

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)

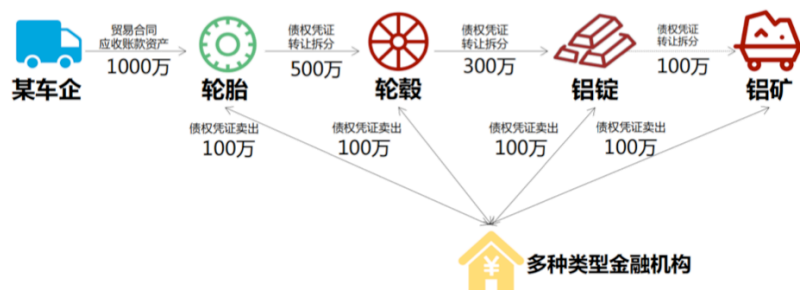
课程名称：区块链原理与技术

任课教师：郑子彬

年级	2017 级	专业（方向）	软件工程
学号	16352091	姓名	温海霖
电话	*****	Email	wenhlin@mail2.sysu.edu.cn
开始日期	2019.12.09	完成日期	2019.12.12

一、项目背景

项目要求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（<https://github.com/FISCOBCOS/FISCO-BCOS>）。



传统供应链金融场景中，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。在接下来的几个月里，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。

而区块链+供应链金融场景中，会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。

本设计基于 FISCO-BCOS 的供应链金融-区块链应用有如下几个功能的实现目标：

- (1) 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。
- (2) 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。
- (3) 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。
- (4) 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。
- (5) 加分项：使用 Java 应用构建了友好高效的用户界面、实现了链上数据的隐私保护、实现了自定金额的单据转让、设置了许多异常提示。

二、 方案设计

存储设计：

在 FISCO-BCOS 链内部部署的智能合约中，使用结构体来存储每一个企业的信息，包括：企业编号、企业名称、企业哈希、企业账额、企业单据收入（其他企业签订单据的欠款）、企业单据支出（签订单据的欠款）、企业类型、企业存活变量、企业收据表。同样的，每一份单据也使用结构体存储，包括：单据金额，签订双方的企业哈希、单据存活变量，具体声明如下：

```
enum CompanyType {
    Bank,
    CarCompany,
    TireCompany,
    HubCompany
}

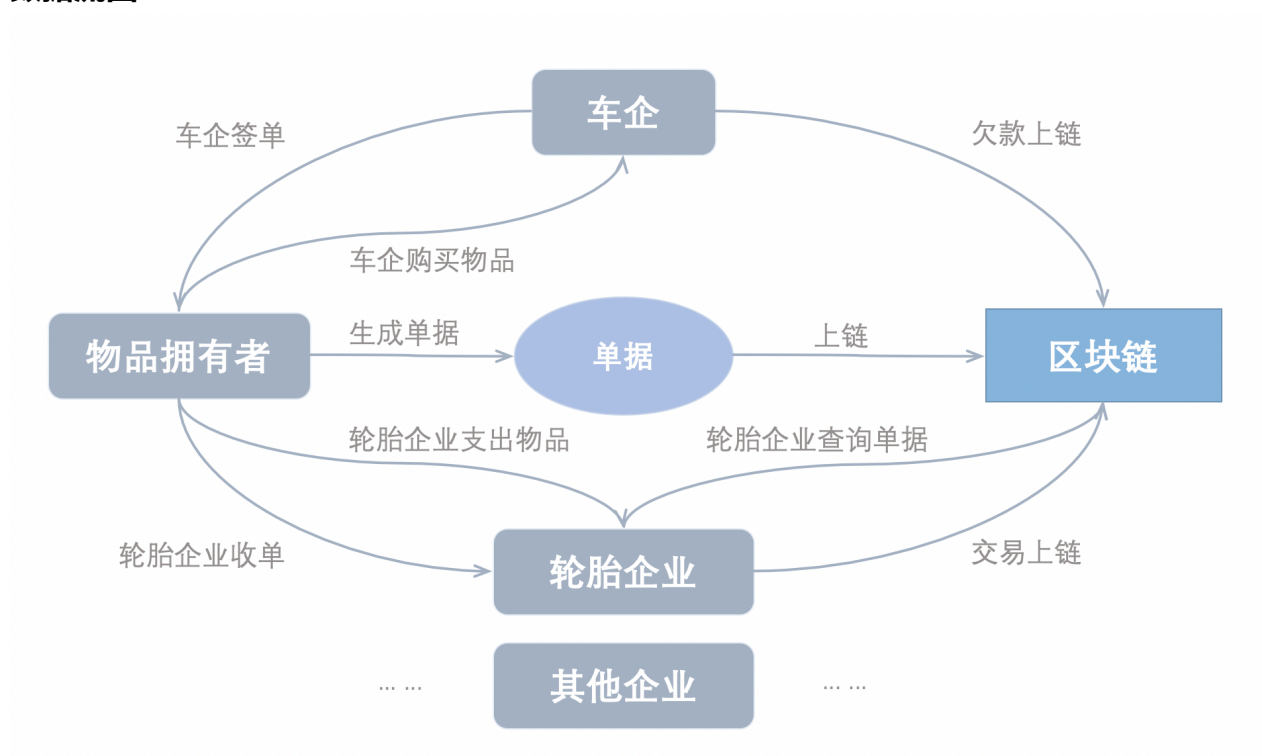
struct Company {
    uint256 company_id;
    bytes32 company_name;
    address company_address; // address->uint160
    uint256 company_asset;
    uint256 company_receipts_in;
    uint256 company_receipts_out;
    CompanyType company_type;
    bool valid; // company is alive?
    mapping (address => Receipt) from_receipts;
    // mapping (address => Receipt) to_receipts;
}

struct Receipt {
    // bytes32 hash;
    uint256 amount;
    address from;
    address to;
    bool valid; // receipt is alive?
}
```

合约中还使用了映射来存储哈希和企业、编号和企业之间的关系：

```
mapping(address => Company) public addressToCompany;  
mapping(uint256 => Company) public idToCompany;
```

数据流图：



完成过程：

该部分主要分析项目的架构、以及构建和运行过程，关键部分的智能合约、后端设计、前端设计在“核心功能介绍”中阐述。首先是项目运行的环境，如下所示：

环境	版本
MacOS	10.13
MySQL	MySQL-5.6
MySQL-python	Python2，已安装
Java	OpenJDK13

然后是链端的创建，生成一条单群组的 FISCO 链，过程如下：

准备环境

安装 FISCO-BCOS 的环境为 macOS10.14, 首先安装依赖, 运行命令:

```
brew install openssl curl
```

然后创建操作目录:

```
cd ~ && mkdir -p fisco && cd fisco
```

并下载 build_chain.sh 脚本:

```
bash <(curl -s https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master/tools/get_buildchain.sh)
```

私有链的搭建: 单群组4节点

在fisco目录下执行下面的指令, 生成一条单群组4节点的FISCO链:

```
bash build_chain.sh -l "127.0.0.1:4" -p 30300,20200,8545
```

构建 FISCO BCOS 的区块链应用, 需要通过 FISCO BCOS 提供的 SDK 来调用合约接口, 本项目选择调用的是 FISCO BCOS 的 Java SDK, 该 SDK 提供调用 FISCO BCOS JSON-RPC 的 Java API, 并且支持预编译合约管理区块链, 目标应用即为 Java 应用。

可以先使用 FISCO BCOS 教程中的初始 Java 工程项目供本项目开发使用, 首先获取 Java 工程项目:

```
# 获取Java工程项目压缩包
$ cd ~
$ curl -LO https://github.com/FISCO-BCOS/LargeFiles/raw/master/tools/asset-app.tar.gz
# 解压得到Java工程项目asset-app目录
$ tar -zxvf asset-app.tar.gz
```

项目拉取完毕后, 需要对文件进行增改, 目录 ASSET-APP/下的 build.Gradle 中增添加速镜像、文件指引等内容:

```

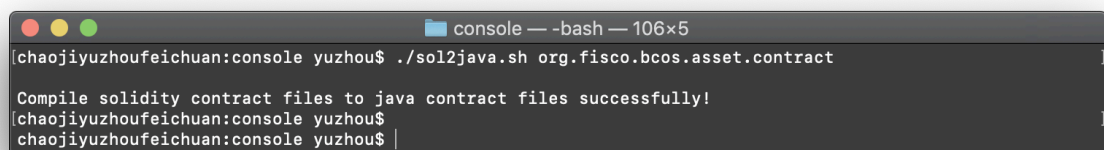
jar {
    destinationDir file('dist/apps')
    archiveName project.name + '.jar'
    exclude '**/*.xml'
    exclude '**/*.properties'
    exclude '**/*.crt'
    exclude '**/*.key'

    doLast {
        copy {
            from configurations.runtime
            into 'dist/lib'
        }
        copy {
            from file('src/test/resources/')
            into 'dist/conf'
        }
        copy {
            from file('tool/')
            into 'dist/'
        }
        copy {
            from file('src/test/resources/contract')
            into 'dist/contract'
        }
    }
}

```

接着，把链端 my_fisco_bcos_chain 下所需要的证书文件全部复制到目录 ASSET-APP/src/main/resources/下。

在目录 ASSET-APP/src/main/java/org/fisco/bcos/asset/client/下，需添加负责后端的 SupplyChainCtrl.java 文件、负责前端的 MyDApp.java 文件，而 AssetClient.java 文件为自带的，没有用处，可以忽略。我们可以把智能合约文件放在目录 ASSET-APP/src/main/resources/contract/中，合约编写完成后复制到链端目录 ASSET-APP/my_fisco_bcos_chain/console/contracts/solidity/中，然后再将合约编译为 Java 类，编译在目录 ASSET-APP/my_fisco_bcos_chain/console/中执行，编译成功后，再将生成的 Java 文件放到目录 ASSET-APP/src/main/java/org/fisco/bcos/asset/contract/下。



```

console — -bash — 106x5
[chaojiyuzhoufeichuan:console yuzhou$ ./sol2java.sh org.fisco.bcos.asset.contract
]
Compile solidity contract files to java contract files successfully!
[chaojiyuzhoufeichuan:console yuzhou$
[chaojiyuzhoufeichuan:console yuzhou$

```

再需要更改目录 ASSET-APP/tool/asset_run.sh 脚本：

```
java -Djdk.tls.namedGroups="secp256k1" -cp 'apps/*:conf/:lib/*' org.fisco.bcos.asset.client.MyDApp $@
```

至此，整个项目部署完毕，各部分代码编写完成后，在目录 ASSET-APP/执行 gradle build 命令进行编译（如果下载 Web3SDK 的依赖 solcJ-all-0.4.25.jar 速度过慢，可以手

动下载 <https://github.com/FISCO-BCOS/LargeFiles/raw/master/tools/solcj/solcJ-all-0.4.25.jar> , 再放置到/Users/mac_username/.gradle/caches/modules-2/files-2.1中), 再于目录 ASSET-APP/dist/下执行 `bash asset_run.sh deploy` 命令即可启动 Java 区块链应用:



```
输出 调试控制台 终端
chaojiyuzhoufeichuan:asset-app yuzhou$ gradle build
BUILD SUCCESSFUL in 5h 30m 6s
4 actionable tasks: 4 up-to-date
chaojiyuzhoufeichuan:asset-app yuzhou$ 
chaojiyuzhoufeichuan:dist yuzhou$ bash asset_run.sh deploy
```

核心功能介绍:

该部分主要分析项目的智能合约、后端设计、前端设计等内容。

智能合约的基本功能: 通过传入新的企业名称, 可以创建新的企业; 企业登录账号后, 可以查询自己账户的余额, 且根据欠款方的企业哈希, 可以查询欠款单据的金额。

```
function registerCompany(bytes32 company_name, uint8 company_type) {
    company_id_count = company_id_count + 1;
    idToCompany[company_id_count] = Company(company_id_count, company_name, msg.sender,
                                              50000, 0, 0, CompanyType(company_type), true);
    addressToCompany[msg.sender] = idToCompany[company_id_count];
}

function getCompanyAsset() public returns(uint256) {
    return addressToCompany[msg.sender].company_asset;
}

function getReceiptAmount(address to_address) public returns(uint256) {
    if (addressToCompany[msg.sender].from_receipts[to_address].valid) {
        return addressToCompany[msg.sender].from_receipts[to_address].amount;
    }
    return 0;
}
```

智能合约的签订单据功能: 企业登录账号后, 可以用自己的账户向其他企业签订单据, 相当于借款, 需要传入借款方的企业哈希, 以及单据的金额, 此处需要注意检查双方是否已有存活的单据。

```
function signReceipt(address from_temp, uint256 amount) public {
    // from -> this
    // this receipt is alive?
    if (addressToCompany[from_temp].from_receipts[msg.sender].valid == true) {
        addressToCompany[from_temp].from_receipts[msg.sender].amount += amount;
    } else {
        addressToCompany[from_temp].from_receipts[msg.sender] = Receipt(amount, from_temp, msg.sender);
    }
    addressToCompany[msg.sender].company_receipts_in += amount;
    addressToCompany[from_temp].company_receipts_out += amount;
}
```

智能合约的转让单据功能: 企业登录账号后, 可以将单据部分拿来转让, 即将自身欠款的单据去抵消自身借出款的单据, 并让借出款的签订方和欠款的支出方之间再生成一笔

单据，抵消的金额可以不是最大金额，传入一个参数即可，函数需检查该参数。

```
function transferReceipt(address from_temp, address to_temp, uint256 amount) public {
    // from -> mid/this ->to
    address mid_temp = msg.sender;
    // these receipts are alive?
    if (addressToCompany[from_temp].from_receipts[mid_temp].valid == true
        && addressToCompany[mid_temp].from_receipts[to_temp].valid == true) {
        // from -a1-> mid -a2-> to
        // => from -(a1-a)-> mid
        // mid -(a2-a)-> to
        // from -a-> to
        // => a1>a, a2>a
        if (addressToCompany[from_temp].from_receipts[mid_temp].amount >= amount
            && addressToCompany[mid_temp].from_receipts[to_temp].amount >= amount) {
            addressToCompany[from_temp].from_receipts[mid_temp].amount -= amount;
            addressToCompany[mid_temp].from_receipts[to_temp].amount -= amount;
            // amount == 0?
            if (addressToCompany[from_temp].from_receipts[mid_temp].amount == 0) {
                addressToCompany[from_temp].from_receipts[mid_temp].valid = false;
            }
            if (addressToCompany[mid_temp].from_receipts[to_temp].amount == 0) {
                addressToCompany[mid_temp].from_receipts[to_temp].valid = false;
            }
            addressToCompany[mid_temp].company_receipts_in -= amount;
            addressToCompany[mid_temp].company_receipts_out -= amount;
            // from -> to exist?
            if (addressToCompany[from_temp].from_receipts[to_temp].valid == true) {
                addressToCompany[from_temp].from_receipts[to_temp].amount += amount;
            } else {
                // to -> from?
                // Not for the moment, bill cancellation should be implemented in a separate funct
                addressToCompany[from_temp].from_receipts[to_temp] =
                    Receipt(amount, from_temp, to_temp, true);
            }
        }
    }
}
```

智能合约有根据单据向银行申请融资的功能：可以利用应收账款单据向银行申请融资，判断申请是否成功的条件现实中有很多种，下列代码是一种思路：

```
function financingWithReceipt(uint256 amount) public returns(bool) {
    // Financing failure
    if (addressToCompany[msg.sender].company_receipts_out < amount ||
        bank.valid == false ||
        bank.company_asset < amount) {
        return false;
    }
    // bank -> this exist?
    if (bank.from_receipts[msg.sender].valid == true) {
        bank.from_receipts[msg.sender].amount += amount;
    } else {
        bank.from_receipts[msg.sender] = Receipt(amount, bank.company_address,
            msg.sender, true);
    }
    addressToCompany[msg.sender].company_receipts_in += amount;
    bank.company_receipts_out += amount;
    return true;
}
```

智能合约的单据结算功能：企业登录账号后，可以选择另一个企业并向其支付相应的单据欠款，函数中需要注意的是欠款还清后应把单据的金额清零，并灭活处理。

```
function clearReceipt(address from_temp) public returns(bool) {
    // from -> this => this -payback-> from
    // Payback failure
    if (addressToCompany[from_temp].from_receipts[msg.sender].valid == false ||
        addressToCompany[msg.sender].company_asset <
            addressToCompany[from_temp].from_receipts[msg.sender].amount) {
        return false;
    }
    uint256 thisAmount = addressToCompany[from_temp].from_receipts[msg.sender].amount;
    addressToCompany[from_temp].from_receipts[msg.sender].amount = 0;
    addressToCompany[from_temp].from_receipts[msg.sender].valid = false;
    addressToCompany[msg.sender].company_asset -= thisAmount;
    addressToCompany[from_temp].company_asset += thisAmount;
    addressToCompany[msg.sender].company_receipts_in -= thisAmount;
    addressToCompany[from_temp].company_receipts_out -= thisAmount;
    return true;
}
```

后端设计中，需要引入合约类，并需要初始化服务、部署该合约，部署时调用合约类中的 deploy 函数即可，关键内容如下：

```
import org.fisco.bcos.asset.contract.VehicleSupplyChain;

public void initialize() throws Exception {

    // init the Service
    @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 = Web3j.build(channelEthereumService, 1);

    // init Credentials
    Credentials credentials = Credentials.create(Keys.createEcKeyPair());

    setCredentials(credentials);
    setWeb3j(web3j);

    logger.debug(" web3j is " + web3j + " ,credentials is " + credentials);
}

public void deployAssetAndRecordAddr() {

    try {
        VehicleSupplyChain asset = VehicleSupplyChain.deploy(web3j, credentials, new StaticGasProvider(gasPrice, gasLimit)).send();
        System.out.println(" deploy Asset success, contract address is " + asset.getContractAddress());

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

后端设计中，在函数 ctrlInit()中调用上述函数，并注册除银行外的三个公司账户（银行行为特殊企业，在合约的构造函数中就已注册）：


```

public void ctrlInit() {
    try {
        initialize();
        deployAssetAndRecordAddr();
        String contractAddr = loadAssetAddr();
        VehicleSupplyChain contr = VehicleSupplyChain.load(contractAddr, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        contr.registerCompany("CarCompany").send();
        contr.registerCompany("TireCompany").send();
        contr.registerCompany("HubCompany").send();
    } catch (Exception e) {
        System.out.println("error: "+e.getMessage());
    }
}
}

```

后端设计中，在函数 getAssetCtrl()中查询企业的账户余额，需要对收到对 output 内容进行处理，再返回给前端。

```

public String getAssetCtrl() {
    try {
        String contractAddr = loadAssetAddr();
        VehicleSupplyChain contr = VehicleSupplyChain.load(contractAddr, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        //
        TransactionReceipt result = contr.getCompanyAsset(username).send();
        return formatInt(result.getOutput());
    } catch (Exception e) {
        System.out.println("error: "+e.getMessage());
    }
    return "0";
}
}

```

后端设计中，在函数 getReceiptAmountCtrl 中根据另一企业名称查询其签订过的单据，即查询自身借出了多少金额，代码如下

```

public String getReceiptAmountCtrl(String to_company_name) {
    try {
        String contractAddr = loadAssetAddr();
        VehicleSupplyChain contr = VehicleSupplyChain.load(contractAddr, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        TransactionReceipt result = contr.getReceiptAmount(username, to_company_name).send();
        return formatInt(result.getOutput());
    } catch (Exception e) {
        System.out.println("error: "+e.getMessage());
    }
    return "0";
}
}

```

后端设计中，分别在四个自定义函数 signReceiptCtrl()、transferReceiptCtrl()、financingWithReceiptCtrl()、clearReceiptCtrl()中完成四个应用功能，代码逻辑与上面几段代码类似。

前端设计中，主要是 UI 的布局，总体布局完成后通过调用各个函数来完成 Java 应用界面内，各行(JPanel)的 JLabel、JtextField、JButton 各元素的摆放，其中触发 JButton 事件时会调用后端接口，完成用户操作。

```

public MyDApp() {
    super("My DApp");
    this.setSize(700, 600);
    this.setBackground(Color.WHITE);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setLayout(new GridLayout(17,5));
    //
    loginAndAssetPanel();
    serchReceiptPanel();
    functionalTestLabelPanel();
    signReceiptPanel();
    transferReceiptPanel();
    financingWithReceiptPanel();
    clearReceiptPanel();
    this.setVisible(true);
    //
    supplyChainCtrl = new SupplyChainCtrl();
    supplyChainCtrl.ctrlInit();
}
}

```

```
Run | Debug
public static void main(String[] args) {
    MyDApp my_DApp = new MyDApp();
}
```

三、 功能测试(过程请参见录制的视频)

功能测试 1:

登录车企账户

-> SignReceipt -> 车企向轮胎公司签订 1000 单据 -> 确定

----- MyApp Login -----

Company Name:

Company Asset: 50000

----- Query Receipt -----

To Company Name:

Receipt Amount: 0

----- Functional Test -----

----- Sign Receipt -----

From Company Name Receipt Amount

登录轮胎公司账户

-> 查询车企签订的单据 -> 1000? -> 成功

----- MyApp Login -----

Company Name:

Company Asset: 50000

----- Query Receipt -----

To Company Name:

Receipt Amount: 1000

功能测试 2:

登录轮胎公司账户

-> SignReceipt -> 轮胎公司向轮毂公司签订 700 单据 -> 确定

登录轮毂公司账户

-> 查询轮胎公司签订的单据 -> 700? -> 成功

----- MyApp Login -----

Company Name:

Company Asset: 50000

----- Query Receipt -----

To Company Name:

Receipt Amount: 700

-> Hub-700->Tire-1000->Car

登录轮胎公司账户

-> TransferReceipt -> 转让车企、轮毂 600 的单据 -> 确定

----- MyApp Login -----

Company Name:

Company Asset: 50000

----- Query Receipt -----

To Company Name:

Receipt Amount: 700

----- Functional Test -----

----- Sign Receipt -----

From Company Name Receipt Amount

----- Transfer Receipt -----

From CN To CN Amount

-> Tire-400->Car、Hub-600->Car、Hub-100->Tire

-> 查询车企签订的单据 -> 400? -> 成功

----- MyApp Login -----

Company Name:

Company Asset: 50000

----- Query Receipt -----

To Company Name:

Receipt Amount: 400

登录轮毂公司账户

-> 查询车企签订的单据 -> 600? -> 成功

----- MyApp Login -----

Company Name:

Company Asset: 50000

----- Query Receipt -----

To Company Name:

Receipt Amount: 600

-> 查询轮胎公司签订的单据 -> 100? -> 成功

----- MyApp Login -----

Company Name:

Company Asset: 50000

----- Query Receipt -----

To Company Name:

Receipt Amount: 100

功能测试 3:

登录轮毂公司账户

-> FinancingWithReceipt -> 向银行发起 500 融资申请

----- Financing With Receipt -----

Amount

-> Failure?

-> Success?

----- Financing With Receipt -----

Amount

-> FinancingWithReceipt -> 向银行发起 1000000 融资申请

----- Financing With Receipt -----

Amount

-> Success?

-> Failure?

----- Financing With Receipt -----

Amount Failure!

return

功能测试 4:

登录车企账户

-> ClearReceipt -> 车企跟轮胎公司结算单据

----- Clear Receipt -----

Company Name TireCompany

return

-> Tire-400->Car

----- Clear Receipt -----

Company Name Success!

return

-> 查询账户余额 -> 50000-400? -> 成功

----- MyApp Login -----

Company Name: CarCompany

login

Company Asset: 49600

登录轮胎公司账户

-> 查询账户余额 -> 50000+400? -> 成功

----- MyApp Login -----

Company Name: TireCompany

login

Company Asset: 50400

-> 查询车企签订的单据 -> 0? -> 成功

----- MyApp Login -----

Company Name: TireCompany

login

Company Asset: 50400

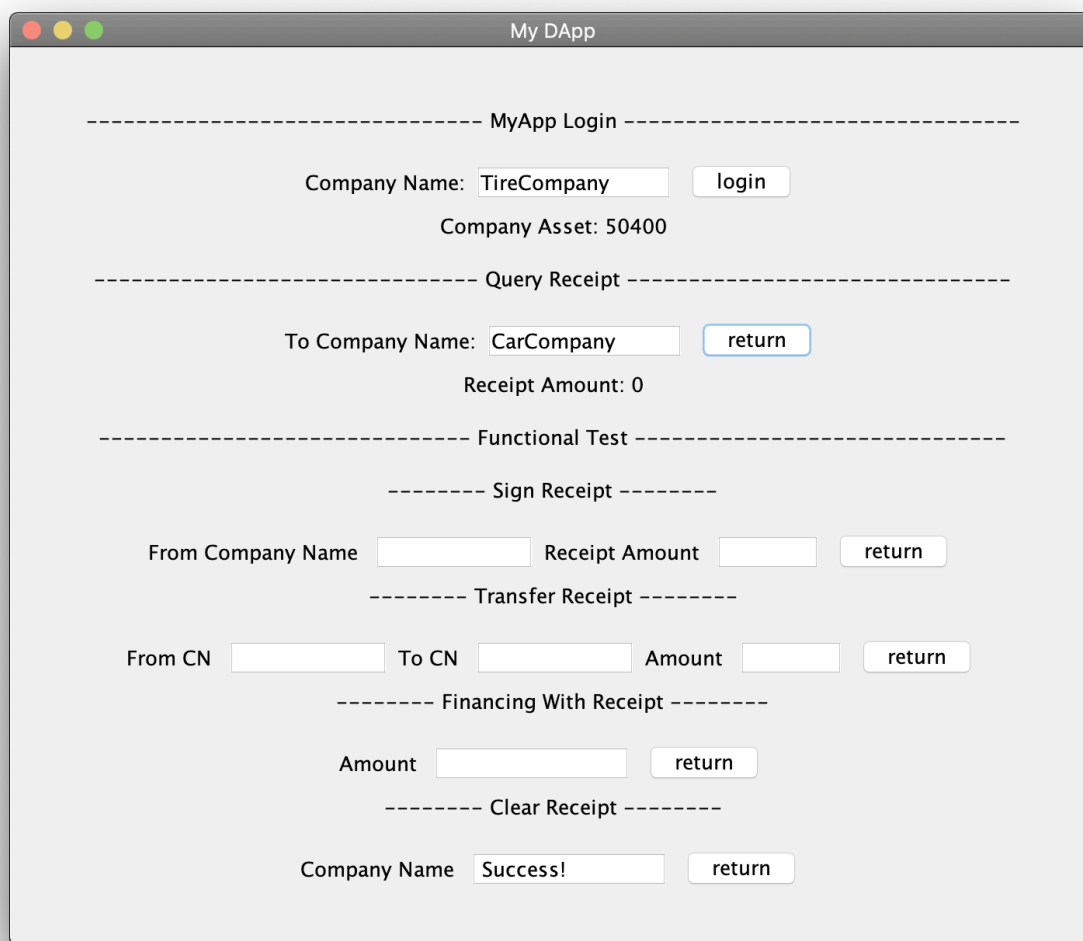
----- Query Receipt -----

To Company Name: CarCompany

return

Receipt Amount: 0

四、 界面展示



The screenshot displays a web application titled "My DApp" with a light gray background and a dark gray header bar. The interface is organized into several sections, each separated by dashed lines:

- MyApp Login**: Contains a "Company Name" input field with the value "TireCompany", a "login" button, and a "Company Asset" label with the value "50400".
- Query Receipt**: Contains a "To Company Name" input field with the value "CarCompany", a "return" button, and a "Receipt Amount" label with the value "0".
- Functional Test**: This section contains three sub-forms:
 - Sign Receipt**: Includes "From Company Name" and "Receipt Amount" input fields, a "return" button.
 - Transfer Receipt**: Includes "From CN", "To CN", and "Amount" input fields, a "return" button.
 - Financing With Receipt**: Includes an "Amount" input field and a "return" button.
- Clear Receipt**: Includes a "Company Name" input field with the value "Success!" and a "return" button.

五、 心得体会

本次作业写了一个基于已有的开源区块链系统 FISCO-BCOS、集成了服务端和客户端的供应链金融的 JAVA 应用，前几次作业已经让我学习到了很多关于区块链、智能合约、FISCO-BCOS 系统的很多东西，这一次的难题主要是如何使用好 FISCO-BCOS 的 SDK、如何把前端、后端、链端链接起来。刚开始做大作业项目构建的时候，我先是不自觉地使用了以太坊常用的 DApp 开发工具 Truffle，很方便但可惜写了很多之后才发现 Truffle 并不支持 FISCO-BCOS 链，虽然它也支持很多非以太坊的区块链平台。后来就直接使用 FISCO-BCOS 的 SDK，但不确定哪个比较好用就都试试水，先试了一下 nodeJS SDK，配置好之后却发现接口调用步步扎心、如履薄冰，再用了 Java SDK，发现简便易编程，就选定了它来完成整个项目。相关的 API 给我提供了很大的便利，在整个过程中也对

FISCO-BCOS 的内部架构有了一定的了解。项目也还有很多可以进一步完善的地方，比如 UI 可以更为美观，融资可以加大审核力度，改进区块链平台底层等等，之后会继续进行 DApp 的学习。