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	17343138	姓名	杨淇淳
电话	18186292219	Email	11434682@qq.com
开始日期	2019.12.08	完成日期	2019.12.13

一、 项目背景

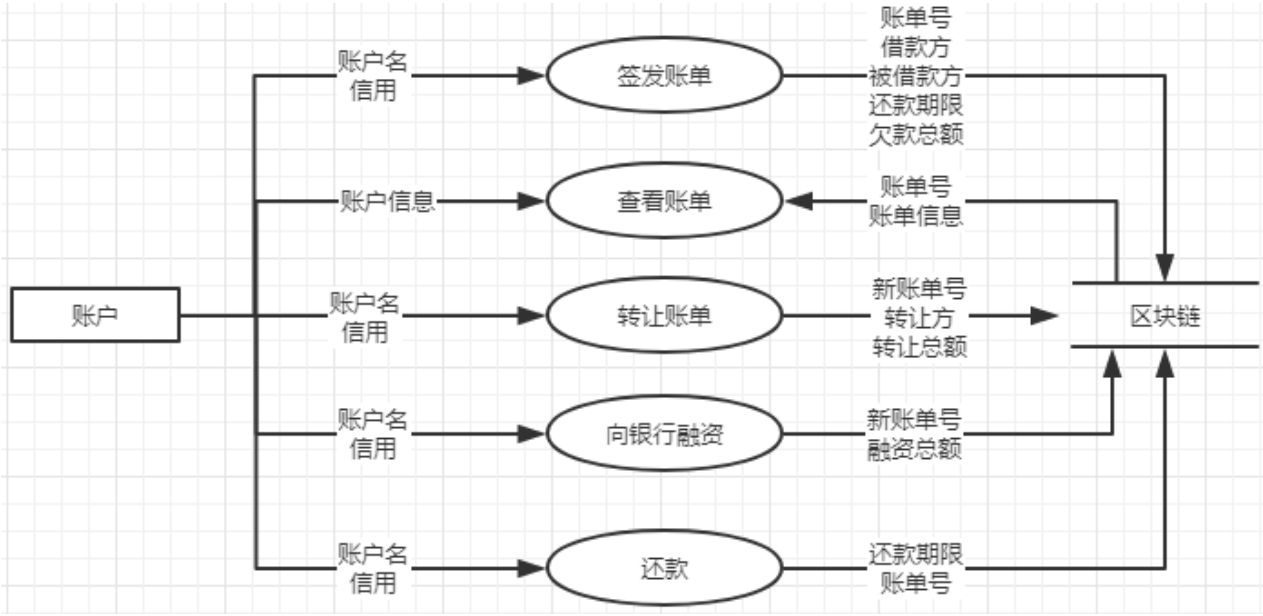
基于区块链、智能合约等，实现基于区块链的供应链金融平台。基于已有的开源区块链系统 FISCO-BCOS，以联盟链为主，开发基于区块链 或区块链智能合约的供应链金融平台，实现供应链应收账款资产的溯源、流转。

二、 方案设计

存储设计

存储使用 AMDB，使用了 AMDB 专用的智能合约 Table.sol 接口。

数据流图



核心功能介绍

本次大作业实现功能如下：

基本功能有：创建用户、查询用户、查询账单。

以及实现了作业要求的四大功能：签发应收账款、应收账款的转让、应收账款向银行融资、应收账款支付结算。

接下来先介绍后端文件 MyProject.sol 的功能逻辑：

首先在构造函数中创建两个表，如下：

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

function createTable() private {
    TableFactory tf = TableFactory(0x1001);

    // 资产管理表, key : bill, field : from_account  to_account  amount  ddl
    // |      账单号(主键)     |    借出钱的用户     |    欠钱的用户       |      总额          |    还款时间        |
    // |---------------------|-------------------|-------------------|-------------------|-------------------|
    // |        bill         |    from_account   |    to_account     |    amount         |      ddl          |
    // |---------------------|-------------------|-------------------|-------------------|-------------------|
    //
    // 创建表:账单
    tf.createTable("t_bill", "bill", "from_account,to_account,amount,ddl");


    // 资产管理表, key : account, field :  credit(0不受信用, 1受信用)
    // |      用户名(主键)     |   是否受银行信用   |
    // |---------------------|-------------------|
    // |       account       |      credit       |
    // |---------------------|-------------------|
    //
    // 创建表: 用户
    tf.createTable("t_user", "account", "credit");
}
```

其中用户表含有用户名和是否受银行信用（credit）。账单表含有账单号、借出钱的用户、欠钱的用户、总额、还款时间。

接下来使用私有方法将打开表的操作封装起来：

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

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

event 如下：

```
//添加用户：返回值，用户名
event RegisterEvent(int256 ret, string account, int256 credit);
//功能一 签发应收账款：返回值，账单号，借出钱的用户，欠钱的用户，欠款总额，还款时间(多少天后)
event IssueEvent(int256 ret, string bill, string from_account, string to_account,  uint256 amount, uint256 ddl);
//功能二 转让应收账款：返回值，分解的账单号，新的账单号，新的借出钱的用户，转让总额
event TransferEvent(int256 ret, string from_bill, string new_bill, string new_account, uint256 amount);
//功能三 向银行融资：返回值，融资用户，融资总额，还款时间
//event FinancingEvent(int256 ret, string account,  uint256 amount, uint256 ddl);
event FinancingEvent(int256 ret);
//功能四 应收账款支付结算：返回值，账单号，距离借钱日过去的时间（天为单位）
//event PayEvent(int256 ret, string bill, uint256 time);
event PayEvent(int256 count);
```

查询用户表的方法如下：

```
/*
描述 : 根据用户名查询是否受银行信用
参数 :
        account : 用户名

返回值:
        参数一:  成功返回0，账户不存在返回-1
        参数二:  第一个参数为0时有效，是否受信用
*/
function selectAccount(string account) public constant returns(int256, uint256) {
    // 打开表
    Table table = openTableUser();
    // 查询
    Entries entries = table.select(account, table.newCondition());
    uint256 credit = 0;
    if (0 == uint256(entries.size())) {
        return (-1, credit);
    } else {
        Entry entry = entries.get(0);
        return (0, uint256(entry.getInt("credit")));
    }
}
```

描述 : 根据用户名查询是否受银行信用

参数 :

account : 用户名

返回值：

参数一： 成功返回 0, 账户不存在返回-1
参数二： 第一个参数为 0 时有效，是否受信用

查询账单表的方法如下：

```
/*
描述 : 根据账单号查询借出钱的用户、欠钱的用户、总额、还款时间
参数 :
        bill : 账单号

返回值:
        参数一: 成功返回0, 账户不存在返回-1
        参数二: 第一个参数为0时有效，借出钱的用户
        参数三: 第一个参数为0时有效，欠钱的用户
        参数四: 第一个参数为0时有效，总额
        参数五: 第一个参数为0时有效，还款时间

*/
function selectBill(string bill) public constant returns(int256, string, string, uint256, uint256) {
    // 打开表
    Table table = openTableBill();
    // 查询
    Entries entries = table.select(bill, table.newCondition());
    //string  storage from_account = "0";
    //string storage to_account = "0";
    //uint256 amount = 0;
    //uint256 ddl = 0;
    if (0 == uint256(entries.size())) {
        //return (-1, from_account, to_account, amount, ddl);
        return (-1,"0","0",0,0);
    } else {
        Entry entry = entries.get(0);
        //from_account = entry.getString("from_account");
        //to_account = entry.getString("to_account");
        //amount = entry.getInt("amount");
        //ddl = entry.getInt("ddl");
        return (int256(0), entry.getString("from_account"), entry.getString("to_account"), uint256(entry.getInt("amount")), uint256(entry.getInt("ddl")));
    }
}
```

描述 : 根据账单号查询借出钱的用户、欠钱的用户、总额、还款时间
参数 :
bill : 账单号
返回值:
参数一： 成功返回 0, 账户不存在返回-1
参数二： 第一个参数为 0 时有效，借出钱的用户
参数三： 第一个参数为 0 时有效，欠钱的用户
参数四： 第一个参数为 0 时有效，总额
参数五： 第一个参数为 0 时有效，还款时间

用户注册功能如下：

```
/*
描述 : 用户注册
参数 :
        account : 资产账户
        credit  : 是否受银行信用
返回值:
        0  资产注册成功
        -1 资产账户已存在
        -2 其他错误
*/
function registerUser(string account, int256 credit) public returns(int256){
    int256 ret_code = 0;
    int256 ret= 0;
    uint256 temp_credit = 0;
    // 查询账户是否存在
    (ret, temp_credit) = selectAccount(account);
    if(ret != 0) {
        Table table = openTableUser();

        Entry entry = table.newEntry();
        entry.set("account", account);
        entry.set("credit", credit);
        // 插入
        int count = table.insert(account, entry);
        if (count == 1) {
            // 成功
            ret_code = 0;
        } else {
            // 失败? 无权限或者其他错误
            ret_code = -2;
        }
    } else {
        // 账户已存在
        ret_code = -1;
    }

    emit RegisterEvent(ret_code, account, credit);
    return ret_code;
}
```

用户注册的逻辑很简单，先查询账户是否存在，若不存在则新建 entry 然后插入然后返回 0，若存在则直接返回-1。

功能一签发应收账款如下：

```
/*

event IssueEvent(int256 ret, string bill, string from_account, string to_account,  uint256 amount, uint256 ddl);

描述 : 功能一 签发应收账款：返回值，账单号，借出钱的用户，欠钱的用户，欠款总额，还款时间(多少天后)
参数 :
        bill : 账单号
        from_account  : 借出钱的用户
        to_account : 欠钱的用户
        amount : 欠款总额
        ddl : 还款时间(多少天后)
返回值：
        0  签发应收账款成功
        -2 其他错误
*/
function issueBill(string bill, string from_account, string to_account, uint256 amount, uint256 ddl) public returns(int256){
    int256 ret_code = 0;
    int256 ret= 1;

    if(ret != 0) {
        Table table = openTableBill();

        Entry entry = table.newEntry();
        entry.set("bill", bill);
        entry.set("from_account", from_account);
        entry.set("to_account", to_account);
        entry.set("amount", int(amount));
        entry.set("ddl", int(ddl));

        // 插入
        int count = table.insert(bill, entry);
        if (count == 1) {
            // 成功
            ret_code = 0;
        } else {
            // 失败? 无权限或者其他错误
            ret_code = -2;
        }
    }

    emit IssueEvent(ret_code, bill, from_account, to_account, amount, ddl);
    return ret_code;
}
```

描述 : 功能一 签发应收账款：返回值，账单号，借出钱的用户，欠钱的用户，欠款总额，还款时间(多少天后)

参数 :

bill : 账单号
from_account : 借出钱的用户
to_account : 欠钱的用户
amount : 欠款总额
ddl : 还款时间(多少天后)

返回值：

0 签发应收账款成功
-2 其他错误

此方法的内部逻辑很直观：先打开账单表，然后 new 一个 entry，将账单的相关信息 set 进 entry 中，然后插入账单表。

功能二实现应收账款的转让如下：

```
function transfer(string from_bill, string new_bill,string new_account, uint256 amount) public returns(int256) {
    uint256 from_amount = 0;

    Table table = openTableBill();

    Entries entries_temp0 = table.select(from_bill, table.newCondition());
    Entry entry_temp0 = entries_temp0.get(0);
    from_amount = uint256(entry_temp0.getInt("amount"));
nt256(0), entry.getString("from_account"), entry.getString("to_account"), uint256(entry.getInt("amount")), uint256

    Entry entry0 = table.newEntry();
    entry0.set("bill", from_bill);
    entry0.set("from_account", entry_temp0.getString("from_account"));
    entry0.set("to_account", entry_temp0.getString("to_account"));
    entry0.set("amount", int256(from_amount - amount));
    entry0.set("ddl", int256(entry_temp0.getInt("ddl")));

    // 更新被分解的账单
    int count = table.update(from_bill, entry0, table.newCondition());
    if(count != 1) {
        // 失败? 无权限或者其他错误?
        emit TransferEvent(-2, from_bill, new_bill, new_account, amount);
        return -2;
    }

    Entry entry1 = table.newEntry();
    entry1.set("bill", new_bill);
    entry1.set("from_account", new_account);
    entry1.set("to_account", entry_temp0.getString("to_account"));
    entry1.set("amount", int256(amount));
    entry1.set("ddl", int256(entry_temp0.getInt("ddl")));

    // 插入新账单
    int count2 = table.insert(new_bill, entry1);


    emit TransferEvent(0, from_bill, new_bill, new_account, amount);
    return 0;
}
```

描述：功能二 转让应收账款：返回值，分解的账单号，新的账单号，新的借出钱的用户，转让总额

参数：

from_bill : 分解的账单号

new_bill ： 新的账单号

new_account : 新借出钱的用户

amount : 转让总额

返回值：

0 签发应收账款成功

-2 其他错误

此方法的内部逻辑为：先打开账单表，然后用一个 entry_temp 选择将要分解的账单号，拿出将分解的账单的总额，然后 new 一个 entry，将各种相关信息 set 进去。其中更新老账单的 amount 等于将分解的账单的总额减去参数 amount（转让总额）。然后 update 数据库。

之后再 new 一个新账单的 entry，将各种相关信息 set 进去。其中新账单的 amount 等于参数 amount（转让总额）。新账单的借出钱的用户为参数中的 from_account，新账单的欠钱用户等于老账单的欠钱用户。之后插入新账单。

功能三利用应收账款向银行融资如下：

```
/*
功能三 向银行融资：返回值，融资用户，融资总额，还款时间
event FinancingEvent(int256 ret, string account,  uint256 amount, uint256 ddl);

    参数 :
        from_bill : 分解的账单号
        new_bill :  新的账单号
        amount : 转让总额
    返回值：
        0 银行融资成功
        -1 欠款方的账单不受信用，银行不融资
        -2 其他问题，融资失败
*/
function fancing(string from_bill, string new_bill, uint256 amount) public returns(int256) {
    Table table0 = openTableBill();
    Table table1 = openTableUser();
    Entries entries_temp0 = table0.select(from_bill, table0.newCondition());
    Entry entry_temp0 = entries_temp0.get(0);

    Entries entries_temp1 = table1.select(entry_temp0.getString("to_account"), table1.newCondition());
    Entry entry_temp1 = entries_temp1.get(0);

    //1为受信用
    if(entry_temp1.getInt("credit")==int256(0)){
        emit FinancingEvent(-1);
        return -1;
    }

    if(entry_temp1.getInt("credit")==int256(1)){
        transfer(from_bill, new_bill,"bank",amount);
        emit FinancingEvent(0);
        return 0;
    }

    emit FinancingEvent(-2);
    return -2;
}
```

功能三 向银行融资：返回值，融资用户，融资总额，还款时间

参数 :

from_bill : 分解的账单号

new_bill : 新的账单号

amount : 转让总额

返回值：

0 银行融资成功

-1 欠款方的账单不受信用，银行不融资

-2 其他问题，融资失败

内部逻辑为：打开账单表和用户表，用 entry_temp0 选择账单表中的融资用户的账单，来获取该账单的欠款人。然后用 entry_temp1 选择用户表中的“该账单的欠款人”条目中的 credit（是否受银行信用）。如果 credit 为 0，则不受信用，直接返回-1。如果 credit 为 1，则受信用，直接调用功能二 transfer 方法，向银行融资。

（注：这里用功能三调用功能二来实现大作业要求，而不将功能二和三合并，是因为最开始出现了报错 stack too deep，方法中的变量太多导致堆栈炸了，所以分两个方法来写减少参数数量）

功能四应收账款支付结算如下：

```
event PayEvent(int256 ret, string bill, uint256 time);

描述 :  功能四 应收账款支付结算：返回值，账单号，距离借钱日过去的时间（天为单位）

参数 :
        bill : 账单号
        time :  距离借钱日过去的时间
*/

function pay(string bill, uint256 time) public returns(int){
    Table table = openTableBill();

    Condition condition = table.newCondition();
    condition.EQ("bill", bill);
    condition.LE("ddl", int256(time));

    int count = table.remove(bill, condition);

    emit PayEvent(count);
    return count;
}
```

描述 : 功能四 应收账款支付结算：返回值，账单号，距离借钱日过去的时间（天为单位）

参数 :

bill : 账单号

time : 距离借钱日过去的时间

内部逻辑为：打开账单表，然后 new 一个 condition，调用 EQ 等于参数 bill 来选择账单号，调用 LE 小于等于参数 time 来比较 bill 表中的 ddl。如果表中有等于 bill 且 ddl 小于等于 time 的账单，则还钱（删除）。

通过 MyProject.sol 文件生成 MyProject.java 文件，然后在前端文件 AssetClient.java 文件中 import MyProject.java 文件。

接下来先介绍前端文件 AssetClient.java 的功能逻辑：

部署合约到链上

```
public void deployAssetAndRecordAddr() {

    try {
        MyProject asset = MyProject.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());
    }
}
```

查询用户信息，对应后端文件 MyProject.sol 的 selectAccount 方法。

```
public void getCredit(String account) {
    try {
        String contractAddress = loadAssetAddr();

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

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

查询账单信息，对应后端文件 MyProject.sol 的 selectBill 方法。

```
public void getBill(String bill) {
    try {
        String contractAddress = loadAssetAddr();

        MyProject asset = MyProject.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        Tuple5<BigInteger, String, String, BigInteger, BigInteger> result = asset.selectBill(bill).send();
        if (result.getValue1().compareTo(new BigInteger("0")) == 0) {
            System.out.printf(" bill: %s\n from_account: %s\n to_account: %s\n amount: %s\n ddl: %s\n", bill, result.getValue2(), result.getValue3(), result.getValue4(), result.getValue5());
        } else {
            System.out.printf(" %s bill is not exist \n", bill);
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        // e.printStackTrace();
        logger.error(" getBill exception, error message is {}", e.getMessage());

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

注册用户，对应后端文件 MyProject.sol 的 registerUser 方法。

```java
public void registerAccount(String account, BigInteger credit) {
    try {
        String contractAddress = loadAssetAddr();

        MyProject asset = MyProject.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        TransactionReceipt receipt = asset.registerUser(account, credit).send();
        List<RegisterEventEventResponse> response = asset.getRegisterEventEvents(receipt);
        if (!response.isEmpty()) {
            if (response.get(0).ret.compareTo(new BigInteger("0")) == 0) {
                System.out.printf(" registerAccount success!\n account: %s\n credit: %s \n", account, credit);
            } else {
                System.out.printf(" registerAccount 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(" registerAccount exception, error message is {}", e.getMessage());
        System.out.printf(" registerAccount, error message is %s\n", e.getMessage());
    }
}
```

创建账单，对应后端文件 MyProject.sol 的 issueBill 方法。

```java
public void createBill(String bill, String from_account, String to_account, BigInteger amount, BigInteger ddl) {
    try {
        String contractAddress = loadAssetAddr();

        MyProject asset = MyProject.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        TransactionReceipt receipt = asset.issueBill(bill, from_account, to_account, amount, ddl).send();
        List<IssueEventEventResponse> response = asset.getIssueEventEvents(receipt);
        if (!response.isEmpty()) {
            if (response.get(0).ret.compareTo(new BigInteger("0")) == 0) {
                System.out.printf(" createBill success!\n bill: %s\n from_account: %s\n to_account: %s\n amount: %s\n ddl: %s\n", bill, from_account, to_account, amount, ddl);
            } else {
                System.out.printf(" createBill 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(" createBill exception, error message is {}", e.getMessage());
        System.out.printf(" createBill, error message is %s\n", e.getMessage());
    }
}
```

转让账单，对应后端文件 MyProject.sol 的 transfer 方法。

```java
public void transferBill(String from_bill, String new_bill, String new_account, BigInteger amount) {
    try {
        String contractAddress = loadAssetAddr();
        MyProject asset = MyProject.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        TransactionReceipt receipt = asset.transfer(from_bill, new_bill, new_account, amount).send();
        List<TransferEventEventResponse> response = asset.getTransferEventEvents(receipt);
        if (!response.isEmpty()) {
            if (response.get(0).ret.compareTo(new BigInteger("0")) == 0) {
                System.out.printf(" transferBill success!\n from_bill: %s\n new_bill: %s\n new_account: %s\n amount: %s\n", from_bill, new_bill, new_account, amount);
            } else {
                System.out.printf(" transferBill 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(" transferBill exception, error message is {}", e.getMessage());
        System.out.printf(" transferBill failed, error message is %s\n", e.getMessage());
    }
}
```

向银行融资，对应后端文件 MyProject.sol 的 fancing 方法。

```
public void financing(String from_bill, String new_bill, BigInteger amount) {
    try {
        String contractAddress = loadAssetAddr();
        MyProject asset = MyProject.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        TransactionReceipt receipt = asset.fancing(from_bill, new_bill, amount).send();
        List<FinancingEventEventResponse> response = asset.getFinancingEventEvents(receipt);
        if (!response.isEmpty()) {
            if (response.get(0).ret.compareTo(new BigInteger("0")) == 0) {
                System.out.printf(" financing success!\n from_bill: %s\n new_bill: %s\n new_account: bank\n amount: %s\n", from_bill, new_bill, amount);
            } else if(response.get(0).ret.compareTo(new BigInteger("-1")) == 0){
                System.out.printf(" financing failed, because from_account is not credit\n");
            } else {
                System.out.printf(" financing failed, because amount is bigger than from_bill");
            }
        } else {
            System.out.println(" event log not found, maybe transaction not exec. ");
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        // e.printStackTrace();

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

还款，对应后端文件 MyProject.sol 的 pay 方法。

```
public void payBill(String bill, BigInteger time) {
    try {
        String contractAddress = loadAssetAddr();
        MyProject asset = MyProject.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
        TransactionReceipt receipt = asset.pay(bill, time).send();
        List<PayEventEventResponse> response = asset.getPayEventEvents(receipt);
        if (!response.isEmpty()) {
            if (response.get(0).count.compareTo(new BigInteger("1")) == 0) {
                System.out.printf(" payBill success!\n");
            } else {
                System.out.printf(" payBill failed, the time is not up to the deadline.\n");
            }
        } else {
            System.out.println(" event log not found, maybe transaction not exec. ");
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        // e.printStackTrace();

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

Usage 方法，给出如何使用该程序的模板。

```
public static void Usage() {
    System.out.println(" Usage:");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient deploy");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient getCredit account");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient getBill bill");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient registerAccount account credit");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient createBill bill from_account to_account amount ddl");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient transferBill from_bill new_bill new_account amount");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient financing from_bill new_bill amount");
    System.out.println("\t java -cp conf/:lib/*:apps/* org.fisco.bcos.asset.client.AssetClient payBill bill time");
    System.exit(0);
}
```

最后在 main 函数中通过 switch 来调用相应方法。

```
public static void main(String[] args) throws Exception {

    if (args.length < 1) {
        Usage();
    }

    AssetClient client = new AssetClient();
    client.initialize();

    switch (args[0]) {
    case "deploy":
        client.deployAssetAndRecordAddr();
        break;
    case "getCredit":
        if (args.length < 2) {
            Usage();
        }
        client.getCredit(args[1]);
        break;
    case "getBill":
        if (args.length < 2) {
            Usage();
        }
        client.getBill(args[1]);
        break;
    case "registerAccount":
        if (args.length < 3) {
            Usage();
        }
        client.registerAccount(args[1], new BigInteger(args[2]));
        break;
    case "createBill":
        if (args.length < 6) {
            Usage();
        }
        client.createBill(args[1], args[2], args[3], new BigInteger(args[4]), new BigInteger(args[5]));
        break;
    case "transferBill":
        if (args.length < 5) {
            Usage();
        }
        client.transferBill(args[1], args[2], args[3], new BigInteger(args[4]));
        break;
    case "financing":
        if (args.length < 4) {
            Usage();
        }
        client.financing(args[1], args[2], new BigInteger(args[3]));
        break;
    case "payBill":
        if (args.length < 3) {
            Usage();
        }
        client.payBill(args[1], new BigInteger(args[2]));
        break;
    default: {
        Usage();
    }
    }

    System.exit(0);
```

三、 功能测试

首先进入 fisco/nodes/127.0.0.1 文件夹启动链端的节点

```
fisco-bcos@fiscobcos-VirtualBox:~/fisco/nodes/127.0.0.1$ ./start_all.sh
try to start node0
try to start node1
try to start node2
try to start node3
 node2 start successfully
 node1 start successfully
 node3 start successfully
 node0 start successfully
```

然后进入 asset-app 文件夹调用 build 指令

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app$ ./gradlew build

BUILD SUCCESSFUL in 1s
4 actionable tasks: 4 up-to-date
```

进入 dist 文件夹然后部署合约

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app$ cd dist
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh deploy
 deploy Asset success, contract address is 0x80f1822ca64718cfdfc073a737984f1f273fee4d
```

创建银行 bank 账户，创建成功。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh registerAccount bank 1
 registerAccount success!
 account: bank
 credit: 1
```

尝试重复创建 bank 账户，因为账户已存在，提示创建失败。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh registerAccount bank 1
 registerAccount failed, ret code is -1
```

创建账户 aUser1、aUser2、aUser3，其中设置 aUser1 为受信用账户、aUser2 和 aUser3 为不受信用账户。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh registerAccount aUser1 1
 registerAccount success!
 account: aUser1
 credit: 1
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh registerAccount aUser2 0
 registerAccount success!
 account: aUser2
 credit: 0
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh registerAccount aUser3 0
 registerAccount success!
 account: aUser3
 credit: 0
```

查询用户 aUser1、aUser2 的信用，查询成功。尝试查询不存在的用户 missTestUser，查询失败，提示用户不存在。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getCredit aUser1
 account aUser1 has credit 1
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getCredit aUser2
 account aUser2 has credit 0
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getCredit missTestUser
 missTestUser account is not exist
```

创建账单 aBill1，aUser2 借给 aUser3 1000 元，还钱时间为 300 天。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh createBill aBill1 aUser2 aUser3 1000 300
 createBill success!
 bill: aBill1
 from_account: aUser2
 to_account: aUser3
 amount: 1000
 ddl: 300
```

查询账单 aBill1，结果正确。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getBill aBill1
 bill: aBill1
 from_account: aUser2
 to_account: aUser3
 amount: 1000
 ddl: 300
```

尝试查询不存在的账单 missTestBill，查询失败，提示账单不存在。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getBill missTestBill
 missTestBill bill is not exist
```

使用 aBill1 向银行融资 100 元，保存到 aBill2 中：（由于 aBill1 的欠款方 aUser3 不受银行信用，所以融资失败）

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh financing aBill1 aBill2 100
 financing failed, because from_account is not credit
```

创建账单：aUser2 借给 aUser1 500 元，还钱 dll 为 100 天。账单号为 aBill2。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh createBill aBill2 aUser2 aUser1 500 100
 createBill success!
 bill: aBill2
 from_account: aUser2
 to_account: aUser1
 amount: 500
 ddl: 100
```

使用 aBill2 向银行融资 99 元，保存到 aBill3 中：（由于 aBill2 的欠款方 aUser1 受银行信用，所以会融资成功）

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh financing aBill2 aBill3 99
 financing success!
 from_bill: aBill2
 new_bill: aBill3
 new_account: bank
 amount: 99
```

查看 aBill2 和 aBill3：aBill2 还有 500 - 99 = 401 元，正确。aBill3 中 aUser1 欠银行 99 元，还款 ddl100 天，正确。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getBill aBill2
 bill: aBill2
 from_account: aUser2
 to_account: aUser1
 amount: 401
 ddl: 100
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getBill aBill3
 bill: aBill3
 from_account: bank
 to_account: aUser1
 amount: 99
 ddl: 100
```

测试 transferBill 功能，从 aBill1 分裂出 300 元，成为 aBill4（从 aUser3 欠 aUser2 1000 元，变成 aUser3 欠 aUser1 300 元，和 aUser3 欠 aUser2 700 元），调用成功。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh transferBill aBill1 aBill4 aUser1 300
 transferBill success!
 from_bill: aBill1
 new_bill: aBill4
 new_account: aUser1
 amount: 300
```

查询 aBill1 和 aBill4，结果正确。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getBill aBill1
 bill: aBill1
 from_account: aUser2
 to_account: aUser3
 amount: 700
 ddl: 300
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getBill aBill4
 bill: aBill4
 from_account: aUser1
 to_account: aUser3
 amount: 300
 ddl: 300
```

测试还款功能，付款账单写：aBill1，时间写：过了 301 天。显示还款成功。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh payBill aBill1 301
 payBill success!
```

查询已经还款的账单 aBill1，显示账单不存在。逻辑正确。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh getBill aBill1
 aBill1 bill is not exist
```

四、 界面展示

前端采用命令行界面。

```
fisco-bcos@fiscobcos-VirtualBox:~/asset-app/dist$ bash asset_run.sh
 Usage :
   bash asset_run.sh deploy
   bash asset_run.sh getCredit account
   bash asset_run.sh getBill bill
   bash asset_run.sh registerAccount account credit
   bash asset_run.sh createBill bill from_account to_account amount ddl
   bash asset_run.sh transferBill from_bill new_bill new_account amount
   bash asset_run.sh financing from_bill new_bill amount
   bash asset_run.sh payBill bill time
```

界面展示详见功能测试模块。

五、 心得体会

本学期通过做区块链大作业，极大程度得巩固了课堂所学知识。只听课不写代码，感觉学的是浮在表面上的，很没有底，而这三个阶段的大作业让我们亲自配置链端、写后端和前端，对区块链有了更深的理解和自己的体会。所以我认为这种学习+实践的模式是很棒的。
学无止境，希望在区块链的课程结束以后，能有机会继续深入学习这个领域的知识!