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	17343111	姓名	王子雄
电话	18782190594	Email	673730136@qq.com
开始日期	2019-11-17	完成日期	2019-12-11

一、 项目背景

基于区块链、智能合约等，实现基于区块链的供应链金融平台。

二、 方案设计（附加分项设计）

存储设计：

Receipt.sol 合约的实现需要引入 FISCO BCOS 提供的一个系统合约接口文件 Table.sol ，该系统合约文件中的接口由 FISCO BCOS 底层实现。

FISCO BCOS 提供合约 CRUD 接口开发模式，可以通过 Table.sol 合约创建表，并对创建的表进行增删改查操作。

Receipt.sol 合约中需要创建一个存储订单信息的表 Receipt，该表字段包含：

from_to：订单欠收款公司(string 类型)

Value：订单金额(int 类型)

```
constructor() public {
    // 构造函数中创建receipt表
    createTable();
}

function createTable() private {
    TableFactory tf = TableFactory(0x1001);
    // 资产管理表, key : from_to, field : value
    // |      公司名(主键)      |        订单金额       |
    // |-------------------- |-------------------|
    // |          from_to        |           value          |
    // |---------------------|-------------------|
    //
    // 创建表
    tf.createTable("receipt", "from_to", "value");
}

function openTable() private returns(Table) {
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("receipt");
    return table;
}
```

数据流图：

核心功能介绍：

功能一：实现采购商品——签发应收账款交易上链。

```
/*
描述 : 添加订单
参数 :
        from_to : 欠收款公司名
        value   : 订单金额
返回值：
        -0  添加订单成功
        -1 订单已存在，更新订单成功
        -2 其他错误
*/
function add(string from_to, uint256 value) public returns(int256){
```

首先通过查询函数 select 判断是否存在该订单，若不存在，则利用 insert 在表中添加该条订单；若存在，则利用 update 更新该订单，即加上新的欠款金额。

功能二：实现应收账款的转让上链。

```
    /*
    描述 : 转移欠款
    参数 :
            from_to1：初始欠收款公司
            from_to2：中间欠收款公司
            from_to3：目标欠收款公司
            value ： 转移金额
    返回值：
            -0  转移欠款成功
            -1 初始欠收款公司不存在
            -2 中间欠收款公司不存在
            -3 金额不足
            -4 其他错误
    */
    function transfer(string from_to1, string from_to2, string from_to3, uint256 value) public
returns(int256) {
```

首先通过查询函数 select 判断初始订单和中间订单是否存在，若不存在则返回相应数值；若存在则利用功能一的 add 函数更新新的订单，同时要减去初始订单和中间订单的金额，若为 0，则用功能四 remove 函数删去该订单；不为 0 则用 update 函数更新金额。

功能三：实现利用应收账款向银行融资上链。

该项功能其实本质与功能二相似，只不过把第三方改为银行，功能二是 B 拿着 A 的欠条找 C 打欠条，而功能三是 B 拿着 A 的欠条找银行融资，函数实现是一样的。

功能四：应收账款支付结算上链。

```
/*
描述 : 支付订单
参数 :
        from_to : 欠收款公司名
返回值：
        -0  支付订单成功
        -1 订单不存在
        -2 其他错误
*/
function remove(string from_to) public returns(int256) {
```

首先通过查询函数 select 判断需要支付的订单是否存在，若存在则用 remove 函数从表中移除该订单信息。

功能五：查询账单。

```
/*
描述 : 根据公司名查询订单
参数 :
        from_to : 欠收款公司名

返回值：
        参数一： 成功返回0，订单不存在返回-1
        参数二： 第一个参数为0时有效，订单金额
*/
function select(string from_to) public constant returns(int256, uint256) {
```

用 table 中的 select 函数查询关键字 from_to，并返回 from_to 和它的金额数值。

加分项：构建一个区块链 Java 应用

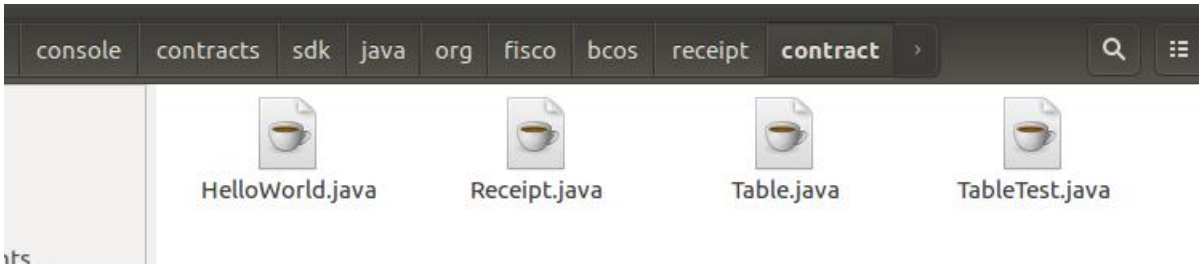
将 Solidity 合约转化成 Java 类

控制台提供了编译工具，利用 console 目录下提供的 sol2java.sh 脚本进行编译。

```
wzx123@ubuntu:~$ cd ~/fisco/console/
wzx123@ubuntu:~/fisco/console$ ./sol2java.sh org.fisco.bcos.receipt.contract

Compile solidity contract files to java contract files successfully!
```

运行成功之后，将会在 console/contracts/sdk 目录下生成 java、abi 和 bin 目录，在 java/org/fisco/bcos/receipt/contract 下有编译好的 java 文件。



同时，在 sdk 文件下创建一个 contracts 文件夹，将需要编译的合约拷贝到该目录下。

配置 Web3SDK（采用 Java SDK）

新建 receipt-app 文件夹作为区块链应用文件夹，进行 SDK 的环境配置。

根据官网教程，在 Java 应用的 gradle 配置文件 build.gradle 中添加以太坊的远程仓库，并通过 maven 引入 SDK 到 Java 应用

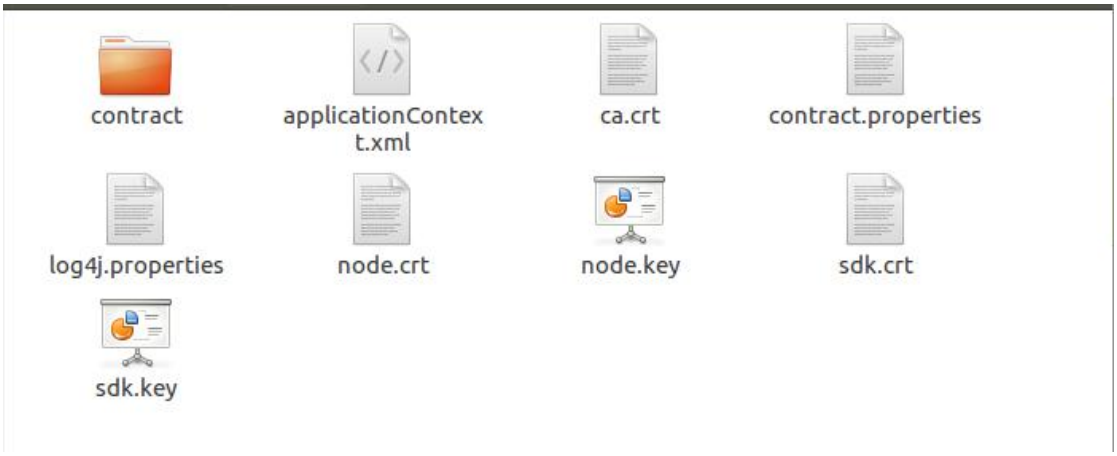
```
repositories {
    maven {
        url "http://maven.aliyun.com/nexus/content/groups/public/"
    }
    maven { url "https://dl.bintray.com/ethereum/maven/" }
    maven { url "https://oss.sonatype.org/content/repositories/snapshots" }
    mavenCentral()
}


List logger = [
        'org.slf4j:slf4j-log4j12:1.7.25'
]
```

再根据官网有关 Java SDK 配置的教程，配置以下文件（很多都提供了代码）：

- gradlew：Linux 或 Unix 下用于执行 wrapper 命令的 Shell 脚本
- applicationContext.xml：项目配置文件
- contract.properties：存储部署合约地址的文件
- log4j.properties：日志配置文件

区块链节点证书配置：拷贝区块链节点对应的 SDK 证书



构建一个应用，并集成 Web3SDK 到应用工程

设计一个类 ReceiptClient.java：该类通过调用 Receipt.java 实现对合约的部署和调用，路径为/src/main/java/org/fisco/bcos/receipt/client，初始化以及调用流程都在该类中进行。

下面介绍该类的设计思路：

recordReceiptAddr()：功能为存储合约的地址。

```
        public void recordReceiptAddr(String address) throws FileNotFoundException, IOException {
                Properties prop = new Properties();
                prop.setProperty("address", address);
                final Resource contractResource = new ClassPathResource("contract.properties");
                FileOutputStream fileOutputStream = new FileOutputStream(contractResource.getFile
());
                prop.store(fileOutputStream, "contract address");
        }
```

loadReceiptAddr()：功能为读取合约的地址。

```
        public String loadReceiptAddr() throws Exception {
                // load Asset contact address from contract.properties
                Properties prop = new Properties();
                final Resource contractResource = new ClassPathResource("contract.properties");
                prop.load(contractResource.getInputStream());

                String contractAddress = prop.getProperty("address");
                if (contractAddress == null || contractAddress.trim().equals("")) {
                        throw new Exception(" load Receipt contract address failed, please deploy
it first. ");
                }
                logger.info(" load Receipt address from contract.properties, address is {}",
contractAddress);
                return contractAddress;
        }
```

Initialize()：初始化函数，主要功能为构造 Web3j 与 Credentials 对象，这两个对象在创建对应的合约类对象(调用合约类的 deploy 或者 load 函数)时需要使用。

```
        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);
        }
```

deployReceiptAndRecordAddr()：功能为部署合约上链。

```
        public void deployReceiptAndRecordAddr() {

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

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

selectReceiptValue()：功能为作为接口，调用合约中 select 查询函数。

```
        public void selectReceiptValue(String from_to) {
                try {
                        String contractAddress = loadReceiptAddr();

                        Receipt receipt = Receipt.load(contractAddress, web3j, credentials, new
StaticGasProvider(gasPrice, gasLimit));
                        Tuple2<BigInteger, BigInteger> result = receipt.select(from_to).send();
                        if (result.getValue1().compareTo(new BigInteger("0")) == 0) {
                                System.out.printf(" from_to %s, value %s \n", from_to,
result.getValue2());
                        } else {
                                System.out.printf(" %s from_to is not exist \n", from_to);
                        }
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        // e.printStackTrace();
                        logger.error(" selectReceiptValue exception, error message is {}",
e.getMessage());

                        System.out.printf(" select receipt value failed, error message is %s\n",
e.getMessage());
                }
        }
```

addReceiptAccount()：功能为作为接口，调用合约中 add 添加函数。

```
        public void addReceiptAccount(String from_to, BigInteger value) {
                try {
                        String contractAddress = loadReceiptAddr();

                        Receipt receipt = Receipt.load(contractAddress, web3j, credentials, new
StaticGasProvider(gasPrice, gasLimit));
                        TransactionReceipt receipt1 = receipt.add(from_to, value).send();
                        List<AddEventEventResponse> response = receipt.getAddEventEvents(receipt1);
                        if (!response.isEmpty()) {
                                if (response.get(0).ret.compareTo(new BigInteger("0")) == 0) {
                                        System.out.printf(" add receipt account success =>
receipt: %s, value: %s \n", from_to,
                                                        value);
                                } else if (response.get(0).ret.compareTo(new BigInteger("1")) == 0)
{
                                        System.out.printf(" receipt is already exit, update
receipt account success => "
                                                        + "receipt: %s, value: %s \n", from_to,
value);
                                } else {
                                        System.out.printf(" register asset account failed, ret
code is %s \n",
                                                        response.get(0).ret.toString());
                                }
                        } else {
                                System.out.println(" event log not found, maybe transaction not
exec. ");
                        }
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        // e.printStackTrace();

                        logger.error(" addReceiptAccount exception, error message is {}",
e.getMessage());
                        System.out.printf(" add receipt value failed, error message is %s\n",
e.getMessage());
                }
        }
```

transferReceipt()：功能为作为接口，调用合约中 transfer 转移函数。

```
        public void transferReceipt(String from_to1, String from_to2, String from_to3, BigInteger
value) {
                try {
                        String contractAddress = loadReceiptAddr();
                        Receipt receipt = Receipt.load(contractAddress, web3j, credentials, new
StaticGasProvider(gasPrice, gasLimit));
                        TransactionReceipt receipt1 = receipt.transfer(from_to1, from_to2,
from_to3, value).send();
                        List<TransferEventEventResponse> response = receipt.getTransferEventEvents
(receipt1);
                        if (!response.isEmpty()) {
                                if (response.get(0).ret.compareTo(new BigInteger("0")) == 0) {
                                        System.out.printf(" transfer success => from_to1: %s,
from_to2: %s, from_to3: %s, value: %s \n",
                                                        from_to1, from_to2, from_to3, value);
                                } else {
                                        System.out.printf(" transfer receipt account failed, ret
code is %s \n",
                                                        response.get(0).ret.toString());
                                }
                        } else {
                                System.out.println(" event log not found, maybe transaction not
exec. ");
                        }
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        // e.printStackTrace();

                        logger.error(" transferReceipt exception, error message is {}",
e.getMessage());
                        System.out.printf(" transfer receipt account failed, error message is %s
\n", e.getMessage());
                }
        }
```

编写一个 receipt_run.sh 简化项目的运行。

```
#!/bin/bash

function usage()
{
    echo " Usage : "
    echo "   bash receipt_run.sh deploy"
    echo "   bash receipt_run.sh select from_to "
    echo "   bash receipt_run.sh add from_to value "
    echo "   bash asset_run.sh transfer from_to1 from_to2 from_to3 value "
    echo " "
    echo " "
    echo "examples : "
    echo "   bash asset_run.sh deploy "
    echo "   bash asset_run.sh add  A_B  500 "
    echo "   bash asset_run.sh add  B_C  500 "
    echo "   bash asset_run.sh transfer  A_B B_C A_C 500 "
    echo "   bash asset_run.sh select A_B"
    echo "   bash asset_run.sh select A_C"
    exit 0
}

    case $1 in
    deploy)
            [ $# -lt 1 ] && { usage; }
            ;;
    add)
            [ $# -lt 3 ] && { usage; }
            ;;
    transfer)
            [ $# -lt 5 ] && { usage; }
            ;;
    select)
            [ $# -lt 2 ] && { usage; }
            ;;
    *)
        usage
            ;;
    esac

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

通过 Web3SDK 调用合约接口

首先编译代码，在 receipt-app 文件下生成一个 dist 文件

```
wzx123@ubuntu:~$ cd ~/receipt-app
wzx123@ubuntu:~/receipt-app$ ./gradlew build

BUILD SUCCESSFUL in 9s
4 actionable tasks: 1 executed, 3 up-to-date
```

dist 文件下有 receipt_run.sh，表明编译成功。

部署合约。

```
wzx123@ubuntu:~$ cd ~/receipt-app
wzx123@ubuntu:~/receipt-app$ cd dist
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh deploy
 deploy Receipt success, contract address is 0xcb2345e4b0436b9cd39c16c10d1090d9a
f3d3ae4
wzx123@ubuntu:~/receipt-app/dist$
```

三、 功能测试

添加 A 与 B 签订的账单，A 欠 B 500，成功。

```
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh add "A_B" 500
 add receipt account success => receipt: A_B, value: 500
```

添加 A 与 B 签订的账单，A 欠 B 500，该订单已存在，更新该订单成功。并且查询到此时 A 欠 B 1000。

```
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh add "A_B" 500
 receipt is already exit, update receipt account success => receipt: A_B, value:
 500
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "A_B"
 from_to A_B, value 1000
```

添加 B 与 C 签订的账单，B 欠 C 500，成功。

```
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh add "B_C" 500
 add receipt account success => receipt: B_C, value: 500
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "B_C"
 from_to B_C, value 500
wzx123@ubuntu:~/receipt-app/dist$
```

将 A 与 B 的账单金额转 500 到 B 与 C 的账单上，即 A 欠 B 500，并欠 C 500，同时 B 不再欠 C。

```
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "B_C"
 from_to B_C, value 500
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "A_B"
 from_to A_B, value 1000
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh transfer "A_B" "B_C" "A_C"
 500
 transfer success => from_to1: A_B, from_to2: B_C, from_to3: A_C, value: 500
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "A_B"
 from_to A_B, value 500
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "B_C"
 B_C from_to is not exist
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "A_C"
 from_to A_C, value 500
```

C 将 A 与 C 之间的账单拿去银行融资 500，此时则为 A 欠银行 500，A 与 C 之间无账单。

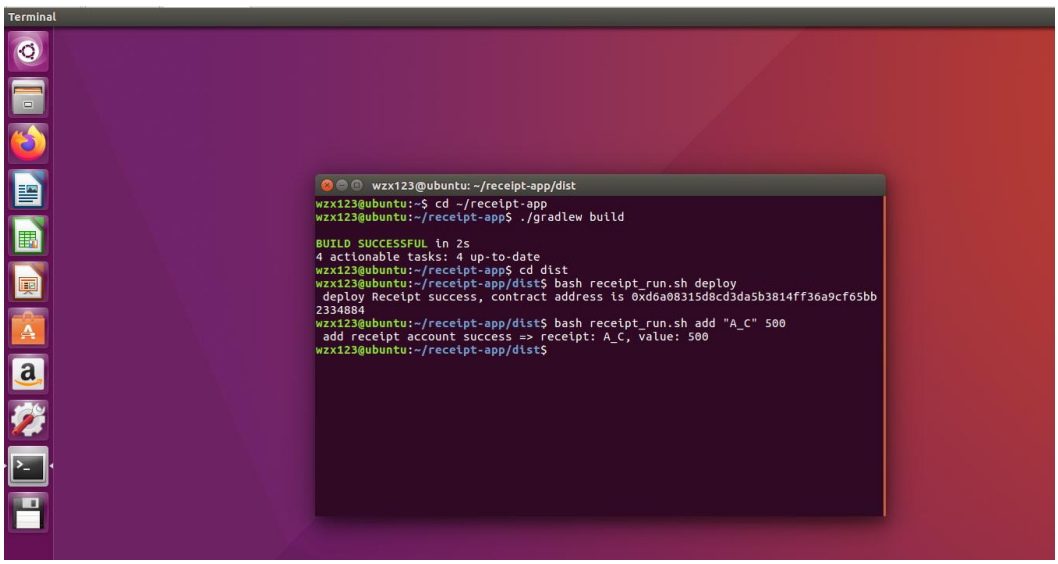
```
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh add "C_bank" 500
 add receipt account success => receipt: C_bank, value: 500
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh transfer "A_C" "C_bank" "A
_bank" 500
 transfer success => from_to1: A_C, from_to2: C_bank, from_to3: A_bank, value: 5
00
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "A_bank"
 from_to A_bank, value 500
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "A_C"
 A_C from_to is not exist
wzx123@ubuntu:~/receipt-app/dist$
ount Alice, value 50000
```

A 向银行还款，二者将不存在

```
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh remove "A_bank"
 remove receipt success
wzx123@ubuntu:~/receipt-app/dist$ bash receipt_run.sh select "A_bank"
 A_bank from_to is not exist
```

四、 界面展示

界面采用简单的命令行形式，编写一个 receipt_run.sh 文件简化项目的运行。



五、 心得体会

本次区块链大作业共分为三个阶段，历时两个月，经过一步步的学习完成了这次的大作业项目设计。

做完全部内容再回顾大作业的三个阶段，才发现大作业的布置非常人性化。本次大作业并不像其他科目一样直接布置项目，而是将任务分成三步，一步步引领我们学习并完成项目的编写。

第一阶段，名为前期热身，其实是让我们在虚拟机上安装和配置 FISCO BCOS，在安装和配置的过程中，也就熟悉了搭建、启动 FISCO 链以及部署、调用合约等基础操作。我在第一阶段并没有编写太复杂的 solidity 合约，只是模仿平台提供的 HelloWorld 编写了一个输出“mycontract”合约。

第二阶段，是需要我们编写项目的 solidity 合约，来实现项目的四个功能，在此之前其实我并没有编写过复杂的 solidity 合约，很多语法都不知道，因此只能从头学习。FISCO BCOS 平台的官网里有很多教程和实例，根据这些实例和教程我逐渐熟悉了 solidity 语法，我发现在 contracts 文件中提供了 table 这种数据结构的 solidity 合约，于是我采用 table 来存储 receipt 即账单，这种数据结构非常方便，提供了诸如 remove、insert 等接口，令项目的实现简化了许多。

第三阶段，是完善项目。在第二阶段，我仅是使用控制台部署写好的合约，于是在第三阶段我用 java 编写了一个应用程序，通过编写一个 ReceiptClient 类来实现对合约的部署和调用。这样就涉及到了 Java SDK 的配置问题，官网有实例，可以依照教程和实例配置，虽然遇到了各种小 bug，不过也跌跌撞撞解决了，最终完成了该应用程序。

本次大作业收获颇丰，亲身完成一个区块链应用的设计和制作，可以深入地了解区块链尤其是 FISCO BCOS 的工作原理，同时也掌握了 Java SDK 的配置。