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	17343119	姓名	吴明章
电话	13632617052	Email	1730174410@qq.com
开始日期	12.1	完成日期	12.13

一、 项目背景

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。

二、 方案设计

存储设计：

利用 Table 存储企业信息和账单数据，包括 t_company 表和 t_bil 表。前者存储企业信息，包括企业名称(company_name)、企业地址(company_address)、企业资产值(asset_value)、企业账户地址(account)、企业私钥(private_key)、企业公钥(public_key)六个字段。其中企业名称为主键，要求企业名称必须是唯一的

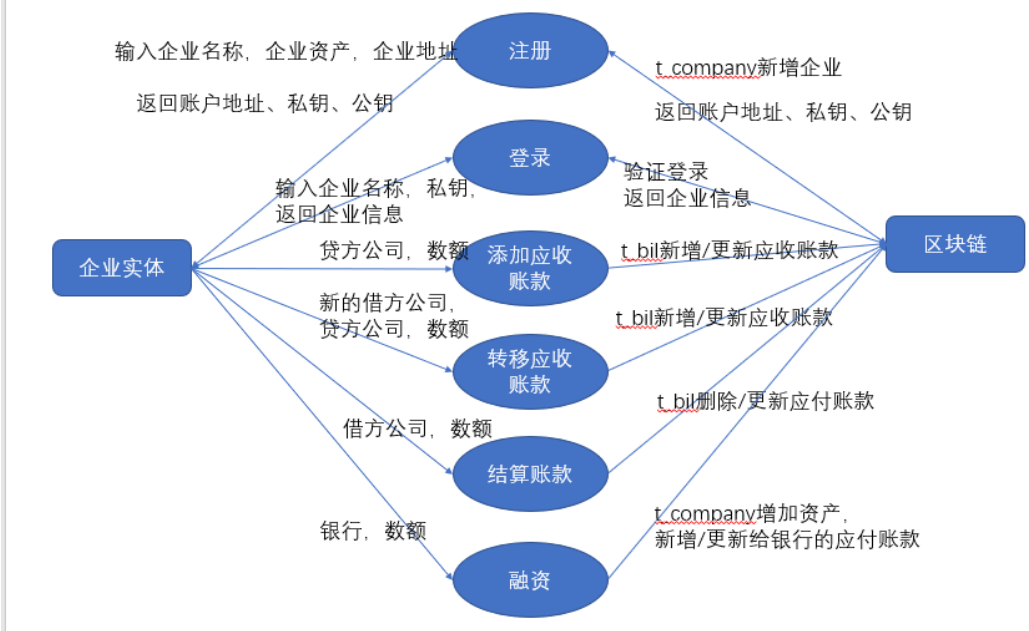
company_name	account	private_key	public_key	company_address	asset_value
BoC	0xbe3aee0c40854e18958e43e67bfec6d793e5e413	b61b8bf29e3aef4274bef9cbedd7ac6d86b27dd3b7b2d1c7c60155b01c911086	4466daf7d35a31947c9f5dc4d679a781aec2d4ee6f78e3a23ac7660519a06b15d8c9d5aaa794c6172dbd05b48a0bf736612bfb964a3b3eafb87f5de8afcce3bd	China	10000000

后者存储应收账款信息，包括借方、贷方、数额三个字段。示例如下，表示 B 企业欠 B 企业 1000 元。

creditor	debtor	Amount

A	B	1000

数据流图示例



核心功能介绍（文字+代码）

合约部署和初始化：采用 java spring-boot 框架，在 WebController.java 中调用 Connector 的方法，部署 Company.sol 合约并初始化相关信息。

// Connector 类实现了后端(WebController)--链端(Company.java)的连接（本质上属于链端），通过对象 c 进行合约的部署和初始化(c.init()和 c.deployAssetAndRecordAddr())、c.register()、c.login()、c.addBil()、c.transferBil()、c.repayBil()和 c.finance()等操作。

```
    private static Connector c;

    WebController() throws Exception{
        c = new Connector();
        c.init();
        c.deployAssetAndRecordAddr();
    }

    // 初始化 web3j 和 credentials (Connector.java)
    public void init() throws Exception {
        @SuppressWarnings("resource")
        ApplicationContext context = new
ClassPathXmlApplicationContext("classpath:applicationContext.xml");
        Service service = context.getBean(Service.class);
        service.run();

        ChannelEthereumService channelEthereumService = new ChannelEthereumService();
```

```
        channelEthereumService.setChannelService(service);
        web3j = Web3j.build(channelEthereumService, 1);

        credentials = GenCredential.create(Keys.createEcKeyPair());
    }

    // 部署 Company 合约并初始化合约地址(Connector.java)
    public void deployAssetAndRecordAddr() {
        try {
            Company company = Company.deploy(web3j, credentials, new
StaticGasProvider(gasPrice, gasLimit)).send();
            contractAddress = company.getContractAddress();
            System.out.println(" deploy Asset success, contract address is " + contractAddress);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            // e.printStackTrace();
            System.out.println(" deploy Asset contract failed, error message is  " +
e.getMessage());
        }
    }
```

注册功能：浏览器访问 localhost:8080/register 进入注册页面（后端返回 register.html 资源），可以填写企业名称、企业地址和企业资产值从而提交表单(前端)，传到 spring-boot 的 WebController 中(后端)，解析参数并传给链端执行 register 操作，注册成功后返回 Cpy（企业结构体，包含企业信息的六个字段和 ret_code），回传给前端。其中：

后端处理 get /register 的代码如下

```
// 将 register 请求映射到 register 函数
@RequestMapping("/register")
public String register() {
    return ResourceReader.readFile("src/main/front/register.html");
}
```

前端提交注册的代码如下：

```
// 请求注册
this.$http.get(
    '/register/' + company_name + "/" + company_address + "/" + company_asset
).then(function(res) {
   // 返回 Cpy 结构体，解析相关参数
    var json_obj = JSON.parse(res.bodyText);
    if (json_obj.ret_code == "0") {
        var header = "<h>账户信息</h>";
        var account = "<p>账户地址: " + json_obj.account + "</p>";
        var public_key = "<p>公钥: " + json_obj.public_key + "</p>";
        var private_key = "<p>私钥: " + json_obj.private_key + "</p>";
        // 注册成功后点击登录
        var jumper = "<p><a href='http://127.0.0.1:8080/login'>点击此处跳转登录...</a></p>"
```

```
            document.write(header + account + public_key + private_key + jumper);
        } else {
            document.getElementById("sign").innerHTML = "Failed to register!";
        }
    },
```

后端处理/register/{company_name}/{company_address}/{company_asset}的代码如下：

```
    @RequestMapping("/register/{company_name}/{company_address}/{company_asset}")
    public Cpy doRegister(@PathVariable("company_name") String company_name,
@PathVariable("company_address") String company_address,
@PathVariable("company_asset") String company_asset) throws Exception
    {
        Cpy cpy = c.register(company_name, company_address, company_asset);
        return cpy;
    }
```

（因为此后功能的前后端代码大同小异，为了观感舒适，此后不予展示。）

链端处理 register 的代码如下：

```
    public Cpy register(String company_name, String company_address, String asset_value)
throws Exception
    {
        // 新建结构体以供返回
        Cpy ret = new Cpy();
        ret.setCompany_address(company_address);
        ret.setCompany_name(company_name);
        ret.setAsset_value(Integer.valueOf(asset_value));
        // 加载 company 合约
        Company company = Company.load(contractAddress, web3j, credentials, new
StaticGasProvider(gasPrice, gasLimit));
        // 注册前先验证 company_name 是否存在
        TransactionReceipt receipt = company.hasCompany(company_name).send();
        // 解码 hasCompany 函数的输出
        BigInteger big = company.getHasCompanyOutput(receipt).getValue1();
        System.out.println("register: " + big.intValue());

        if(big.compareTo(new BigInteger("0")) == 0)
        {
            // 若不存在，则 ret_code=0，代表可以注册
            ret.setRet_code(0);
            // 创建一个新的普通外部账户，并通过 company.register 函数将其添加到表中
            EncryptType.encryptType = 0;
            Credentials credentials2 = GenCredential.create();
            // 获取账户地址、公钥、私钥
            ret.setAccount(credentials2.getAddress());
            ret.setPrivate_key(credentials2.getEcKeyPair().getPrivateKey().toString(16));
            ret.setPublic_key(credentials2.getEcKeyPair().getPublicKey().toString(16));

            receipt = company.register(company_name, company_address,
                    new BigInteger(asset_value), ret.getAccount(), ret.getPrivate_key(),
ret.getPublic_key()).send();
            big = company.getRegisterOutput(receipt).getValue1();
        }
        else {
            // 若存在，则不能注册
```

```
            ret.setRet_code(big.intValue());
        }
        // 返回 Cpy 结构，前端可以通过 ret_code 属性判断是否注册成功。(对于所有注册、登录等功能，我设定了 0 代表成功，其它代表各种错误。由于时间关系，前端暂时没有根据 ret_code 的不同给用户解释失败原因)
        return ret;
    }
```

（因为 solidity 代码在上次作业已经做过，这次作业只是做了一些微调，修复了一些 bug。）

登录功能：浏览器访问 localhost:8080/login 进入登录页面（后端返回 login.html 资源），可以填写企业名称、私钥从而提交表单(前端)，传到 spring-boot 的 WebController 中(后端)，解析参数并传给链端执行 register 操作，登录成功后返回 Cpy，回传给前端。其中

链端处理 login 的代码如下：

```
public Cpy login(String company_name, String privateKey) throws Exception
{
    Cpy ret = new Cpy();
    ret.setCompany_name(company_name);
    ret.setPrivate_key(privateKey);

    Company company = Company.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
    // 登录前先验证是否可以登录
    TransactionReceipt receipt = company.canLogin(company_name, privateKey).send();
    BigInteger big = company.getHasCompanyOutput(receipt).getValue1();
    System.out.println("login: " + big.intValue());

    if (big.compareTo(new BigInteger("0")) == 0)
    {
        // 如果可以登录，则返回企业相关信息
        ret.setRet_code(0);
        receipt = company.login(company_name).send();
        Tuple3<BigInteger, String, String> tuple = company.getLoginOutput(receipt);
        ret.setAsset_value(tuple.getValue1().intValue());
        ret.setCompany_address(tuple.getValue2());
        ret.setPrivate_key(tuple.getValue3());
    }
    else {
        ret.setRet_code(big.intValue());
    }

    return ret;
}
```

新增应付款项：在用户主页中，开具应付款项。其中借方代表债权企业，贷方默认是自己。换句话说，己方不能胡编乱造自己为债权人的单据，因为 A 无法在未经 B 同意的情况下，让 B 承受债务。

链端处理 addBil()的代码如下：

```
public int addBil(String creditor, String debtor, String amount) throws Exception
{
    Company company = Company.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
    // 添加账单。如果{ creditor , debtor }不存在，则新增一条记录{ creditor , debtor , amount}，否则，更新 amount = old_amount + amount
    TransactionReceipt receipt = company.addBil(creditor, debtor, new BigInteger(amount)).send();
    BigInteger big = company.getAddBilOutput(receipt).getValue1();
    System.out.println("addBil: " + big.intValue());
    if (big.compareTo(new BigInteger("0")) == 0)
    {
        return 0;
    }
    else {
        return big.intValue();
    }
}
```

转移应收款项：在用户主页中，转移应收款项。其中己方(A)默认是转移债权的企业，借方(B)表示接受债权的企业，贷方(C)表示债务承担者。例如，C 欠 A 共 1000 元，那么 A 可以转移给 B 共 500 元的债权，最终结果导致 C 欠 A、B 各 500 元。

链端处理 transferBil ()的代码如下：

```
public int transferBil(String new_creditor, String creditor, String debtor, String amount) throws Exception
{
    Company company = Company.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
    System.out.println(new_creditor + creditor + debtor + amount);
    TransactionReceipt receipt = company.transferBil(new_creditor, creditor, debtor, new BigInteger(amount)).send();
    BigInteger big = company.getTransferBilOutput(receipt).getValue1();
    System.out.println("transferBil: " + big.intValue());

    if (big.compareTo(new BigInteger("0")) == 0) {
        return 0;
    }
    else {
```

```
            return big.intValue();
        }
    }
```

支付应付款项：在用户主页中，比如有一条己方欠企业 B 共 100 元的债务，那么己方可以支付全部或部分债务并扣除相关资产（债务全部支付之后，t_bil 中应付款项的记录也会随之删除）

链端处理 repayBil ()的代码如下：

```
    public int repayBil(String creditor, String debtor, String amount) throws Exception
    {
        Company company = Company.load(contractAddress, web3j, credentials, new
StaticGasProvider(gasPrice, gasLimit));

        TransactionReceipt receipt = company.repayBil(creditor, debtor, new
BigInteger(amount)).send();
        BigInteger big = company.getRepayBilOutput(receipt).getValue1();
        System.out.println("repayBil: " + big.intValue());
        if (big.compareTo(new BigInteger("0")) == 0)
        {
            return 0;
        }
        else {
            return big.intValue();
        }
    }
```

融资：在用户主页中，根据己方所持有的债权总额 s，可以向银行请求融资 a(a <= s)。

链端处理 finance ()的代码如下：

```
    public int finance(String company_name, String bank_name, String amount) throws
Exception
    {
        Company company = Company.load(contractAddress, web3j, credentials, new
StaticGasProvider(gasPrice, gasLimit));

        TransactionReceipt receipt = company.finance(company_name, bank_name, new
BigInteger(amount)).send();
        BigInteger big = company.getFinanceOutput(receipt).getValue1();
        System.out.println("finance: " + big.intValue());
        if (big.compareTo(new BigInteger("0")) == 0)
        {
            return 0;
        }
        else {
            return big.intValue();
        }
    }
```

三、 功能测试及界面展示

注册：注册 A，B、C 成功

账户信息

账户地址: 0x5cfe13a7d9c7ff00b427b32a235ae93d5cc6905f

公钥: 490fe10e2379375de875f97a0f1f5a1b89b3becf27a15f12cdc2e5559255738f59a05e02cfd285a231d643b487815e8f65740c55e9092aa6590706c7c5f49ef5

私钥: cc1177f36a851183888ee2e03f413e1b6a79c3ed13b5056926e554edb1554040

点击此处跳转登录...

账户信息

账户地址: 0x4d41db18e5717c2e523a564a94c12a56040aaf43

公钥: 753e703774e9b3929ae527716da4047588ed4cbd9264e852e30096bc22f6892c4cabab3c00149aefcd7436c2525105c2f411a5f2fab9ec98f58fc89a0348a3e0

私钥: fc8666c91b362cf96e61efc58cffbd594e22759a5d0b9d9308dd6b3a69b1e5a5

点击此处跳转登录...

登录：错误秘钥登录 B 失败，正确秘钥登录 A 成功。

四大功能： A 开具结单、B 将借单转移给 C、C 融资、A 偿还借款给 C

四、 心得体会

难点：对我来说，本次作业最主要的难点在于如何使 fisco 的 sdk 与 webserver 建立连接。第一次，我尝试使用 apach tomcat 架构构建，但是始终没有办法将 sdk 中生成的 java 文件揉入到 tomcat 项目。后来采用 spring-boot，花了两天时间才解决路径和 build.gradle 的配置问题。

学习：通过这次作业，我学会了如何将链端与后端结合起来，并对合约的使用有了更深的理解。此外，我还学会了有关 web 的一些知识，比如不同 url 的解析，Vue.js 框架的使用等。

作业要求：

1. 命名要求：学号_姓名，例如 16340000_林 XX。
2. 实验报告提交格式为 pdf。
3. 实验内容不允许抄袭，我们要进行代码相似度对比。如发现抄袭，按 0 分处理。