供应链期末项目实验报告

课程名称	区块链原理与技术
小组成员	郭佳怡(18342020)
	刘心怡(18342060)
	赵嘉仪(18342136)
开始日期	2020/11
结束日期	2021/01

项目简介

本次项目是区块链课程的期末项目，在 fisco bcos 框架下完成了一个基于区块链的供应链金融平台，主要目的是实现了供应链上应收账款资产的溯源和流转。具体实现功能包括公司采购商品时的交易上链、不同公司之间进行融资时的转让上链、利用应收款凭据向银行要求融资、以及应收账款结算时上链。总体而言，利用区块链去中心化的特点保证整个供应链条不可被篡改，节点也不能恶意抵赖。并且通过这次项目能够真实的将区块链知识与代码结合起来。

区块链初步搭建及环境配置

首次安装配置 fisco bcos 以及联盟链搭建

首先按照官方文档进行相关环境的配置，因为手册上的内容很详细，所以整个配置环境的过程没有什么难点。

首先搭建一个单群组的4节点联盟链，并且启动所有节点。可以看到成功启动。

```
gxpy@ubuntu:~/fisco$ bash build_chain.sh -l 127.0.0.1:4 -p 30300,20200,8545
[INFO] Downloading fisco-bcos binary from https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.6.0/fisco-bcos.tar.gz ...
############################################################################## 100.0%######
curl: (28) Operation too slow. Less than 102400 bytes/sec transferred the last 20 seconds
[INFO] Download speed is too low, try https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/FISCO-BCOS/FISCO-BCOS/releases/v2.6.0/fisco-bcos.tar.gz
############################################################################## 100.0%
============================================================
Generating CA key...
============================================================
Generating keys and certificates ...
Processing IP=127.0.0.1 Total=4 Agency=agency Groups=1
============================================================
Generating configuration files ...
Processing IP=127.0.0.1 Total=4 Agency=agency Groups=1
============================================================
[INFO] Start Port      : 30300 20200 8545
[INFO] Server IP       : 127.0.0.1:4
[INFO] Output Dir      : /home/gxpy/fisco/nodes
[INFO] CA Path         : /home/gxpy/fisco/nodes/cert/
============================================================
[INFO] Execute the download_console.sh script in directory named by IP to get FISCO-BCOS console.
e.g.  bash /home/gxpy/fisco/nodes/127.0.0.1/download_console.sh -f
============================================================
[INFO] All completed. Files in /home/gxpy/fisco/nodes
gxpy@ubuntu:~/fisco$
```

```
gxpy@ubuntu:~/fisco$ bash nodes/127.0.0.1/start_all.sh
try to start node0
try to start node1
try to start node2
try to start node3
 node0 start successfully
 node1 start successfully
 node3 start successfully
 node2 start successfully
gxpy@ubuntu:~/fisco$
```

同时检查日志的输出，可以看到三个节点之间相互连接，并且存在共识：

```
gxpy@ubuntu:~/fisco$ tail -f nodes/127.0.0.1/node0/log/log*  | grep connected
info|2020-11-26 07:44:39.666763|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:44:49.667233|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:44:59.667469|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:45:09.667846|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:45:19.668183|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:45:29.670896|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:45:39.671397|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:45:49.671619|[P2P][Service] heartBeat,connected count=3
info|2020-11-26 07:45:59.671990|[P2P][Service] heartBeat,connected count=3
```

并且按照步骤配置控制台，需要确定相关节点在启动控制台之前处于开启并且监听的状态。

```
gxpy@ubuntu:~$ cd ~/fisco/console && bash start.sh
=====================================================================================================
Welcome to FISCO BCOS console(2.7.0)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________ ______  ______   ______   ______       ________  ______  ______  ______
|        |      \/      \ /      \ /      \     |        \/      \/      \/      \
| $$$$$$$$\$$$$$|  $$$$$$|  $$$$$$|  $$$$$$\    | $$$$$$$|  $$$$$$|  $$$$$$|  $$$$$$\
| $$__       | $$ | $$___\$| $$   \$| $$  | $$    | $$__/ $| $$   \$| $$  | $| $$___\$$
| $$  \      | $$  \$$    \| $$      | $$  | $$    | $$    $| $$      | $$  | $$\$$    \
| $$$$$      | $$  _\$$$$$$| $$   __| $$  | $$    | $$$$$$$| $$   __| $$  | $$_\$$$$$$\
| $$       _| $$_|  \__| $| $$__/  | $$__/ $$    | $$__/ $| $$__/  | $$__/ $|  \__| $$
| $$      |   $$ \\$$    $$\$$    $$\$$    $$    | $$    $$\$$    $$\$$    $$\$$    $$
 \$$       \$$$$$$ \$$$$$$  \$$$$$$  \$$$$$$      \$$$$$$$  \$$$$$$  \$$$$$$  \$$$$$$

=====================================================================================================
[group:1]>
```

同时，掌握了基本的智能合约编写知识，在console/contract/solidity下编写相应的sol文件，并可以通过编译生成Java代码。

```
gxpy@ubuntu:~/fisco/console$ bash sol2java.sh temp
*** Compile solidity KVTableTest.sol***
INFO: Compile for solidity KVTableTest.sol success.
*** Convert solidity to java  for KVTableTest.sol success ***

*** Compile solidity Table.sol***
INFO: Compile for solidity Table.sol success.
*** Convert solidity to java  for Table.sol success ***

*** Compile solidity TableTest.sol***
INFO: Compile for solidity TableTest.sol success.
*** Convert solidity to java  for TableTest.sol success ***

*** Compile solidity ShaTest.sol***
INFO: Compile for solidity ShaTest.sol success.
*** Convert solidity to java  for ShaTest.sol success ***

*** Compile solidity Self_Contract.sol***
INFO: Compile for solidity Self_Contract.sol success.
*** Convert solidity to java  for Self_Contract.sol success ***

*** Compile solidity HelloWorld.sol***
INFO: Compile for solidity HelloWorld.sol success.
*** Convert solidity to java  for HelloWorld.sol success ***
```

到这一步为止，已经做好了实验的准备工作，可以进行实际的业务逻辑开发了。

供应链链端业务逻辑设计

结构体

Company

```
    struct Company{
        uint id;
        address cAddress;
        bytes32 cName;
        bytes32 password;
        bool isBank;
    }
```

(1) id：公司id，便于遍历。

(2) cAddress：公司地址。

(3) cName：公司名称，在函数调用时以名称而不是地址为变量，便于操作。

(4) password：公司密码。

(5) isBank：是否为银行，此变量为true说明这个公司是一个银行。

Invoice

```
    struct Invoice{
        address payerAddress;
        address payeeAddress;
        uint amount;
        uint deadline;
    }
```

(1) payerAddress：支付者的地址。

(2) payeeAddress：待支付者的地址。

(3) amount：待支付数额。

(4) deadline：支付期限。

变量和映射

```
    uint cNum;
    mapping (bytes32 => address) public cAddressName;
    mapping (uint => address) public cAddressInt;
    mapping (address => Company) public companys;
    mapping (address => Invoice[]) public invoices;
```

（1）cNum：初始值为0，表示公司数目。

（2）cAddressName：从公司名字到公司地址的映射。

（3）cAddressInt：从公司id到公司地址的映射。

（4）companys：从公司地址到Company的映射。

（5）invoices：从公司地址到Invoice[]的映射。

事件

```
event Sign(address payerAddress, address payeeAddress, uint amount, uint deadline);
event Transfer(address senderAddress, uint tarInvoiceIndex, address receiverAddress, uint amount);
event Finance(address borrowerAddress, uint tarInvoiceIndex, address bankAddress, uint amount);
event Settle(address payerAddress, address payeeAddress, uint amount);
```

分别对应需要实现的四个功能。发送事件时，会触发参数存储到交易的日志中。

函数

Constructor

```
    constructor()
    public
    {
        cNum = 0;
    }
```

构造函数，将公司数量cNum置为0。

stringToBytes32

```
    function stringToBytes32(string source)
    public
    pure
    returns (bytes32 result)
    {
        bytes memory temp = bytes(source);
        if (temp.length == 0) {
            return 0x0;
        }
        assembly {
            result := mload(add(source, 32))
        }
    }
```

作用是将string类型转换成bytes32类型，因为string类型长度可变，不能作为映射的自变量，而bytes32长度固定，可以作为映射的自变量，所以使用这个函数进行转换。

register

```
    function register(address cAddress, string cName, string password, bool
isBank)
    public
    returns(bool)
    {
        bytes32 cNamebytes = stringToBytes32(cName);
        for(uint i = 0; i < cNum; i++){
            //保证公司没有重名
            require(cNamebytes != companys[cAddressInt[i]].cName, "There
exists a client who has the same username.");
            //保证公司没有重地址
            require(cAddress != companys[cAddressInt[i]].cAddress, "There
exists a client who has the same address.");
        }
        cAddressName[cNamebytes] = cAddress;
        cAddressInt[cNum] = cAddress;
        companys[cAddress] = Company(cNum++, cAddress, cNamebytes,
stringToBytes32(password), isBank);
        return true;
    }
```

作用是注册公司，输入公司地址、公司名称、密码、是否是银行，输出是否成功注册。首先将公司名称转换成bytes32类型，方便接下来映射。然后遍历所有公司，保证公司名称和公司地址没有重复。接着添加公司名称到公司地址的映射、公司id到公司地址的映射、公司地址到Company的映射。最后返回注册成功。

checkPassword

```
    function checkPassword(address cAddress, string password)
    public
    view
    returns(bool)
    {
        return (companys[cAddress].password == stringToBytes32(password));
    }

```

检查用户地址和密码是否正确匹配。输入是用户名和密码，输出是匹配结果。方法是查找用户地址所对应的密码和输入的密码是否相同。

signInvoice

```
    function signInvoice(string payerName, string password, string payeeName, uint amount, uint deadline)
    public
    returns(bool)
    {
        address payerAddress = cAddressName[stringToBytes32(payerName)];
        //保证存在公司
        require(payerAddress != 0, "No such payer.");
        //保证公司地址和密码匹配
        require(checkPassword(payerAddress, password) == true, "Wrong password.");
        address payeeAddress = cAddressName[stringToBytes32(payeeName)];
        //保证存在公司
        require(payeeAddress != 0, "No such payee.");
        invoices[payeeAddress].push(Invoice(payerAddress, payeeAddress, amount, deadline));
        emit Sign(payerAddress, payeeAddress, amount, deadline);
        return true;
    }
```

完成功能一（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）。输入单据的付款公司名称、付款公司密码、收款公司名称，单据数额，还款时间，输出是否成功。首先根据付款公司名称得到付款公司地址，保证存在付款公司，保证付款公司的密码正确匹配。根据收款公司名称得到收款公司地址，保证存在收款公司。新建Invoice，push进收款者的invoice列表里。最后调用Sign事件，返回true。

transferInvoice

```
    function transferInvoice(string senderName, string password, uint tarInvoiceIndex, string receiverName, uint amount)
    public
    returns(bool)
    {
        address senderAddress = cAddressName[stringToBytes32(senderName)];
        //保证存在公司
        require(senderAddress != 0, "No such sender.");
        //保证公司地址和密码匹配
        require(checkPassword(senderAddress, password) == true, "Wrong password.");
        //保证存在单据
```

```
        require(getInvoiceNum(senderName) > tarInvoiceIndex, "No such invoice.");
        address receiverAddress = cAddressName[stringToBytes32(receiverName)];
        //保证存在公司
        require(receiverAddress != 0, "No such receiver.");
        Invoice memory tar = invoices[senderAddress][tarInvoiceIndex];
        //保证单据的金额大于等于转账金额
        require(tar.amount >= amount, "Transfer amount should less than or equal to the invoice amount.");
        invoices[receiverAddress].push(Invoice(tar.payerAddress, receiverAddress, amount, tar.deadline));
        invoices[senderAddress][tarInvoiceIndex].amount -= amount;
        emit Transfer(senderAddress, tarInvoiceIndex, receiverAddress, amount);
        return true;
    }
```

实现功能二（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）。输入转让公司名称、转让公司密码、待转让单据的序号、接收公司名称、转让金额。首先根据转让公司名称得到转让公司地址，保证存在转让公司，保证转让公司的密码正确匹配，保证存在待转让单据。根据接收公司名称得到接收公司地址，保证存在接收公司。保证待转让单据的金额大于转让金额。向接收公司的invoice列表push一个新的invoice，要注意的是，这个invoice的payerAddress应该和待转让单据的payerAddress一致。修改待转让单据的金额。最后调用Transfer事件，返回true。

finance

```
    function finance(string borrowerName, string password, uint tarInvoiceIndex, string bankName, uint amount)
    public
    returns(bool)
    {
        address borrowerAddress = cAddressName[stringToBytes32(borrowerName)];
        //保证存在公司
        require(borrowerAddress != 0, "No such borrower.");
        //保证公司地址和密码匹配
        require(checkPassword(borrowerAddress, password) == true, "Wrong password.");
        address bankAddress = cAddressName[stringToBytes32(bankName)];
        //保证存在公司
        require(bankAddress != 0, "No such bank.");
        //保证贷款对象是银行
        require(companys[bankAddress].isBank == true, "It is not a bank.");
        //保证存在单据
        require(getInvoiceNum(borrowerName) > tarInvoiceIndex, "No such invoice.");
        Invoice memory tar = invoices[borrowerAddress][tarInvoiceIndex];
        //保证单据的金额大于等于贷款金额
        require(tar.amount >= amount, "Finance amount should less than or equal to the invoice amount.");
        invoices[bankAddress].push(Invoice(tar.payerAddress, bankAddress, amount, tar.deadline));
        invoices[borrowerAddress][tarInvoiceIndex].amount -= amount;
        emit Finance(borrowerAddress, tarInvoiceIndex, bankAddress, amount);
```

```
        return true;
    }
```

实现功能三（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）。输入需融资公司名称、需融资公司密码、用于融资单据的序号、银行名称、融资金额。首先根据需融资公司名称得到需融资公司地址，保证存在需融资公司，保证需融资公司的密码正确匹配，保证存在用于融资的单据。根据银行名称得到银行地址，保证存在银行，保证这个公司是银行。保证用于融资的单据的金额大于融资金额。向银行的invoice列表push一个新的invoice，要注意的是，这个invoice的payerAddress应该和用于融资的单据的payerAddress一致。修改用于融资的单据的金额。最后调用Finance事件，返回true。

settle

```
    //没有偿还某一个账单的选项，因为所有衍生账单还款时限一样，所以对于还款这个操作没有区分的必要
    function settle(string payerName, string password, uint amount)
    public
    returns(bool)
    {
        address payerAddress = cAddressName[stringToBytes32(payerName)];
        //保证存在公司
        require(payerAddress != 0, "No such payer.");
        //保证公司地址和密码匹配
        require(checkPassword(payerAddress, password) == true, "Wrong password.");
        //遍历公司
        for(uint i = 0; i < cNum; i++){
            address a = cAddressInt[i];
            //遍历公司的单据
            for(uint j = 0; j < invoices[a].length; j++){
                //还款单据要求payerAddress和还款公司地址一致，amount和deadline大于0
                if(invoices[a][j].payerAddress == payerAddress && invoices[a][j].amount > 0 && invoices[a][j].deadline > 0){
                    if(invoices[a][j].amount >= amount){
                        invoices[a][j].amount -= amount;
                        emit Settle(payerAddress, a, amount);
                    }
                    else{
                        uint temp = invoices[a][j].amount;
                        invoices[a][j].amount = 0;
                        emit Settle(payerAddress, a, temp);
                        settle(payerName, password, amount - temp);
                    }
                }
            }
        }

        return true;
    }
```

实现功能四（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）。因为单据和衍生单据的还款时限是一样的，还款的先后顺序没有关系，所以还款的操作并没有指定还款对象。输入还款公司名称，还款公司密码，还款金额，输出是否成功还款。首先根据还款公司名称得到还款公司地址，保证存在还款公司，保证还款公司的密码正确匹配。根据id遍历companys中每一个公司，再遍历这个公司的所有单据，如果这个单据的payerAddress和付款公司地址相同，且单据的

amount和deadline都大于0，就进行还款。具体的还款方法是，如果单据的amount大于等于还款金额，说明可以直接在单据的amount上减去还款金额，调用Settle事件；如果单据的amount小于还款金额，用temp来暂时记录单据的amount，然后将单据的amount设为0，调用Settle事件，调用settle函数继续还款，还款金额变为原还款金额减去temp。

getInvoiceNum

```
    function getInvoiceNum(string cName)
    public
    view
    returns(uint invoiceNumber)
    {
        address cAddress = cAddressName[stringToBytes32(cName)];
        invoiceNumber = invoices[cAddress].length;
    }
```

作用是得到输入公司名拥有的待还款单据数量。

getInvoice

```
    function getInvoice(string cName, uint index)
    public
    view
    returns(address payerAddress, address payeeAddress, uint amount, uint deadline)
    {
        require(getInvoiceNum(cName) > index, "The invoice doesn't exist.");
        address cAddress = cAddressName[stringToBytes32(cName)];
        payerAddress = invoices[cAddress][index].payerAddress;
        payeeAddress = invoices[cAddress][index].payeeAddress;
        amount = invoices[cAddress][index].amount;
        deadline = invoices[cAddress][index].deadline;
    }
```

输入公司名和序号index，作用是得到该公司的第index个单据（从0开始）。

供应链后端API接口框架

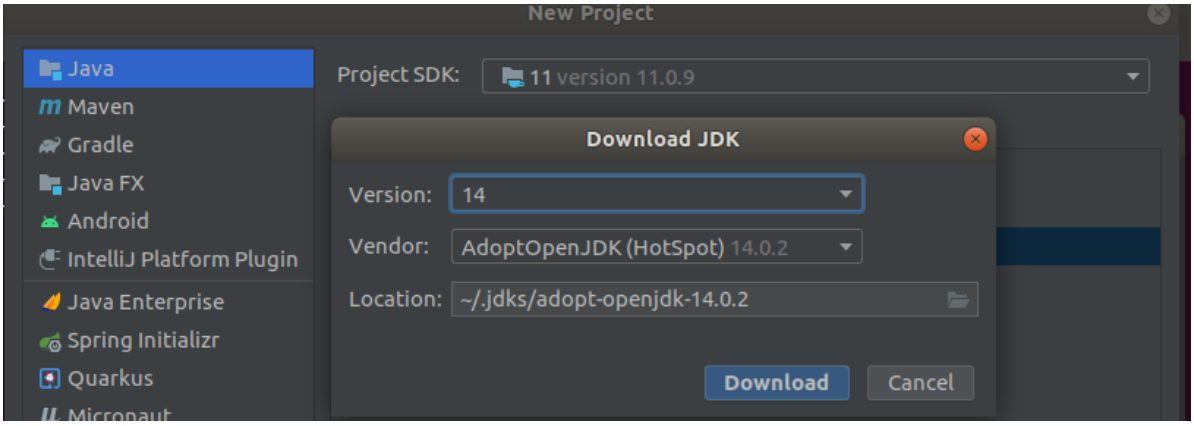
框架简介以及配置过程

本次项目的后端使用spring boot框架，使用的开发环境是Intellij IDEA。实现的API格式为RESTful API（一种符合Rest设计风格的Web API）。

首先在Linux环境下安装Intellij IDEA，下载好压缩包后，执行以下命令（先对压缩包进行解压，再修改权限并执行安装脚本）：

```
idea-IU-203.6682.168/jbr/lib/jexec
idea-IU-203.6682.168/jbr/lib/jspawnhelper
gxpy@ubuntu:~$ cd idea-IU-203.6682.168/
gxpy@ubuntu:~/idea-IU-203.6682.168$ sudo bin/idea.sh
[sudo] password for gxpy:
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.
Jan 22, 2021 8:13:10 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Jan 22, 2021 8:13:10 PM java.util.prefs.FileSystemPreferences$6 run
WARNING: Prefs file removed in background /root/.java/.userPrefs/prefs.xml
```

同时，jdk版本为了与教程中保持一致，又下载了14 version：



而且需要注意的是，后端开发环境虽然需要用到与链段进行交互，但是并不需要在Linux环境下开发运行，只需要将之前链段完成的智能合约用web3j工具进行打包编译，生成对应的Java类文件即可。而这一步的打包编译工作也可以在Windows的环境下进行。

具体开发过程

首先需要使用web3j的打包命令，将之前的SupplyChain.sol文件生成对应的Java文件。

具体命令过程如下。

首先，安装编译工具sol：

```
npm install -g solc
```

之后，用编译命令，根据sol文件产生对应的bin和abi文件：

```
solcjs <sol文件目录>   --optimize  --bin --abi --output-dir <输出目录>
```

最后一步产生对应的java代码：

```
web3j solidity generate <编译的bin文件地址> <编译的abi文件地址> -o <输出目录> -p
<java包名>
```

执行上述命令之后，产生的Java代码大致如下图所示：

```
                Arrays.<Type>asList(new org.web3j.abi.datatypes.Utf8String(cName),
                new org.web3j.abi.datatypes.generated.Uint256(index)),
                Collections.<TypeReference<?>>emptyList());
        return executeRemoteCallTransaction(function);
    }

    public RemoteCall<TransactionReceipt> getInvoiceNum(String cName) {
        final Function function = new Function(
                "getInvoiceNum",
                Arrays.<Type>asList(new org.web3j.abi.datatypes.Utf8String(cName)),
                Collections.<TypeReference<?>>emptyList());
        return executeRemoteCallTransaction(function);
    }

    public RemoteCall<TransactionReceipt> invoices(String param0, BigInteger param1) {
        final Function function = new Function(
                "invoices",
                Arrays.<Type>asList(new org.web3j.abi.datatypes.Address(param0),
                new org.web3j.abi.datatypes.generated.Uint256(param1)),
                Collections.<TypeReference<?>>emptyList());
        return executeRemoteCallTransaction(function);
    }

    public RemoteCall<TransactionReceipt> register(String cAddress, String cName, String password, Boolean isBank) {
        final Function function = new Function(
                "register",
                Arrays.<Type>asList(new org.web3j.abi.datatypes.Address(cAddress),
                new org.web3j.abi.datatypes.Utf8String(cName),
                new org.web3j.abi.datatypes.Utf8String(password),
                new org.web3j.abi.datatypes.Bool(isBank)),
                Collections.<TypeReference<?>>emptyList());
        return executeRemoteCallTransaction(function);
    }
```

具体代码可以在后端的代码文件中查看。

本次供应链项目与其他项目不同之处就在于，后端需要与链段进行交互，恰好利用了智能合约自动产生的Java文件。

而其余部分即定义提供API接口，利用的框架仍然是比较主流的spring boot框架，API接口规范遵循RESTful API定义。

首先修改build.gradle文件，引入spring boot框架：

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

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",
]

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
    compile ("org.fisco-bcos.java-sdk:fisco-bcos-java-sdk:2.8.0-SNAPSHOT")
    compile spring
}
```

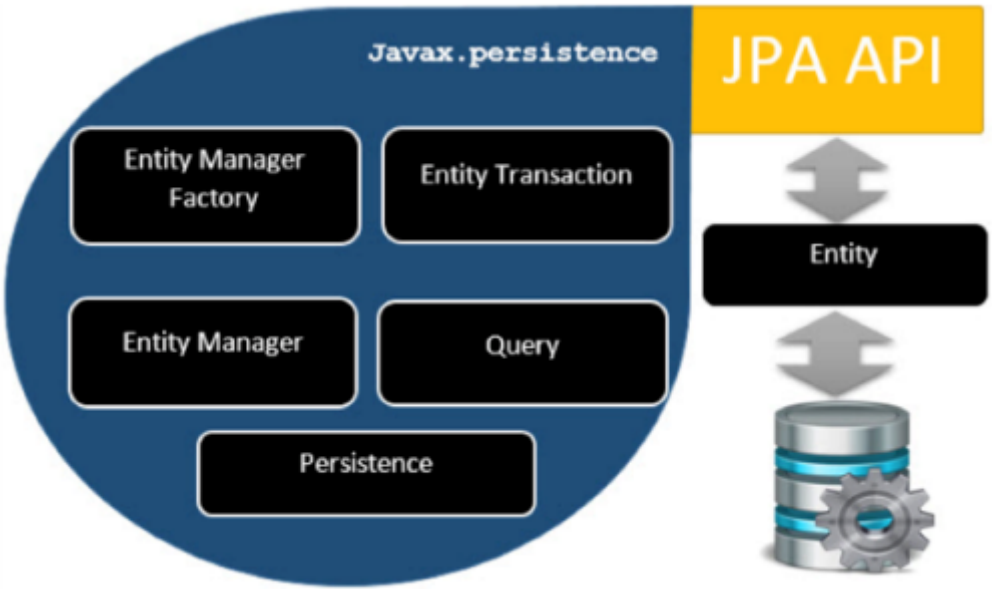
同时，还需要修改配置pom.xml文件。 因为搭链的时候是按照官网提供的教程来的，后续也没有修改节点ip以及对应的通讯端口等，所以直接按照官网提供的sample code进行配置，如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.guides</groupId>
        <artifactId>tut-rest</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>

    <artifactId>rest</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- tag::spring-hateoas[] -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-hateoas</artifactId>
        </dependency>
```

对于 Spring boot 下搭建的 RESTful API，因为网上有比较多的教程和框架代码，所以入门也比较容易。

总的而言，需要一个@Entity的实体类，实体类用于表示对数据的完整建模，比如定义的Company一类，就是用来表示一个注册用户的信息，私有字段即为前期分析设计的字段，包括companyName、companyPassword、companyAddress等。同时，还需要一个@RestRepository的接口，用来扩展JPA，即用来做数据持久化。如果需要的功能不多，其实可以直接基于Repository的接口定义@RestController，定义各种get、put 等请求的具体参数和行为逻辑。但是因为供应链这个实验用到的逻辑更复杂，不仅仅是简单的增删查改，所以又加了一个Service对象，用于具体实现JPA接口，相当于又进行了一层抽象功能。

具体架构简单如图所示：



RESTful API接口文档

状态说明

状态	说明	应用API
success	表明操作成功	所有API
error	表明操作失败	所有API
name_password_notMatch	登陆公司的用户名和密码不匹配	登陆
name_not_exit	登陆的公司用户账号不存在	登陆
company_exit	公司已经存在	注册
not_login	未登录	签订订单、转让订单、进行融资、进行还款
invoice_not_exit	相应的票据不存在	转让、融资、还款
company_not_bank	融资时对应的公司不是银行	融资
money_not_enough	还款时的金额小于应还款的金额	还款
invalid_time_format	还款时间输入不合法	签订订单、转让订单、进行融资
bad_req	错误的请求信息（可能输入的json格式与API接口不符）	所有API

- 注意：所有GET类型默认返回success状态，错误将在http状态码中体现

Company

新公司注册API

```
POST /RegisterNewCompany
```

Request

参数名	类型	描述
companyName	string	公司名称
companyPassword	string	公司账号对应的密码
isBank	Bool	判断公司类型是否是银行
initialBalance	int	公司的初始资金

- 参数使用json形式提交

Example

```
{
    "companyName": "牛牛火锅",
    "companyPassword": "MouMouHotPot",
    "isBank": false,
    "initialBalance" : 2000
}
```

Response

> Status: 201 Created
>
> Location: /RegisterNewCompany

参数名	类型	描述	参数
State	string	状态	success,company_exit,bad_req
Data	string	数据	无

- 参数使用json形式解析

Example

```
{
    "State": "success",
  "Data": ""
}
```

公司登陆

```
POST /LoginCompany
```

Request

参数名	类型	描述
name	string	公司名称
password	string	公司密码

- 参数使用json形式提交

Example

```
{
    "name": "牛牛火锅",
    "password": "MouMouHotPot"
}
```

Response

> Status: 200 OK
>
> Location: /LoginCompany

参数名	类型	描述	参数
State	string	状态	success, name_password_notMatch,name_not_exit,bad_req
Data	string	令牌	暂无

- 参数使用json形式解析

Example

```
{
    "State": "success",
  "Data":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MDg0NTE0OTAsIm5hbWUiOiJzdW5oYW9uYW4iLCJwYXNzd29yZCI6IjEyMzQ1NiJ9.XfEv5awYf7sw6b6wrgiiz691MKGx-sCYKY1FwgaKemQ"
}
```

退出登录

```
POST /CompanyLogout
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: /CompanyLogout

参数名	类型	描述	参数
State	string	状态	success, bad_req
Data	string	数据	暂无

- 参数使用json形式解析

Example

```
{
    "State": "success",
  "Data":""
}
```

获取全部注册公司信息

```
GET /AllRegisteredCompany
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: /AllRegisteredCompany

参数名	类型	描述	状态
State	string	状态	success,not_login,bad_req
Data	array	全部公司信息	暂无

- 参数使用json形式解析

Example

```
{
  "State":"success",
  "Data": {
        "ID": "5fbcb442f5beb22628d4b685",
        "companyName": "牛牛火锅",
        "companyPassword": "MouMouHotPot",
        "isBank": false,
        "initialBalance":2000
    }
}
```

根据ID获取相应的公司信息

```
GET /AllRegisteredCompany/{id}
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: /AllRegisteredCompany/{id}

参数名	类型	描述	状态
State	string	状态	success,not_login,bad_req

> | companyName | string | 公司名称 | 无 | companyPassword | string | 公司账号对应的密码 | 无 | isBank | Bool | 判断公司类型是否是银行 | 无 | initialBalance | int | 公司的初始资金 | 无

- 参数使用json形式解析

Example

```
{
  "State":"success",
  "companyName": "牛牛火锅",
  "companyPassword": "MouMouHotPot",
  "isBank": false,
  "initialBalance" : 2000
}
```

删除ID对应的公司账号信息（公司进行注销）

```
DELETE /DeleteRegisteredCompany/{id}
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: /DeleteRegisteredCompany/{id}

参数名	类型	描述	状态
State	string	状态	success,not_login,bad_req
Data	String	空	暂无

- 参数使用json形式解析

Example

```
{
  "State":"success",
  "Data":""
}
```

Invoices

获取全部Invoice

```
GET /AllInvoices
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: /AllInvoices

参数名	类型	描述
State	string	状态
Data	Array	Invoice信息

- 参数使用json形式解析

Example

```
{
 "State": "success",
 "Data": {
  "ID": "3",
  "payerCompanyName": "牛牛火锅",
  "payeeeCompanyName": "雪花牛肉",
  "OwnID": "2",
  "invoiceAmount": 1500,
  "invoiceDeadline": "2021/06/01 23:59:59"
 }
 }
}
```

发布新Invoice

```
Post /RegisterNewInvoice
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: RegisterNewInvoice

参数名	类型	描述
State	string	状态
ID	string	数据

- 参数使用json形式解析

Example

```
{
    "State": "success",
    "Data": ""
}
```

根据ID查找对应的Invoice

```
GET /AllInvoices/{id}
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: /AllInvoices/{id}

参数名	类型	描述
State	string	状态
ID	Long	Invoice对应的ID
payerCompanyName	String	支付公司名称
payeeeCompanyName	String	收款公司名称
invoiceAmount	int	账单对应金额
invoiceDeadline	String	对应的需要还款日期

- 参数使用json形式解析

Example

```
{
 "State": "success",
 "Data":   {
    "ID": "3",
    "payerCompanyName": "牛牛火锅",
    "payeeeCompanyName": "雪花牛肉",
    "OwnID": "2",
    "invoiceAmount": 1500,
    "invoiceDeadline": "2021/06/01 23:59:59"
 }
}
```

进行交易的转让

```
POST /TransferInvoice
```

Request

参数名	类型	描述
payerCompanyName	string	需要付款的公司名称

payeeeCompanyName| isBank | Bool | 接受付款的公司名称 | | invoiceAmount | int | 账单对应的金额 | | invoice_id | Long | 对应的账单凭据 | | invoiceDeadline | String | 最后还款日期 |

- 参数使用json形式提交

Example

```
{
    "payerCompanyName": "雪花牛肉",
    "payeeeCompanyName": "雪花公司",
  "invoiceAmount":200,
  "invoice_id":2,
  "invoiceDeadline":"2021/06/30 23:43:23"
}
```

Response

> Status: 200 OK
>
> Location: /TransferInvoice

参数名	类型	描述
State	string	状态
Data	String	无

- 参数使用json形式解析

Example

```
{
 "State": "success",
 "Data": ""
}
```

进行账单还款

```
POST /PayInvoice/{id}
```

Request

空

Example

空

Response

> Status: 200 OK
>
> Location: /PayInvoice/{id}

参数名	类型	描述	参数
State	string	状态	success,money_not_enough,bad_req,invoice_not_exit
Data	string	数据	无

Example

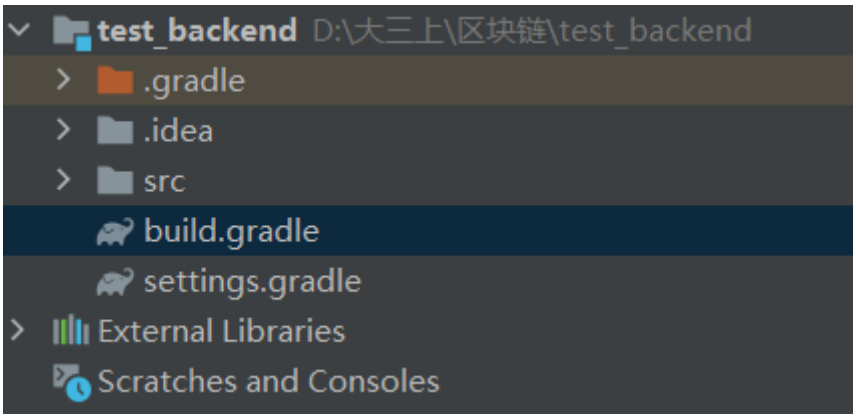
```
{
    "State": "success",
    "Data": ""
}
```

开发过程遇到的问题及解决方法

1. 当使用idea创建gradle项目的时候，发现出现build failed情况，代码全部是灰色的。上网查阅资料发现可能是因为gradle包出现了问题，所以自己下载了gradle最新版，使用的本地gradle包进行创建，成功解决问题。
2. 创建的gradle项目没有src文件夹，搜索之后添加了如下的task，reload之后可以看到文件夹出现：

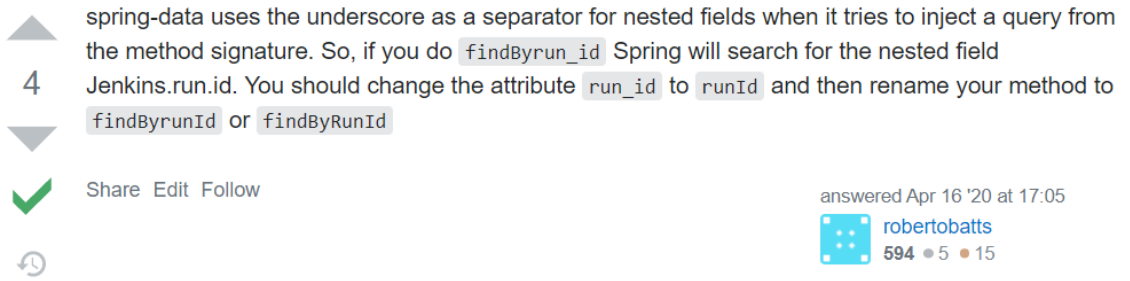
```
task"create-dirs"  {

    sourceSets*.java.srcDirs*.each { Set<File> it ->

        it.mkdirs()

    }

    sourceSets*.resources.srcDirs*.each{ Set<File> it ->

        it.mkdirs()

    }
}
```

（注意，<<操作符在5.x版本中被弃用了）



同时，创建的gradle项目如果每次都用现下载的包下载速度会非常慢，除了换源地址之外，还可以直接修改设置，以本地的gradle包进行项目的导入。

3. 当创建applicationContext的xml配置文件的时候，遇到了比较多的问题。一开始直接在new里面找xml类型的文件，但是发现只能创建一个JSP Tag Library Descriptor文件。从网上查找资料，发现可以创建一个resource bundle，或者创建一个xml类型的模板，将代码拷贝过去。
4. 后端在自定义实体类的时候，需要注意属性名称定义，不能以大写字母开头。遇到了Spring Data failed to create query for method的报错，查了很久没找到具体是哪里的问题，后来上网搜了很多资料才发现是因为命名不规范的问题。 有些比较简单的方法，比如按公司名称或者ID查找对应信息，可以直接用规范的方法名称直接在接口中定义，而不需要自己写实现方法。但是这种方法比如FindByCompanyName()这个方法，会按照companyName去匹配，如果自己定义的实体类中属性名称是CompanyName而不是companyName，那么就会遇到这种问题。



供应链前端UI开发框架

前端开发框架

- 框架
  - Vue.js v2.6.10
- 样式库
  - Primer CSS v15.2.0
  - Ant Design of Vue v1.7.2
- RESTful API
  - 前后端根据 RESTful 风格 API 进行同步开发，API 文档详见上节

前端模块架构

根据项目需要实现的四个主功能：

> - 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。

前端 UI 需要四个主功能页面，此外还需要路由通用模块、主页、注册登录通用模块、导航通用模块、用户信息展示与检索页面。

代码基本结构根据 Vue 项目文件结构和风格指南编写，下面说明项目相关模块

- 根模块：App.vue
- 路由模块：
  - router
- 通用组件模块：
  - NavigationBar.vue
  - Login.vue
- 页面组件模块
  - Financing.vue
  - InvoiceDetail.vue
  - InvoiceInquiry.vue
  - Settlement.vue
  - SignInvoice.vue
  - SignUp.vue
  - UserDetail.vue
  - Welcome.vue

前端实现

主要内容

- Invoice 资源客户端 CRUD 服务请求逻辑
- 四个主功能页面、主页客户端服务请求、功能逻辑、布局样式
- 路由通用模块、注册登录通用模块服务请求、功能逻辑
- 导航通用模块、用户信息展示与检索页面客户端服务请求、功能逻辑、布局样式
- 客户端功能测试

1. Invoice 资源客户端 CRUD 服务请求逻辑
   - 使用 http、axios 服务

   > 在 vue.config.js 中进行 api url 代理配置

```
devServer: {
 open: true,
 host: "localhost",
 port: 8082,
 proxy: {
   '/api': {
     target: 'http://localhost:8080',
     ws: true,
     changeOrigin: true,
     pathRewrite: {
```

```
        '^/api': ''
      }
    }
  }
}
```

在 main.js 中进行服务注册

```
import axios from 'axios'
import VueAxios from 'vue-axios'
Vue.use(VueAxios, axios)
axios.defaults.baseURL = '/api'
```

- 根据 API 文档编写客户端数据结构，预处理表单数据
- 发起服务请求
- 处理回传数据

```
// 请求
this.axios.request({
    // 结构预处理
    headers: {
        'Content-Type': 'application/json;charset=UTF-8'
    },
    url: '/RegisterNewInvoice',
    method: 'POST',
    data: JSON.stringify({
        payerCompanyName: formatValue.payerCompanyName,
        payeeeCompanyName: formatValue.payeeCompanyName,
        invoiceAmount: formatValue.invoiceAmount,
        invoiceDeadline: formatValue.invoiceDeadLine
    }),
    responseType: 'json'
}).then(function (response) {
    console.log(response)
    if (response.status == 200) {
        // ...
        // 根据 API 文档接口内容进行回传处理
    } else {
        // ...
        // 错误处理
    }
})
```

2. 路由通用模块
    - 使用 Vue Router 服务
        1. 使用模块化机制编程，导入 Vue 和 VueRouter，调用 Vue.use(VueRouter)
        2. 定义路由，映射组件
        3. 创建 router 实例，进行 `routes` 配置
        4. 通过 router 配置参数注入路由，创建和挂载根实例
3. 其他组件模块

    基本按照 Invoice 资源客户端 CRUD 服务请求逻辑的思路进行编写

前端测试样例剧本

根据主要功能的需求设计了以下的测试剧本并进行了演示视频的录制和说明：

- 基于区块链的供应链金融平台实现了四个基本功能： 功能一：实现采购商品—签发应收账款交易上链 功能二：实现应收账款的转让上链 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
- 下面进行功能展示

  首先登入牛牛火锅的公司账户 可以看到账户类型和账户余额 以及账号相关的单据：应还账单和应收账单 点击单据详情可以看到单据状态、应收截止日期、应收金额等信息

  - 功能一：实现采购商品—签发应收账款交易上链 下面进行签发单据 牛牛火锅有限公司向雪花牛肉厂购买长期供货服务并签订应收账款单据 生成了单据 40 登录雪花牛肉厂账户 在应收账单中可以看到刚刚生成的单据 40
  - 功能二：实现应收账款的转让上链 下面使用刚刚生成的单据 40 进行转让单据 雪花牛肉厂向切割机器械厂购买一批器械，将于牛牛火锅有限公司的应收账款单据部分转让给切割机器械厂，切割机器械厂可以利用这个新的单据去融资或者要求雪花牛肉厂到期时归还钱款 生成了新单据 41 登录切割机器械厂账户 在应收账单中可以看到刚刚生成的单据 41
  - 功能三：利用应收账款向银行融资上链 下面使用刚刚生成的单据 41 进行融资 切割机器械厂使用和雪花牛肉厂签订的单据向六六银行申请融资
  - 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。登录牛牛火锅的公司账户进行结算 结算后可以看到现在这张账单已经不再是应还账单 登录雪花牛肉厂的公司账户可以看到结算后余额变化
  - 另外也可以根据单据编号进行单据查询
- 演示视频见文件目录内

项目总结

经过这次实验，能够实际体验到区块链相关项目如何开发，以及如何根据实际的业务逻辑编写智能合约。同时，更清楚的了解了 fisco bcos 的整体框架，根据自学官网上的开发教程，能够很快的入手一些比较简单基本的应用开发功能。 也通过智能合约的编写更加熟练地巩固了课堂上学习的理论知识。同时在完整的链段、后端、前端开发过程中，接触到每个部分的开发工作以及各同学负责的各部分之间的协作技术、工具，在项目开发中学到了很多，收获很大。