` 文件创建账户。

```
# 下载脚本。
curl -LO https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/account.html
# 设置权限。
chmod u+x get_account.sh
# 显示帮助信息。
bash get_account.sh -h
# 创建账户
bash get_account.sh
```

我们使用该脚本创建了**中央银行**、**银行**、**汽车厂**、**轮胎厂**和**轮毂厂**对应的账号，账号相关信息如下。

- 中央银行 AdminBank
 - 统一社会信用代码

```
00000000000000000001
```

- 公钥

```
0xa02e687623e4252f98b33b05349d07b04d8488a5
```

- 私钥

```
-----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGByqGSM49AgEGBSuBBAKBG0wawIBAQQggnvzCBz8rK3zVLmdDerJw
Kq+RKzdW+oW2htIVAtKSGKOhRANCAAQR0pn5nsY/uYcKXEKRmunPhSzd5OT0uxb6
w+tP6aZkOtmk0IKspq2/Gq9uh10OGnpbNWSsIbKl/0brC/m4XhL
-----END PRIVATE KEY-----
```

- 普通银行 Bank
 - 统一社会信用代码

```
00000000000000000002
```

- 公钥

```
0x8674c4870eb4e4f8b968af53ff5eee5d626de531
```

- 私钥

```
-----BEGIN PRIVATE KEY-----  
MIGEAgEAMBAGByqGSM49AgEGBSuBBAKKBG0wawIBAQQgEMLEu788mVwaPtj7NJmF  
vQ/Wfq+PhVhTYvldYBuoQFihRANCAARldsiEnNLglKKfgjdit5FmKQr7ZVPDc2T+  
Fzr26/25p5K+an02OMCMOuvslGBq2hrpUeuUYJAGlZDJEm7LeU5S  
-----END PRIVATE KEY-----
```

- 汽车厂 Car

- 统一社会信用代码

```
00000000000000000003
```

- 公钥

```
0xd3584213a3cc2ebbe97dac2c2dc9c49448282d9d
```

- 私钥

```
-----BEGIN PRIVATE KEY-----  
MIGEAgEAMBAGByqGSM49AgEGBSuBBAKKBG0wawIBAQQguM0pCwU5ZYUTVzb9C4Af  
vw4NNTqk3AvoOV20fRUhRlChRANCAATsZtbXXG3M+Fd/Su8hnIPvAh4c6Sw+rbQL  
iCz0jBGYCKUWQF8RIsPBH/+6IsB9z8taOt7qyjtgN1XWlt0v6dkY  
-----END PRIVATE KEY-----
```

- 轮胎厂 Wheel

- 统一社会信用代码

```
00000000000000000004
```

- 公钥

```
0xf115ac35da5b8bf03853fef20565eff978d85df2
```

- 私钥

```
-----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGByqGSM49AgEGBSuBBAKBG0wawIBAQQgkSu0G1LOoyU2YPhp2/OM
zPtnULnX/bHSv1BRSMCKSjmhRANCAASCiei6x2WrtYDhfk+huexiKV/atqEFv3a1
/1TIXLZqu2LQeFVQsLYHYnMxENyMxCQ6tKCyv7voyBT4XzMUSQzj
-----END PRIVATE KEY-----
```

- 轮毂厂 Hub

- 统一社会信用代码

```
00000000000000000005
```

- 公钥

```
0x40524ebc4188993b7ab834db756a5d14172f7dcd
```

- 私钥

```
-----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGByqGSM49AgEGBSuBBAKBG0wawIBAQQgdkH8lXXEtQbwU32ET+nS
w4qYfVnl+kLJoHyuhLEOjNihRANCAATtsVZhW/obVGEDFO0xaUKk2zaJoN+wGm2p
3BptRlkeowreIwxkeRj+1YrcN4i0KM/Ht0+EATx5oTuc4lmHpw2m
-----END PRIVATE KEY-----
```

在 FISCO BCOS 平台中，由于当群组内发送交易的账号如果不在私钥管理里添加时会被视为异常用户，我们需要添加上述账户至 **私钥管理** 页面中。由于部署合约需要私钥账户，我们创建并添加了 Admin 账户。

新增用户

群组内发送交易的账号都需要在私钥管理里添加，否则会被判断为异常用户。

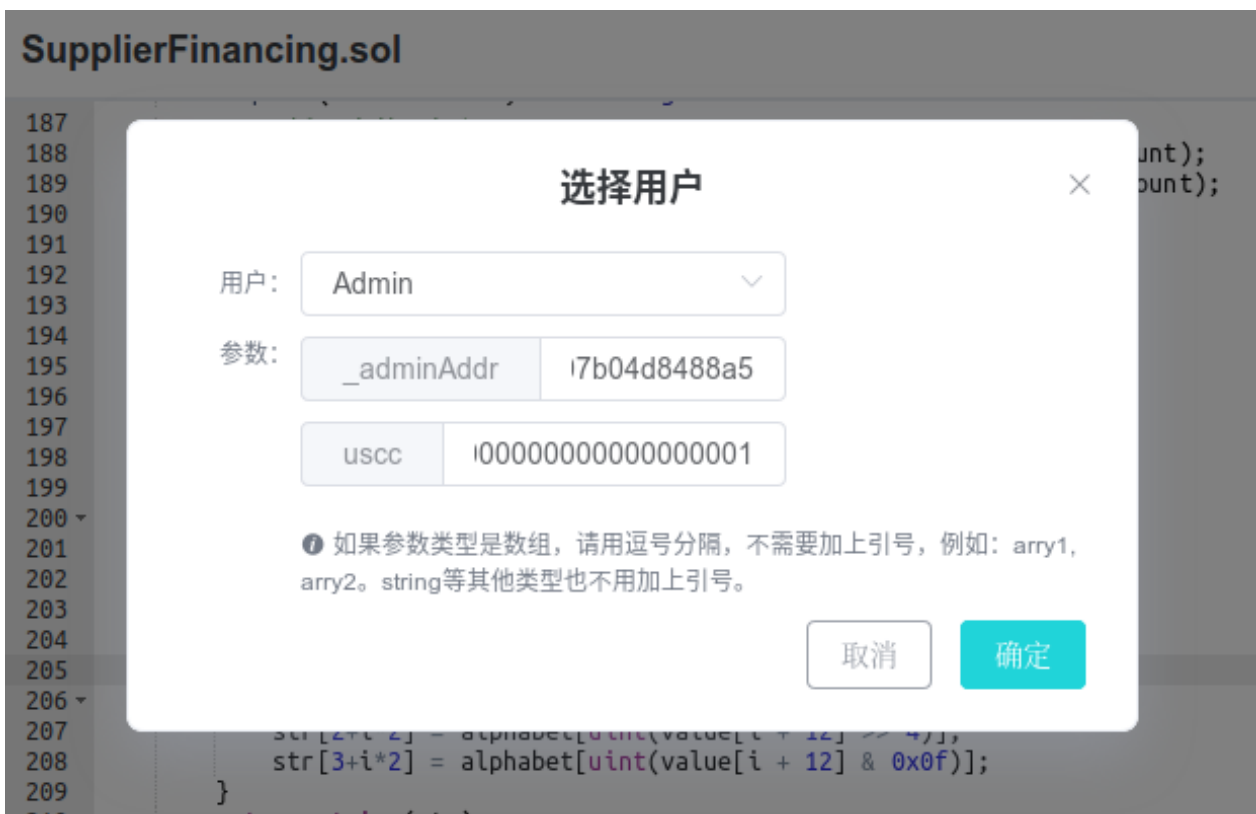
请输入用户名或公钥地址

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
Hub	700007		0x40524ebc4188993b7ab...	正常	修改
Wheel	700006		0xf115ac35da5b8bf03853f...	正常	修改
Car	700005		0xd3584213a3cc2ebbe97d...	正常	修改
Bank	700004		0x8674c4870eb4e4f8b968...	正常	修改
AdminBank	700003		0xa02e687623e4252f98b3...	正常	修改
Admin	700002		0xab336d2a019eeb49ca2...	正常	修改

共 6 条10条/页1 前往1 页

合约部署

我们在 **合约 IDE** 页面中添加 [Table.sol](#) 和 [SupplierFinancing](#) 文件，然后使用 Admin 私钥账户进行部署，部署的参数为 **AdminBank** 的公钥地址和统一社会信用代码。



合约部署成功后，我们即可在 [查看合约](#) 页面获得合约地址、合约 ABI 等信息，具体页面如下。

合约名称	合约目录	合约地址	合约abi	合约bin	创建时间	操作
SupplierFinancing	/	0x661b434be321a5fac3d07d295e7...	[{"constant":false,"inputs":[{"name": "...	608060405260043610610062576...	2019-12-13 15:19:08	发送交易

启动服务

在部署智能合约后，我们需要启动[客户端](#)和[服务端](#)项目。

在启动服务端项目之前，我们需要修改 [utils.ts](#) 文件中的 `CONTRACT_ADDRESS` 和 `CONTRACT_ABI` 的变量定义。

```
// 智能合约部署地址。
export const CONTRACT_ADDRESS = '0x661b434be321a5fac3d07d295e7b042a40699849';
// 智能合约 ABI 定义。
export const CONTRACT_ABI = [
  // .....
];
```

在修改代码中的合约地址和 ABI 变量后，我们即可执行以下命令启动服务端。

```
# 启动服务端项目。
cd Server/
yarn start:dev
```

```

base E:\Users\Code\Blockchain\Supplier-Financing\Server > yarn start:dev
yarn run v1.15.2
$ cross-env NODE_ENV=development nodemon
[nodemon] 2.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: ts,hbs
[nodemon] starting `cross-env NODE_ENV=development ts-node src/main.ts`

```

我们通过执行 `yarn start:dev` 命令启动客户端项目。

```

# 启动客户端项目。
cd Web/
yarn start:dev

```

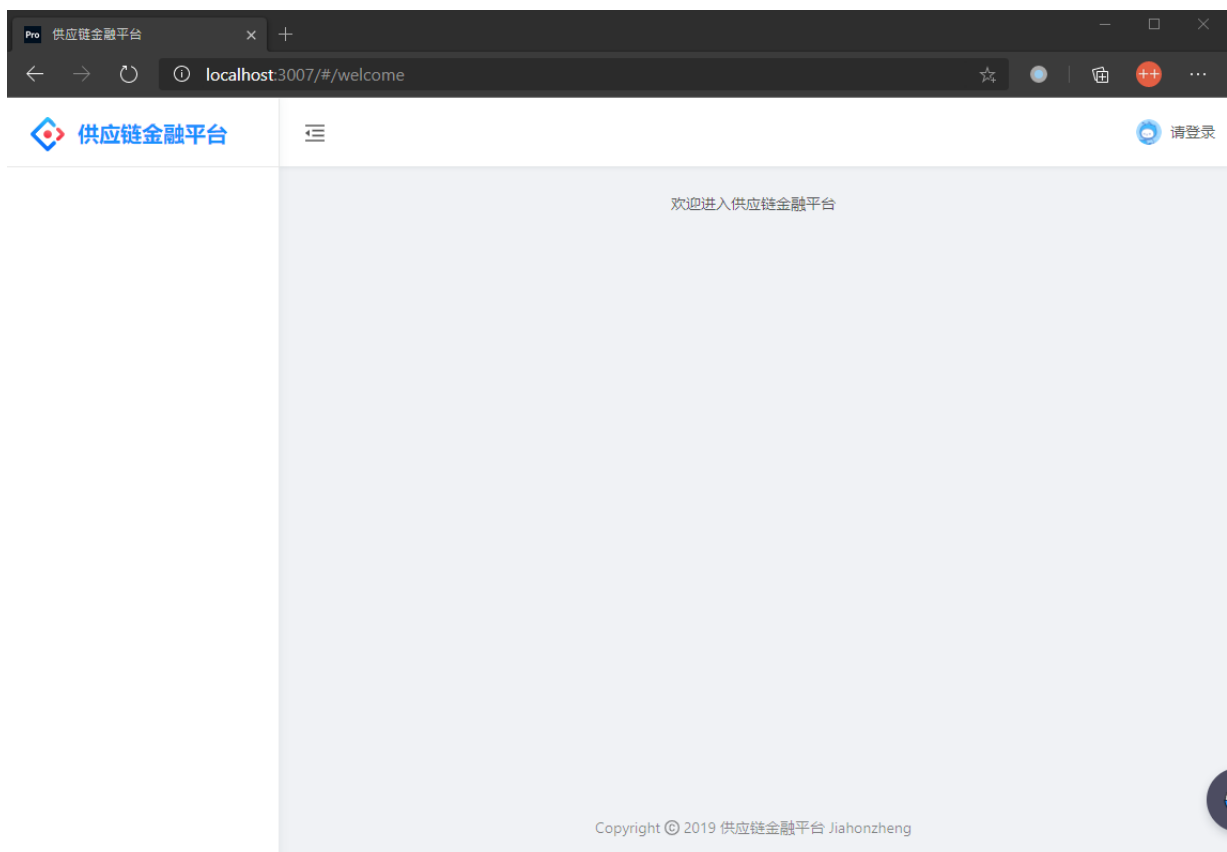
```

base E:\Users\Code\Blockchain\Supplier-Financing\Web > yarn start:dev
yarn run v1.15.2
$ cross-env UMI_ENV=dev umi dev --port=3007
  □ □ Starting Umi UI using umi@2.12.5..
  □ □ Ready on http://localhost:3007/
    build [==          ] 10%Starting the development server...
Build completed in 13.678s
DONE Compiled successfully in 13685ms

App running at:
- Local:   http://localhost:3007/ (copied to clipboard)
- Network: http://192.168.1.106:3007/

```

在服务端和客户端都启动后，我们可访问 <http://localhost:8000> 访问平台页面，界面如下图所示。



银行注册

在实现中，我们约定只有中央银行账户才可以注册银行账户，因此我们需要以中央银行的身份登录平台。

* 统一社会信用代码:

000000000000000001

* 用户密码:

.....

登录

在 账户信息 页面，我们即可查看当前用户（中央银行）的账户信息，如借入凭证总金额、借出凭证总金额、统一社会信用代码、账户类型和公钥地址。

供应链金融平台

中央银行

注册

信用凭证

账户

账户信息

账户公示

借入凭证总金额 (元)

¥100,000,000,000,000,000.00

借出凭证总金额 (元)

¥0.00

统一社会信用代码:

000000000000000001

账户类型:

中央银行

公钥地址:

0xa02e687623e4252f98b33b05349d07b04d8488a5

Copyright © 2019 供应链金融平台 Jiahonzheng

在 账户公示 页面，我们可查看当前平台上所有的账户信息（统一社会信用代码、账户类型、公钥地址）。根据下图可知，当前平台上只有中央银行一个账户。

统一社会信用代码	账户类型	公钥地址
000000000000000001	中央银行	0xa02e687623e4252f98b33b05349d07b04d8488a5

现在，我们在 银行注册 页面完成银行账户 Bank 的注册工作：我们输入 Bank 的 USCC、公钥地址和账户密码至表单中，同时添加 AdminBank 账户的私钥内容，最终完成 Bank 账户的注册。

* 统一社会信用代码:

000000000000000002

* 公钥地址:

0x8674c4870eb4e4f8b968af53ff5eee5d626de531

注册类型:

银行

* 账户密码:

123456

签发者公钥地址:

0xa02e687623e4252f98b33b05349d07b04d8488a5

* 签发者私钥内容:

-----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGByqGSM49AgEGBSuBBAAKBG0wawI
BAQQqnvzCBz8rK3zVLmdDerJw
Kg+RKzdW+oW2htIVAtKSGKOhRANCAAQR0pn5nsY/
uYcKXEKRmunPhSzd5OT0uxb6
w+tP6aZkOtmk0IKspq2/Gq9uh10OGnpbNWSslbKI/
0brC/m4XhL
-----END PRIVATE KEY-----

注册

在完成注册后，我们即可在 账户公示 页面中查看 Bank 账户的公开信息。

统一社会信用代码	账户类型	公钥地址
000000000000000001	中央银行	0xa02e687623e4252f98b33b05349d07b04d8488a5
000000000000000002	银行	0x8674c4870eb4e4f8b968af53ff5eee5d626de531

公司注册

在实现中，我们约定只有银行账户才可以注册公司账户，因此我们需要以银行的身份登录平台。在 注册公司 页面里，我们填写 Car 账户的相关信息，并填写当前账户 Bank 的私钥内容。

* 统一社会信用代码: 000000000000000003

* 公钥地址: 0xd3584213a3cc2ebbe97dac2c2dc9c49448282d9d

注册类型: 公司

* 账户密码: 123456

签发者公钥地址: 0x8674c4870eb4e4f8b968af53ff5eee5d626de531

* 签发者私钥内容: -----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGByqGSM49AgEGBSuBBAAKBG0wawI
BAQQgEMLEu788mVwaPtj7NJmF
vQ/Wfq+PhVhTYvIDYBuoQFihRANCAARldsiEnNLg1K
Kfgjdit5FmKQr7ZVPDc2T+
Fzr26/25p5K+an02OMCMOuvslGBq2hrpUeuUYJAGIZ
DJE7LeU5S
-----END PRIVATE KEY-----

注册

在完成 Car 账户的注册后，我们即可在 账户公示 页面中，查看 Car 的公开信息。

统一社会信用代码	账户类型	公钥地址
0000000000000000001	中央银行	0xa02e687623e4252f98b33b05349d07b04d8488a5
0000000000000000002	银行	0x8674c4870eb4e4f8b968af53ff5eee5d626de531
0000000000000000003	公司	0xd3584213a3cc2ebbe97dac2c2dc9c49448282d9d

在 Bank 账户下，我们依次对 Wheel、Hub 账户进行注册，最终 账户公示 页面如下图所示。

统一社会信用代码	账户类型	公钥地址
000000000000000001	中央银行	0xa02e687623e4252f98b33b05349d07b04d8488a5
000000000000000002	银行	0x8674c4870eb4e4f8b968af53ff5eee5d626de531
000000000000000003	公司	0xd3584213a3cc2ebbe97dac2c2dc9c49448282d9d
000000000000000004	公司	0xf115ac35da5b8bf03853fef20565eff978d85df2
000000000000000005	公司	0x40524ebc4188993b7ab834db756a5d14172f7dcd

创建凭证

我们现在模拟以下的场景：Bank 向 Car 提供 1000w 的信用凭证，即 Bank 转移 1000w 的信用凭证给 Car。

首先，在 **账户信息** 页面中，我们可查看 Bank 当前的借入凭证和借出凭证的总金额，如下图所示。

借入凭证总金额（元）
¥100,000,000,000.00
借出凭证总金额（元）
¥0.00

以下是 Car 账户的借入凭证和借出凭证的总金额信息。

借入凭证总金额（元）
¥0.00
借出凭证总金额（元）
¥0.00

我们在登录 Bank 账户的情况下，在 **转移凭证** 页面完成 1000w 信用凭证的签发，表单填写内容如下。

* 接收者公钥地址:

* 截止日期:

* 转移金额 (分):

当前公钥地址:

* 当前私钥内容:

```
-----BEGIN PRIVATE KEY-----
MIGEAQEAMBAgByqGSM49AgEGBSuBBAAKBG0wawI
BAQQqEMLEu788mVwaPtj7NJmF
vQ/Wfq+PhVhTYvIDYBuoQFihRANCAARldsiEnNLq1K
Kfqjdit5FmKQr7ZVPDc2T+
Fzr26/25p5K+an02OMCMOuvslGBq2hrpUeuUYJAGIZ
DJE7LeU5S
-----END PRIVATE KEY-----
```

[转移信用凭证](#)

在完成信用凭证的签发后，Bank 账户的借入凭证和借出凭证总金额信息如下。由图可知，Bank 账户的借出凭证总金额增加了 1000 万。

借入凭证总金额 (元)
¥100,000,000,000.00
借出凭证总金额 (元)
¥10,000,000.00

与之对应的是，Car 账户的借入凭证总金额增加了 1000 万，在 借入凭据 栏目中，我们发现增加了一条接收来自 Bank 的凭证记录。

借入凭证总金额 (元)

¥10,000,000.00

借出凭证总金额 (元)

¥0.00

借入凭证

借出凭证

凭证标识	凭证签发者	凭证金额 (元)	截止日期
1	0x8674c4870eb4e4f8b968af53ff5eee5d626de531	¥10,000,000.00	2020-12-13

< 1 >

转移凭证

现在, 我们模拟 Car 转移 500 万信用凭证给 Wheel 的场景。在登录 Car 账户的情况下, 我们在 **转移凭证** 页面中填写以下的表单内容。

* 接收者公钥地址:

0xf115ac35da5b8bf03853fef20565eff978d85df2

* 截止日期:

2019/12/31

* 转移金额 (分):

500000000

当前公钥地址:

0xd3584213a3cc2ebbe97dac2c2dc9c49448282d9d

* 当前私钥内容:

-----BEGIN PRIVATE KEY-----
MIGEAQEAMBAgByqGSM49AgEGBSuBBAKBG0wawI
BAQQguM0pCwU5ZYUTVzb9C4Af
vw4NNTqk3AvoOV20fRUhRICHANCAATsZtbXXG3M+
Fd/Su8hnlPvAh4c6Sw+rbQL
iCz0jBGYCKUWQF8RIsPBH/+6lsB9z8taOt7qyjGN1XWI
t0v6dkY
-----END PRIVATE KEY-----

转移信用凭证

在完成信用凭证的转移后，Car 账户的借出凭证总金额增加了 500 万，同时 借出凭据 栏目增加了一条转移凭证至 Wheel 账户的记录。

借入凭证总金额（元）

¥10,000,000.00

借出凭证总金额（元）

¥5,000,000.00

借入凭据

借出凭据

凭据标识	凭据接收者	凭据金额（元）	截止日期
2	0xf115ac35da5b8bf03853fef20565eff978d85df2	¥5,000,000.00	2019-12-31

<

1

>

与之对应，Wheel 账户的借入凭证总金额增加了 500w ，相关信息如下图所示。

供应链金融平台

中央银行

注册

信用凭证

凭证明细

转移凭证

归还凭证

账户

* 接收者公钥地址:

请输入凭证接收者的 42 位公钥地址

* 截止日期:

请输入凭证的截止日期

* 转移金额（分）:

请输入转移的凭证金额（分）

当前公钥地址:

0xa02e687623e4252f98b33b05349d07b04d8488a5

* 当前私钥内容:

请输入当前账户的私钥内容

转移信用凭证

申请融资

我们现在模拟这样的场景：Wheel 账户根据已拥有的 500 万的信任凭证，向 Bank 账户申请 200 万的融资贷款。在登录 Wheel 账户的情况下，我们在 转移凭证 页面完成如下的表单填写。

* 接收者公钥地址:

* 截止日期:

* 转移金额 (分):

当前公钥地址:

* 当前私钥内容:

```
-----BEGIN PRIVATE KEY-----
MIGEAqEAMBAGByqGSM4AgEGBSuBBAAKBG0waw
IBAQQqkSu0G1LOoyU2YPhp2/OM
zPtnULnX/bHSv1BRSMckSjmhRANCAASCiei6x2WrtY
Dhfk+huexiKV/atqEFv3a1
/1TIXLZqu2LQeFVQsLYHYnMxENyMxCQ6tKCyy7voyB
T4XzMUSQzj
-----END PRIVATE KEY-----
```

[转移信用凭证](#)

在完成上述操作后，Wheel 的借出凭证总金额增加了 200 万，同时 借出凭据 栏目增加了一条转移凭证至 Bank 的记录。

借入凭证总金额 (元)

¥5,000,000.00

借出凭证总金额 (元)

¥2,000,000.00

借入凭据

借出凭据

凭据标识	凭据接收者	凭据金额 (元)	截止日期
3	0x8674c4870eb4e4f8b968af53ff5eee5d626de531	¥2,000,000.00	2020-03-13

<

1

>

与之对应，Bank 的借入凭证总金额增加了 200 万，同时 借入凭证 栏目增加了一条接收来自 Wheel 的凭证记录。

借入凭证总金额 (元)

¥100,002,000,000.00

借出凭证总金额 (元)

¥10,000,000.00

借入凭证

借出凭证

凭证标识	凭证签发者	凭证金额 (元)	截止日期
3	0xf115ac35da5b8bf03853fef20565eff978d85df2	¥2,000,000.00	2020-03-13

< 1 >

偿还贷款

现在，我们模拟以下场景：Bank 对 Wheel 收回 200 万的融资贷款。在登录 Bank 账户的情况下，我们完成以下的表单内容填写。

* 归还凭证的标识：

3

* 归还金额 (分)：

200000000

当前公钥地址：

0x8674c4870eb4e4f8b968af53ff5eee5d626de531

* 当前私钥内容：

```
-----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGByqGSM49AgEGBSuBBAKBG0wawI
BAQQgEMLEu788mVwaPtj7NJmF
vQ/Wfq+PhVhTYvIDYBuoQFihRANCAARIdsiEnNLg1K
Kfgjdit5FmKQr7ZVPDc2T+
Fzr26/25p5K+an02OMCMOuvslGBq2hrpUeuUYJAGIZ
DJEm7LeU5S
-----END PRIVATE KEY-----
```

归还信用凭证

在完成上述操作后，Wheel 的借出凭证总金额减少了 200 万。

借入凭证总金额 (元)

¥5,000,000.00

借出凭证总金额 (元)

¥0.00

同时，Bank 的借入凭证总金额也减少了 200 万。

借入凭证总金额 (元)

¥100,000,000,000.00

借出凭证总金额 (元)

¥10,000,000.00

偿还贷款

经过上述各部分的模拟，现在 Car 还有 500 万的贷款未偿还给 Wheel，现在我们 Car 偿还贷款给 Wheel 的场景进行模拟，即 Wheel 归还凭证给 Car。在登录 Wheel 账户的情况下，我们在 归还凭证 页面中完成以下的表单内容设置。

* 归还凭证的标识:

2

* 归还金额 (分):

500000000

当前公钥地址:

0xf115ac35da5b8bf03853fef20565eff978d85df2

* 当前私钥内容:

```
-----BEGIN PRIVATE KEY-----
MIGEAgEAMBAGByqGSM49AgEGBSuBBAKBG0wawI
BAQQgkSu0G1LOoyU2YPhp2/OM
zPtnULnX/bHSv1BRSMCKSjmhRANCAASCiei6x2WrtYD
hfk+huexiKV/atqEFv3a1
/1TIXLZqu2LQeFVQsLYHYnMxENyMxCQ6tKCyv7voyBT
4XzMUSQzj
-----END PRIVATE KEY-----
```

归还信用凭证

在完成上述操作后，Wheel 的借入凭证总金额减少了 500 万。

借入凭证总金额 (元)

¥0.00

借出凭证总金额 (元)

¥0.00

同时，Car 的借出凭证总金额减少了 500 万。

借入凭证总金额 (元)

¥10,000,000.00

借出凭证总金额 (元)

¥0.00

借入凭证

借出凭证

凭证标识	凭证签发者	凭证金额 (元)	截止日期
1	0x8674c4870eb4e4f8b968af53ff5eee5d626de531	¥10,000,000.00	2020-12-13

< 1 >

此时，Car 的借入凭证总金额为 1000 万。

销毁凭证

最后，Car 需要销毁之前申请的 1000 万信用凭证。在登录 Bank 账户的情况下，我们完成以下的表单内容填写。

* 归还凭证的标识:

1

* 归还金额 (分):

1000000000

当前公钥地址:

0xd3584213a3cc2ebbe97dac2c2dc9c49448282d9d

* 当前私钥内容:

-----BEGIN PRIVATE KEY-----

MIGEAgEAMBAGByqGSM49AgEGBSuBBAKBG0wawI

BAQQguM0pCwU5ZYUTVzb9C4Af

vw4NNTqk3AvoOV20fRUhRICHANCAATsZtbXXG3M+

Fd/Su8hnlPvAh4c6Sw+rbQL

iCz0iBGYCKUWQF8RIsPBH/+6lsB9z8taOt7qyjtGN1XWI

t0v6dkY

-----END PRIVATE KEY-----

归还信用凭证

在完成上述操作后，Car 的借入凭证总金额和借出凭证总金额都为 0。

借入凭证总金额 (元)

¥0.00

借出凭证总金额 (元)

¥0.00

Bank 的凭证总金额也恢复至原有状态。

借入凭证总金额 (元)

¥100,000,000,000.00

借出凭证总金额 (元)

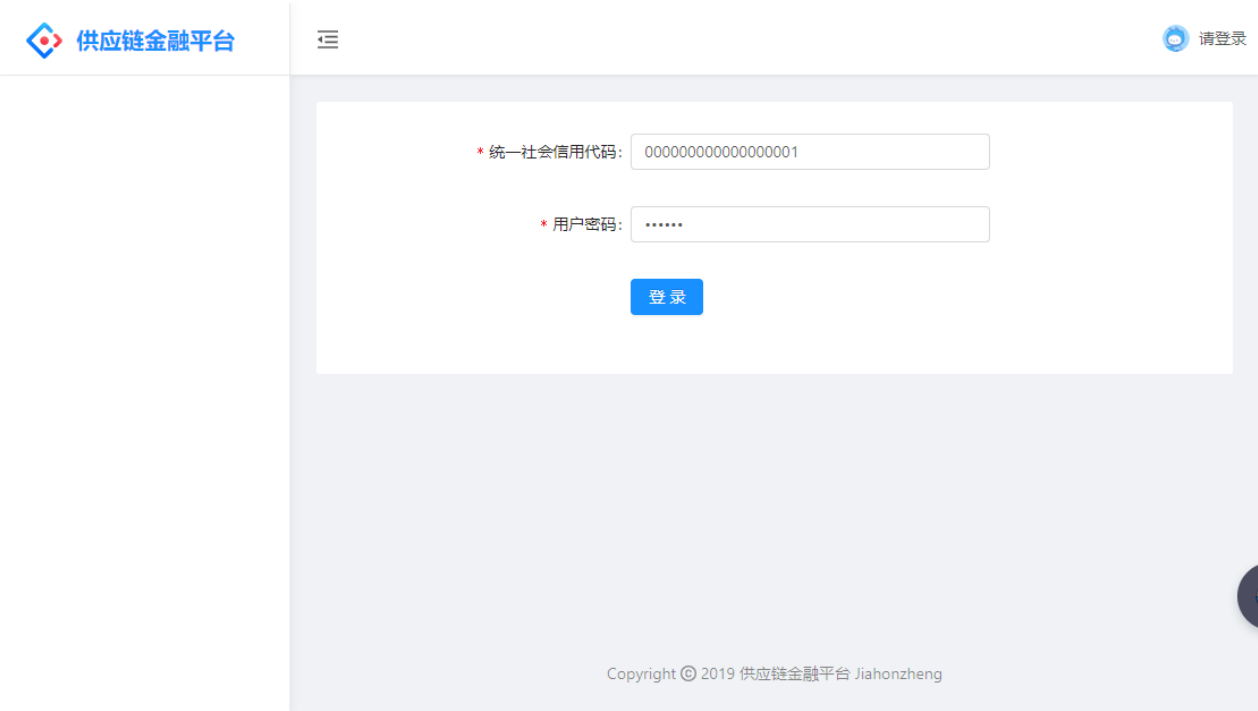
¥0.00

至此，我们成功完成了平台的各部分功能测试。

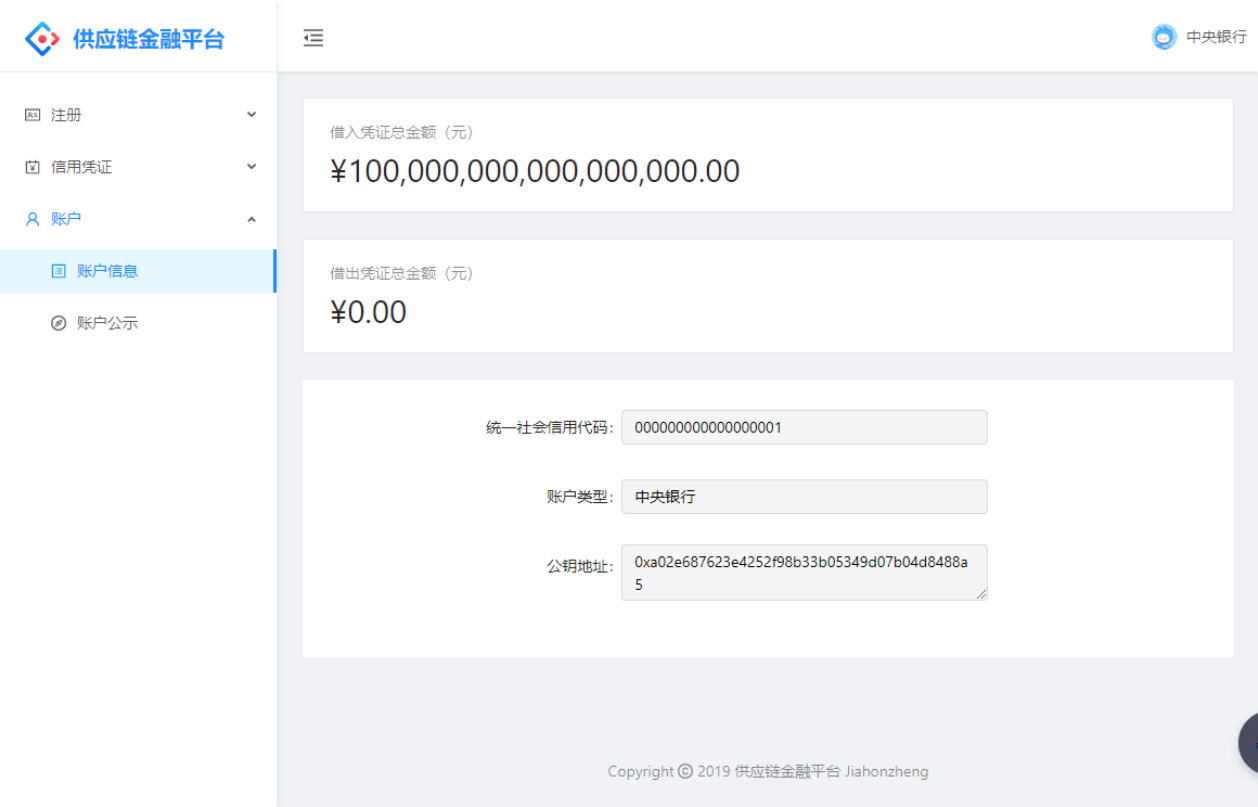
界面展示

视频演示：[demo.mp4](#)

账户登录页面



账户信息页面



账户公示页面

供应链金融平台

中央银行

注册

信用凭证

账户

账户信息

账户公示

1

统一社会信用代码	账户类型	公钥地址
00000000000000000001	中央银行	0xa02e687623e4252f98b33b05349d07b04d8488a5
00000000000000000002	银行	0x8674c4870eb4e4f8b968af53ff5eee5d626de531
00000000000000000003	公司	0xd3584213a3cc2ebbe97dac2c2dc9c49448282d9d
00000000000000000004	公司	0xf115ac35da5b8bf03853fef20565eff978d85df2
00000000000000000005	公司	0x40524ebc4188993b7ab834db756a5d14172f7dcd

1

银行注册页面

供应链金融平台

中央银行

注册

银行注册

信用凭证

账户

* 统一社会信用代码:

请输入注册企业的 18 位统一社会信用代码

* 公钥地址:

请输入注册企业的 42 位公钥地址

注册类型:

银行

* 账户密码:

请输入注册企业的账户密码

签发者公钥地址:

0xa02e687623e4252f98b33b05349d07b04d8488a5

* 签发者私钥内容:

请输入签发者私钥内容

注册

公司注册页面

供应链金融平台

注册

公司注册

信用凭证

账户

银行

* 统一社会信用代码:

请输入注册企业的 18 位统一社会信用代码

* 公钥地址:

请输入注册企业的 42 位公钥地址

注册类型:

公司

* 账户密码:

请输入注册企业的账户密码

签发者公钥地址:

0x8674c4870eb4e4f8b968af53ff5eee5d626de531

* 签发者私钥内容:

请输入签发者私钥内容

注册

凭证明细页面

供应链金融平台

注册

信用凭证

凭证明细

转移凭证

归还凭证

账户

中央银行

借入凭证总金额 (元)

¥100,000,000,000,000,000.00

借出凭证总金额 (元)

¥0.00

借入凭证

借出凭证

凭证标识	凭证签发者	凭证金额 (元)	截止日期
暂无数据			

转移凭证页面

供应链金融平台

注册

信用凭证

凭证明细

转移凭证

归还凭证

账户

中央银行

* 接收者公钥地址:

请输入凭证接收者的 42 位公钥地址

* 截止日期:

请输入凭证的截止日期

* 转移金额 (分):

请输入转移的凭证金额 (分)

当前公钥地址:

0xa02e687623e4252f98b33b05349d07b04d8488a5

* 当前私钥内容:

请输入当前账户的私钥内容

转移信用凭证

归还凭证页面

供应链金融平台

注册

信用凭证

凭证明细

转移凭证

归还凭证

账户

中央银行

* 归还凭证的标识:

请输入归还凭证的标识

* 归还金额 (分):

请输入归还的凭证金额 (分)

当前公钥地址:

0xa02e687623e4252f98b33b05349d07b04d8488a5

* 当前私钥内容:

请输入当前账户的私钥内容

归还信用凭证

Copyright © 2019 供应链金融平台 Jiahonzheng

实验心得

在本次实验中，我实现了基于区块链系统 FISCO-BCOS 的供应链金融项目，涵盖链端、服务端、客户端的开发，实验内容综合且全面。

在实验过程中，我对智能合约开发语言 Solidity 有了更为深入的理解，在实现 `selectCompany` 函数中，原本是想返回结构体，后查阅 Spec 后发现，函数返回结构体是不可行的，于是就想到了两种解决方法：

1. 在函数内部将输出结果修改至 Storage 类型变量，然后在其他函数访问这个变量，从而间接访问函数原本要输出的结构体。
2. 使用元组的方式，返回结构体的各个成员。

由于第 1 种方法会修改世界状态，因此我选择使用了第 2 种方法。在实现过程中，出现了不少的编译错误，最后通过在 `string` 类型变量添加 `memory` 关键字，解决了编译问题。

```
// 根据 addr 获取公司信息。
function selectCompany(address addr) private returns(string memory r1, int r2,
int r3, int r4) {
    Table t_company = openTable("t_company");
    Entries entries = t_company.select(toString(addr),
t_company.newCondition());
    require(entries.size() == 1, "company does not exist or is not unique");
    Entry entry = entries.get(0);
    r1 = entry.getString("uscc");
    r2 = entry.getInt("type");
    r3 = entry.getInt("in_receipts");
    r4 = entry.getInt("out_receipts");
}
```

在实现服务端与区块链的通讯功能的过程中，官方 SDK 的清晰代码实现，让我学习到了很多区块链的底层技术，如交易数据的组装方式、JSONRPC 的响应解析、ABI 的实际用途等。阅读源码的过程中，也学习到了很多工程上的编码技巧，收获满满。

经过了这次供应链金融区块链项目的开发，我对区块链在供应链金融领域的应用有了更为深切的理解：由于区块链的特性，供应链中的应收账款单据记录变得无法篡改，使得信用传递在技术上是可行的，这极大提高了供应链中下游企业的融资能力。当然，融资能力的提高，也需要政策、法律环境的支持。

总体来说，本次实验是成功的，由于实验过程比较顺利，所以没有耗费过多的时间，花了大致 3 天的时间。通过本次实验，我对区块链知识、应用场景有了更为全面深入的理解，感谢老师和 FISCO-BCOS 工程师提供本次宝贵的实验机会，让我收获颇丰。