区块链大作业实验报告

基于区块链的供应链金融平台

一 项目背景

传统供应链金融：

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。在接下里的几个月里，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。

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。

实现功能：

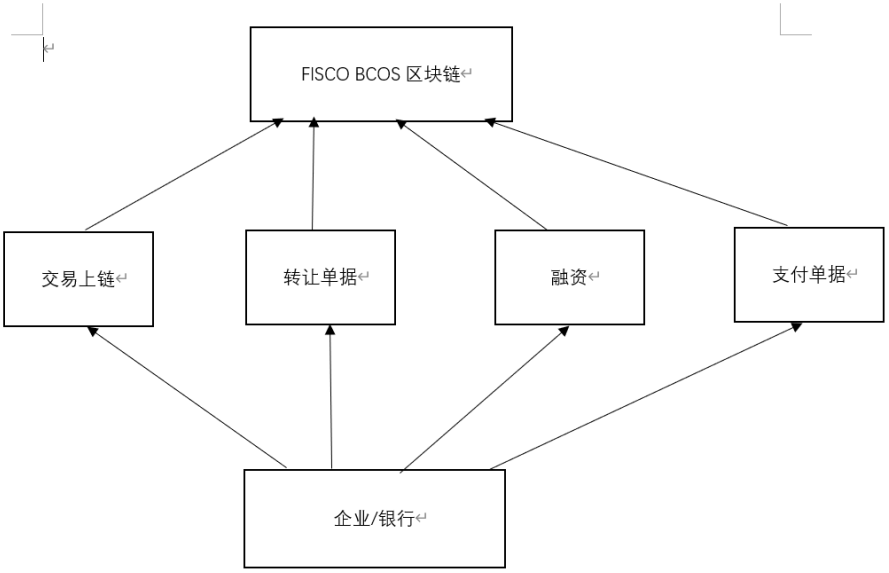
- 功能一：实现采购商品—签发应收账款 交易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。
- 功能二：实现应收账款的转让上链，轮胎公司从轮毂公司购买一笔轮毂，便将于车企的应收账款单据部分转让给轮毂公司。轮毂公司可以利用这个新的单据去融资或者要求车企到期时归还钱款。
- 功能三：利用应收账款向银行融资上链，供应链上所有可以利用应收账款单据向银行申请融资。
- 功能四：应收账款支付结算上链，应收账款单据到期时核心企业向下游企业支付相应的欠款。

二 实验环境

Ubuntu 18.4

FISCO BCOS

三 方案设计



四 实验过程

之前的实验过程中已经完成了设计相关智能合约的部分，在这基础上完成一个基于FISCO BCOS区块链的业务应用场景开发，从业务场景分析，到合约的设计实现，合约编译以及如何部署到区块链，最后实现一个应用模块，通过 FISCO BCOS 提供的Java SDK实现对区块链上合约的调用访问。

4.1 链端

链端即结合业务需求，设计对应的智能合约，确定合约需要储存的数据，在此基础上确定智能合约对外提供的接口，最后给出各个接口的具体实现。FISCO BCOS提供合约CRUD接口开发模式，可以通过合约创建表，并对创建的表进行增删改查操作。针对该项目设计了两个表，一个存储资产信息，一个存储交易记录。

```
    function createTable() private {
        TableFactory tf = TableFactory(0x1001);
        //创建资产管理表
        tf.createTable("assert_manage", "account", "credit_limit");
        //创建交易记录表
        tf.createTable("transaction_record", "id", "acc1, acc2, money, status");
    }
```

进一步设计接口，为了实现需求的四个功能，设计了五个接口，功能分别为资产注册，添加交易记录，欠条拆分，信用额度转移，支付欠条。接口的实现在任务二的实验报告中有详细介绍，在此不过多赘述。

```
    //资产注册
    function register(string account, int256 credit_limit) public returns(int256)
    //添加交易记录
    function addTransaction(string id, string acc1, string acc2, int256 money)
    //欠条拆分
    function splitTransaction(string old_id, string new_id, string acc, int256
money) public returns(int256)
    //信用额度转移
    function transfer(string from, string to, int256 amount) public
```

```
returns(int256)
    //更新交易记录(支付欠条)
    function updateTransaction(string id, int256 money) public returns(int256,
string[])
```

编译智能合约：智能合约需要编译成ABI和BIN文件才能部署至区块链网络上。有了这两个文件即可凭借Java SDK进行合约部署和调用。控制台提供的编译工具不仅可以编译出ABI和BIN文件，还可以自动生成一个与编译的智能合约同名的合约Java类。这个Java类是根据ABI生成的，帮助用户解析好了参数，提供同名的方法。当应用需要部署和调用合约时，可以调用该合约类的对应方法，传入指定参数即可。使用这个合约Java类来开发应用，可以极大简化用户的代码。

命令如下所示，运行成功之后，将会在console/contracts/sdk目录生成java、abi和bin目录。

```
# 创建工作目录~/fisco
mkdir -p ~/fisco
# 下载控制台
cd ~/fisco && curl -#LO https://github.com/FISCO-
BCOS/console/releases/download/v2.7.1/download_console.sh && bash
download_console.sh

# 切换到fisco/console/目录
cd ~/fisco/console/

# 编译合约，后面指定一个Java的包名参数，可以根据实际项目路径指定包名
./sol2java.sh org.fisco.bcos.asset.contract
```

上述编译智能合约后，java目录下生成了org/fisco/bcos/asset/contract/包路径目录，该目录下包含Account.java和Table.java两个文件，其中Account.java是Java应用调用Account.sol合约需要的文件。

Account.java的主要接口：

```
public TransactionReceipt transfer(String from_account, String to_account,
BigInteger amount);

public TransactionReceipt splitTransaction(String old_id, String new_id, String
acc, BigInteger money);

public TransactionReceipt removeTransaction(String id);

public TransactionReceipt addTransaction(String id, String acc1, String acc2,
BigInteger money);

public TransactionReceipt register(String account, BigInteger asset_value);

public TransactionReceipt updateTransaction(String id, BigInteger money);
```

4.2 后端

安装JDK以及集成开发环境

在官网上下载JDK14并安装，然后修改环境变量，命令如下所示：

```
$ java -version
# 确认您的java路径
$ ls Library/Java/JavaVirtualMachines
# 返回
# jdk-14.0.2.jdk

# 如果使用的是bash
$ vim .bash_profile
# 在文件中加入JAVA_HOME的路径
# export JAVA_HOME = Library/Java/JavaVirtualMachines/jdk-14.0.2.jdk/Contents/Home
$ source .bash_profile

# 如果使用的是zash
$ vim .zashrc
# 在文件中加入JAVA_HOME的路径
# export JAVA_HOME = Library/Java/JavaVirtualMachines/jdk-14.0.2.jdk/Contents/Home
$ source .zashrc

# 确认您的java版本
$ java -version
# 返回
# java version "14.0.2" 2020-07-14
# Java(TM) SE Runtime Environment (build 14.0.2+12-46)
# Java HotSpot(TM) 64-Bit Server VM (build 14.0.2+12-46, mixed mode, sharing)
```

进入IntelliJ IDE官网，下载并安装社区版IntelliJ IDE。

创建 JAVA 工程

在IntelliJ IDE中创建一个gradle项目，勾选Gradle和Java，并输入工程名assert-app。

引入FISCO BCOS Java SDK

在build.gradle文件中的dependencies下加入对FISCO BCOS Java SDK的引用。

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


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

配置SDK证书

修改build.gradle文件，引入Spring框架。

```
def spring_version = "4.3.27.RELEASE"
List spring = [
        "org.springframework:spring-core:$spring_version",
        "org.springframework:spring-beans:$spring_version",
        "org.springframework:spring-context:$spring_version",
        "org.springframework:spring-tx:$spring_version",
]

// In this section you declare the dependencies for your production and test code
dependencies {
    compile logger
    runtime logger
    compile ("org.fisco-bcos.java-sdk:fisco-bcos-java-sdk:2.8.0-SNAPSHOT")
    compile spring
}
```

在account-app/test/resources目录下创建配置文件applicationContext.xml，写入配置内容。各配置项的内容可参考Java SDK 配置说明。文件内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd">
    <bean id="defaultConfigProperty"
class="org.fisco.bcos.sdk.config.model.ConfigProperty">
        <property name="cryptoMaterial">
            <map>
            <entry key="certPath" value="conf" />
            <!-- SSL certificate configuration -->
            <!-- entry key="caCert" value="conf/ca.crt" /-->
            <!-- entry key="sslCert" value="conf/sdk.crt" /-->
            <!-- entry key="sslKey" value="conf/sdk.key" /-->
            <!-- GM SSL certificate configuration -->
            <!-- entry key="caCert" value="conf/gm/gmca.crt" /-->
            <!-- entry key="sslCert" value="conf/gm/gmsdk.crt" /-->
            <!-- entry key="sslKey" value="conf/gm/gmsdk.key" /-->
            <!--entry key="enSslCert" value="conf/gm/gmensdk.crt" /-->
            <!--entry key="enSslKey" value="conf/gm/gmensdk.key" /-->
            </map>
        </property>
        <property name="network">
            <map>
                <entry key="peers">
```

```
                    <list>
                        <value>127.0.0.1:20200</value>
                        <value>127.0.0.1:20201</value>
                    </list>
                </entry>
            </map>
        </property>
        <!--
        <property name="amop">
            <list>
                <bean id="amopTopic1"
class="org.fisco.bcos.sdk.config.model.AmopTopic">
                    <property name="topicName" value="PrivateTopic1" />
                    <property name="password" value="" />
                    <property name="privateKey" value="" />
                    <property name="publicKeys">
                        <list>
                            <value>conf/amop/consumer_public_key_1.pem</value>
                        </list>
                    </property>
                </bean>
            </list>
        </property>
        -->
        <property name="account">
            <map>
                <entry key="keyStoreDir" value="account" />
                <entry key="accountAddress" value="" />
                <entry key="accountFileFormat" value="pem" />
                <entry key="password" value="" />
                <entry key="accountFilePath" value="" />
            </map>
        </property>
        <property name="threadPool">
            <map>
            <entry key="channelProcessorThreadSize" value="16" />
            <entry key="receiptProcessorThreadSize" value="16" />
            <entry key="maxBlockingQueueSize" value="102400" />
            </map>
        </property>
    </bean>

    <bean id="defaultConfigOption" class="org.fisco.bcos.sdk.config.ConfigOption">
        <constructor-arg name="configProperty">
                <ref bean="defaultConfigProperty"/>
        </constructor-arg>
    </bean>

    <bean id="bcosSDK" class="org.fisco.bcos.sdk.BcosSDK">
        <constructor-arg name="configOption">
            <ref bean="defaultConfigOption"/>
        </constructor-arg>
    </bean>
</beans>
```

把SDK用于连接节点的证书放到指定的conf目录下，命令如下所示：

```
# 假设我们将account-app放在~/fisco目录下 进入~/fisco目录
$ cd ~/fisco
# 创建放置证书的文件夹
$ mkdir -p account-app/src/test/resources/conf
# 拷贝节点证书到项目的资源目录
$ cp -r nodes/127.0.0.1/sdk/* account-app/src/test/resources/conf
# 若在IDE直接运行，拷贝证书到resources路径
$ mkdir -p account-app/src/main/resources/conf
$ cp -r nodes/127.0.0.1/sdk/* account-app/src/main/resources/conf
```

业务逻辑开发

```
cd ~/fisco
# 将编译好的合约Java类引入项目中。
cp console/contracts/sdk/java/org/fisco/bcos/asset/contract/Account.java account-
app/src/main/java/org/fisco/bcos/asset/contract/Account.java
```

在相应路径下创建 AccountClient.java类，通过调用Account.java实现对合约的部署与调用

参考官网提供的 AssetClient.java 文件，完善本次实验所需的接口，具体接口如下所示：

```
public void queryAssetValue(String assetAccount);
public void queryTransaction(String id);
public void registerAccount(String assetAccount, BigInteger amount);
public void transferAccount(String fromAccount, String toAccount, BigInteger
amount);
public void addTransaction(String id, String acc1, String acc2, BigInteger money);
public void updateTransaction(String id, BigInteger money);
public void splitTransaction(String old_id, String new_id, String acc, BigInteger
money);
public void removeTransaction(String id);
```

各接口都对应智能合约中的各个功能函数，以registerAccount的实现为例介绍接口实现的具体代码：

首先调用load函数加载合约地址，创建Account实例，然后调用合约的register功能，response存储合约事件发生之后的返回列表，如果返回列表不为空（即有效），判断返回列表中的返回值是否有效，有效即打印出注册的资产用户信息，否则打印提示信息。其他接口的实现与之类似。

```
public void registerAccount(String assetAccount, BigInteger amount) {
    try {
      String contractAddress = loadAccountAddr();
```

```
      Account account = Account.load(contractAddress, client, cryptoKeyPair);
      TransactionReceipt receipt = account.register(assetAccount, amount);
      List<Account.RegisterEventEventResponse> response =
account.getRegisterEventEvents(receipt);
      if (!response.isEmpty()) {
        if (response.get(0).ret.compareTo(new BigInteger("0")) == 0) {
          System.out.printf(
              " register Account success => Account: %s, value: %s \n",
assetAccount, amount);
        } else {
          System.out.printf(
              " register 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(" registerAccount exception, error message is {}",
e.getMessage());
      System.out.printf(" registerAccount failed, error message is %s\n",
e.getMessage());
    }
  }
```

之后在tools目录下创建account_run.sh脚本用来调用AccountClient.java。

```
#!/bin/bash

function usage()
{
    echo " Usage : "
    echo "   bash account_run.sh deploy"
    echo "   bash account_run.sh select account "
    echo "   bash account_run.sh select_transaction  id "
    echo "   bash account_run.sh register account asset_value "
    echo "   bash account_run.sh addTransaction id acc1 acc2 money "
    echo "   bash account_run.sh updateTransaction id money "
    echo "   bash account_run.sh splitTransaction old_id new_id acc money "
    echo "   bash account_run.sh removeTransaction id"
    echo " "
    :'
    echo " "
    echo "examples : "
    echo "   bash account_run.sh deploy "
    echo "   bash account_run.sh register  acc1  1000000 "
    echo "   bash account_run.sh register  acc2  10000 "
    echo "   bash account_run.sh register  acc3  10000 "
    echo "   bash account_run.sh register  bank  1000000000 "
```

```
    echo "   bash account_run.sh select acc1"
    echo "   bash account_run.sh select acc2"
    echo "   bash account_run.sh addTransaction 0001 acc1 acc2 100"
    echo "   bash account_run.sh select_transaction 0001"
    echo "   bash account_run.sh splitTransaction 0001 0002 acc3 50"
    echo "   bash account_run.sh splitTransaction 0001 0003 bank 10"
    echo "   bash account_run.sh select_transaction 0002"
    echo "   bash account_run.sh select_transaction 0003"
    echo "   bash account_run.sh updateTransaction 0001 40"
    echo "   bash account_run.sh updateTransaction 0002 50"
    echo "   bash account_run.sh updateTransaction 0003 10"
    '
    exit 0
}

    case $1 in
    deploy)
            [ $# -lt 1 ] && { usage; }
            ;;
    select)
            [ $# -lt 2 ] && { usage; }
            ;;
    select_transaction)
            [ $# -lt 2 ] && { usage; }
            ;;
    register)
            [ $# -lt 3 ] && { usage; }
            ;;
    addTransaction)
            [ $# -lt 5 ] && { usage; }
            ;;
    splitTransaction)
            [ $# -lt 5 ] && { usage; }
            ;;
    updateTransaction)
            [ $# -lt 3 ] && { usage; }
            ;;
    removeTransaction)
            [ $# -lt 2 ] && { usage; }
            ;;
    *)
        usage
            ;;
    esac
    java -Djdk.tls.namedGroups="secp256k1" -cp 'apps/*:conf/:lib/*'
org.fisco.bcos.account.client.AccountClient $@
```

然后配置log，在src/test/resources目录下创建log4j.properties。

```
### set log levels ###
log4j.rootLogger=DEBUG, file
```

```
### output the log information to the file ###
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
log4j.appender.file.DatePattern='_'yyyyMMddHH'.log'
log4j.appender.file.File=./log/sdk.log
log4j.appender.file.Append=true
log4j.appender.file.filter.traceFilter=org.apache.log4j.varia.LevelRangeFilter
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=[%p] [%-d{yyyy-MM-dd HH:mm:ss}]
%C{1}.%M(%L) | %m%n

###output the log information to the console ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%p] [%-d{yyyy-MM-dd HH:mm:ss}]
%C{1}.%M(%L) | %m%n
```

通过配置gradle中的Jar命令，指定复制和编译任务。并引入日志库，在/src/test/resources目录下，创建一个空的contract.properties文件，用于应用在运行时存放合约地址。

开发过程完成，实验结果测试在后文中叙述。

4.3 前端

由于期末时间紧张以及在之前并未学习过前端开发的有关知识，暂时还没有进行该应用的前端开发，在之后的学习中有机会会继续完善。

五 实验结果

应用组织

参照官网给出的区块链应用开发方法，我们最终开发出的应用account-app的目录结构如下：

```
|-- build.gradle // gradle配置文件
|-- gradle
|   |-- wrapper
|       |-- gradle-wrapper.jar // 用于下载Gradle的相关代码实现
|       |-- gradle-wrapper.properties // wrapper所使用的配置信息，比如gradle的版本等
信息
|-- gradlew // Linux或者Unix下用于执行wrapper命令的Shell脚本
|-- gradlew.bat // Windows下用于执行wrapper命令的批处理脚本
|-- src
|   |-- main
|   |   |-- java
|   |   |     |-- org
|   |   |          |-- fisco
|   |   |                |-- bcos
|   |   |                      |-- account
|   |   |                            |-- client // 放置客户端调用类
|   |   |                                   |-- AccountClient.java
|   |   |                            |-- contract // 放置Java合约类
```

```
|   |   |                                       |-- Account.java
|   |   |-- resources
|   |        |-- conf
|   |               |-- ca.crt
|   |               |-- node.crt
|   |               |-- node.key
|   |               |-- sdk.crt
|   |               |-- sdk.key
|   |               |-- sdk.publickey
|   |        |-- 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
|   |        |-- contract //存放solidity约文件
|   |                |-- Account.sol
|   |                |-- Table.sol
|   |-- test
|       |-- resources // 存放代码资源文件
|           |-- conf
|                  |-- ca.crt
|                  |-- node.crt
|                  |-- node.key
|                  |-- sdk.crt
|                  |-- sdk.key
|                  |-- sdk.publickey
|           |-- 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
|           |-- contract //存放solidity约文件
|                  |-- Account.sol
|                  |-- Table.sol
|
|-- tool
    |-- account_run.sh // 项目运行脚本
```

运行结果

- **编译**

```
# 切换到项目目录
$ cd ~/fisco/account-app
# 编译项目
$ ./gradlew build
```

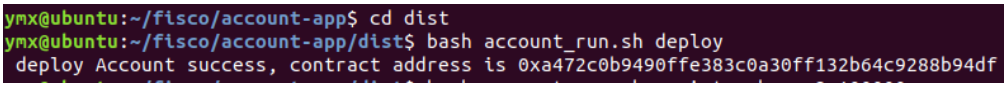
```
ymx@ubuntu:~/fisco/account-app$ ./gradlew build

BUILD SUCCESSFUL in 8s
4 actionable tasks: 4 executed
```

编译成功之后，将在项目根目录下生成`dist`目录。dist目录下有一个`account_run.sh`脚本，简化项目运行。

- **部署Account.sol合约**

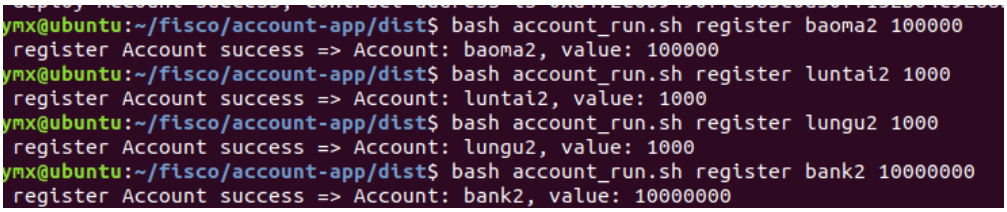
```
# 进入dist目录
$ cd dist
$ bash account_run.sh deploy
deploy Account success, contract address is
0xa472c0b9490ffe383c0a30ff132b64c9288b94df
```



- **测试功能**

  用户登记

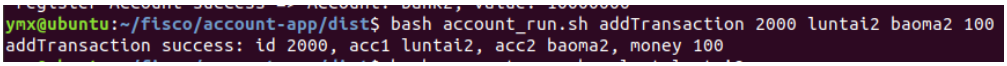
```
$ bash account_run.sh register baoma2 100000
 register Account success => Account: baoma2, value: 100000
$ bash account_run.sh register luntai2 1000
 register Account success => Account: luntai2, value: 1000
$ bash account_run.sh register lungu2 1000
 register Account success => Account: lungu2, value: 1000
$ bash account_run.sh register bank2 10000000
 register Account success => Account: bank2, value: 10000000
```



采购商品-签收应收账款-交易上链

宝马公司向轮胎公司购买100万轮胎：

```
$ bash account_run.sh addTransaction 2000 luntai2 baoma2 100
```



查看二者的信息和交易信息

```
$ bash account_run.sh select luntai2
 Account luntai2, value 1100
$ bash account_run.sh select baoma2
 Account baoma2, value 99900
$ bash account_run.sh select_transaction 2000
 ID 2000, Transaction luntai2 baoma2 100 100
```

```
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select luntai2
 Account luntai2, value 1100
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select baoma2
 Account baoma2, value 99900
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2000
 ID 2000, Transaction luntai2 baoma2 100 100
```

应收账款转让上链

```
$ bash account_run.sh splitTransaction 2000 2001 lungu2 50
```

```
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh splitTransaction 2000 2001 lungu2 50
splitTransaction success: old id 2000, new id 2001, acc lungu2, money 50
```

查看交易信息

```
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2000
 ID 2000, Transaction luntai2 baoma2 100 50
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2001
 ID 2001, Transaction lungu2 baoma2 50 50
```

利用应收账款向银行融资上链

```
$ bash account_run.sh splitTransaction 2000 2002 bank 10
```

```
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh splitTransaction 2000 2002 bank 10
splitTransaction success: old id 2000, new id 2002, acc bank, money 10
```

查看交易信息

```
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2000
 ID 2000, Transaction luntai2 baoma2 100 40
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2002
 ID 2002, Transaction bank baoma2 10 10
```

应收账款支付结算上链

```
$ bash account_run.sh updateTransaction 2000 40
$ bash account_run.sh updateTransaction 2001 50
$ bash account_run.sh updateTransaction 2002 10
```

```
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh updateTransaction 2000 40
updateTransaction success: id 2000, money 40
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2000
 ID 2000, Transaction luntai2 baoma2 100 0
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh updateTransaction 2001 50
updateTransaction success: id 2001, money 50
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2001
 ID 2001, Transaction lungu2 baoma2 50 0
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh updateTransaction 2002 10
updateTransaction success: id 2002, money 10
ymx@ubuntu:~/fisco/account-app/dist$ bash account_run.sh select_transaction 2002
 ID 2002, Transaction bank baoma2 10 0
```

六 实验总结

这次实验分成了三个阶段，从配环境到编写智能合约，再到最后构建成一个应用，我认为安排还是非常合理的，不至于一开始就上手造飞机。一开始确实是什么都不知道，官网文档起到了很大的帮助作用。除了第一阶段手把手学着搭环境比较简单，后面两个阶段各有困难吧。编写智能合约用的是solidiity这种新语言，我们参考官网的智能合约例子编写，也没有语言编译工具，只能每次看部署合约时报什么错，中间出了很多bug，多亏组员们共同debug，才终于把代码跑通。第三阶段一开始也是无从下手，尤其是对于我们这些没有学过什么web开发、没学过java语言的计科同学们，连前端和后端的区别都没有分清，然后也是参考着官方文档，一步一步学着来。感谢官方文档，我们仿照官方应用的结构，改了自己的AccountClient.java和account_run.sh，在这个过程中也算是学习了java语言，而且也明白了整个应用的文件之间的调用关系，如何通过在AccountClient.java中利用Account.java的接口，从而实现智能合约的调用，是很大的进步。

比较遗憾的是这次没有实现前端，一是我们确实对前端开发一无所知，二来期末的项目实在太多、复习任务艰巨。总而言之，这次大作业，让我们对区块链的组织结构，尤其是它在供应链金融平台的应用有了更多的认识，受益匪浅。在这里，尤其要对组员们共同的努力表示感谢，如果是一个人做这个大作业，怕是会因为太多困难无法克服而想要放弃。