2023 年四川省职业院校技能大赛

高职组

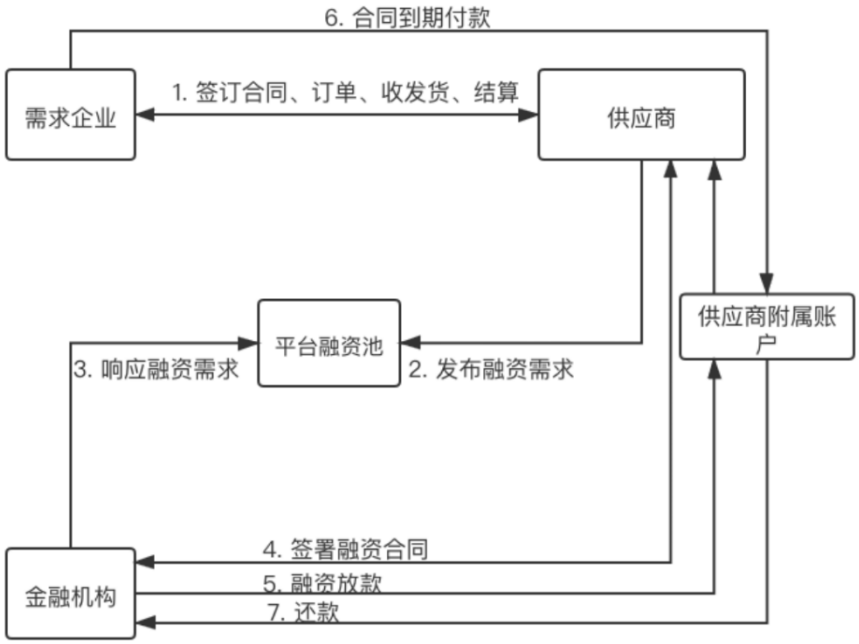
“区块链技术应用”赛项

样题

背景描述

在供应链金融这个万亿级市场中，区块链正在快速商业化落地，助力产业革新。基于区块链的供应链金融业务的理念是：以源自企业的应收账款为底层资产，通过区块链技术实现债券凭证的转让拆分。其中，在原始资产上链时，通过对应收账款进行审核校验，确认贸易关系和身份真实有效，和保证上链资产的真实可信。再者，债权凭证可基于供应链进行层层拆分与流转，都可完整追溯到最底层资产，以实现核心企业和金融机构对供应商的“信用穿透”。

某公司规划开发一个区块链供应链金融平台，包括核心企业、供应商、银行等角色，通过智能合约代码逐步构建区块链供应链金融平台的基本功能，实现银行向核心企业提供授信并发行数字凭证，企业与企业之间转让数字凭证。此外需要完成区块链供应链金融平台的前后端，实现基本的业务逻辑。



模块一：区块链产品方案设计及系统运维（35 分）

任务 1-1：区块链系统部署与运维（25 分）

围绕供应链金融区块链平台部署与运维需求，进行项目相关系统、节点以及管理工具的部署工作。通过监控工具完成对网络、节点服务的监控。最终根据业务需求规范，完成系统日志、网络参数、节点服务等系统结构的维护。

1. 根据参数与端口设置要求，部署区块链系统并验证；

2. 根据参数与端口设置要求，部署区块链网络管理平台并验证；

3. 基于区块链系统相关管理平台，按照任务指南实施系统运维工作并验证。

4. 基于区块链系统相关监管工具，按照任务指南对区块链系统进行监管。

子任务 1-1-1： 搭建区块链系统并验证（8 分）

基于给定服务器环境以及软件（地址“/root/tools”），使用 Docker 以默认配置安装单机 4 节点的区块链系统，并完成控制台工具的部署：

(1) 完成系统搭建配置与启动。（2 分）

(2) 使用基于 Docker 命令查看区块链系统状态。（2 分）

(3) 检查区块链系统节点 node0 连接状态输出。（2 分）

(4) 配置控制台，管理相关证书并启动。（2 分）

子任务 1-1-2：区块链管理平台部署与验证（8 分）

基于给定服务器环境以及软件（地址“/root/tools”），按要求部署区块链管理平台，具体工作如下：

(1) 配置 Mysql 数据库（2 分）

(2) 配置管理平台连接区块链系统（2 分）

(3) 使用命令启动管理平台服务（2 分）

(4) 验证管理平台启动情况（2 分）

子任务 1-1-3：区块链系统节点运维（5 分）

基于已完成的区块链系统与管理平台搭建工作，开展相关节点运维工作：

(1) 生成新节点(node4)，启动并检查（2 分）

(2) 修改新节点配置，并查看节点的 nodeid（2 分）

(3) 将新节点作为观察节点加入 group1 当中，并检查是否加入成功（1 分）

子任务 1-1-4：区块链系统管理平台运维（4 分）

基于已部署的区块链系统管理平台，进行系统相关运维工作：

(1) 基于管理平台功能页面，添加新主机（2 分）

(2) 基于管理平台功能页面，修改新节点（node4）节点状态，并监控。（2 分）

任务 1-2：区块链系统测试（10 分）

设计对区块链系统的测试流程；结合实际业务需求，调用部署的智能合约中进行系统测试、性能测试等；根据业务需求，分析并且修复给定智能合约中的安全漏洞。利用模拟业务和测试工具来完成对区块链系统服务数据的测试。

1. 基于区块链系统的中间件服务的部署脚本完成中间件服务环境搭建以及搭建结果验证，最后将执行结果截图保存。（3 分）

(1) 实现区块链系统中间件服务平台部署。（1 分）

(2) 实现区块链系统中间件服务签名功能启动情况验证。（1 分）

(3) 区块链中间件服务节点管理进程启动情况验证和浏览器验证。（1 分）

2. 智能合约安全漏洞测试。（7 分）

有如下智能合约：

```
pragma solidity ^0.7.6;

contract TimeLock {
    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = block.timestamp + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

    function withdraw() public {
        require(balances[msg.sender] > 0, "Insufficient funds");
        require(block.timestamp > lockTime[msg.sender], "Lock time not expired");
```

```
        uint amount = balances[msg.sender];
        balances[msg.sender] = 0;

        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent, "Failed to send Ether");
    }
}

contract Attack {
    TimeLock timeLock;

    constructor(TimeLock _timeLock) {
        timeLock = TimeLock(_timeLock);
    }

    fallback() external payable {}

    function attack() public payable {
        timeLock.deposit{value: msg.value}();
        timeLock.increaseLockTime(
            type(uint).max + 1 - timeLock.lockTime(address(this))
        );
        timeLock.withdraw();
    }
}
```

如上代码主要实现功能为规定了转账冻结时间，在冻结时间内用户不能提取存款的金额。

（1）分析智能合约中存在问题，并说明危害。（2 分）

（2）根据测试工具中的代码文件，编写测试用例，复现智能合约中存在的漏洞。（3 分）

（3）创建新的智能合约，修复其中问题，说明修复内容并测试。（2 分）

模块二：智能合约开发与测试（30 分）

任务 2-1：智能合约开发（20 分）

使用 Solidity 语言完成智能合约开发、部署和调用，要求如下：

1.供应链金融实体信息编码（6 分）

（1）编写供应链金融智能合约的实体接口，完成实体通用数据的初始化，实现企业和票据实体信息上链的功能；（2 分）

表 2.2.2.1 SupplyChain 实体说明

名称	类型	说明
companyName	string	公司名称
companyAddress	address	公司地址
creditAsset	uint	信用资产
acceptReceiptIndex	uint[]	接收的凭证
sendReceiptIndex	uint[]	发送的凭证
senderAddress	address	发送票据的地址
accepterAddress	address	接收票据的地址
receiptType	uint8	凭证类型
transferType	uint8	交易类型
amount	uint	交易数量

```
//公司信息结构体
        struct Company {
            //①公司名称
            //②公司地址
            //③信用资产
            //④接收的凭证
            //⑤发送的凭证
        }
```

```
//数字发票收据信息
    struct Receipt {
        //⑥发送票据的地址
        //⑦接收票据的地址
        //⑧凭证类型
        //⑨交易类型
        //⑩交易数量
    }
```

（2）编写企业上链信息接口，实现供应链金融的企业信息上链；（2 分）

```
function addCompany(string name, address companyAddress) returns(bool) {
        //①实例化公司
        //②添加公司地址
        //③将实例化的公司添加到公司映射
        //④返回添加成功标识
    }
```

（3）基于给定的智能合约代码以及注释，完成银行向企业交易的接口函数；（2 分）

```
function bankToCompanyReceipt(address senderAddress, address accepterAddress, uint amount, uint8 receiptType) returns(uint) {
①判断接收地址存在
        ②实例化银行
        ③实例化公司

        if (keccak256(bank.bankName) == keccak256("")) {
            return 404001;
        }
```

```
        //确认公司存在
        if (keccak256(company.companyName) == ④) {
            return 404002;
        }

        if (bank.creditAsset < amount) {
            return 500001;
        }
```

2.供应链金融公司与公司接口编码（6分）

（1）编写公司与公司之间进行交易的历史存证上链接口，实现公司与公司之间的交易功能；（2分）

```
function companyToCompanyReceipt(①, address accepterAddress, uint amount, uint8 receiptType) returns(uint) {

        //②接收地址判断
        Company memory senderCompany = companyMap[③];
        Company memory ④ = companyMap[accepterAddress];
        //确认发送公司存在
        if (keccak256(senderCompany.⑤) == keccak256("")) {
            return 404001;
        }

        //确认接收公司存在
        if (keccak256(accepterCompany.companyName) == ⑥) {
            return 404002;
        }

        //如果存证接收的公司资产小于存证数额,那么就不能交易发送存证
```

```
            if (accepterCompany.creditAsset ⑦ ⑧) {
                return 500001;
            }
```

（2）编写创建存证的接口，实现创建存证的功能；（2 分）

```
Receipt memory newReceipt = Receipt(①, accepterAddress, receiptType, 2, amount);
            receiptIndex += 1;
            //记录存证(存证 Map,公司 Map 对应地址的发送和接收存证列表)
            receiptMap[receiptIndex] = ②;
            companyMap[③].sendReceiptIndex.push(receiptIndex);
            companyMap[accepterAddress].acceptReceiptIndex.push(④);
```

（3）编写交易金额数量变化的接口，实现凭证交易双方资金的变化功能；（2 分）

```
companyMap[①].creditAsset ② amount;
            companyMap[③].creditAsset ④ amount;
            return 200;
        }
```

3.供应链金融公司与银行交易的接口编码（4分）

（1）编写公司与银行之间进行交易的历史存证上链接口，实现公司与银行之间的交易功能；（2分）

```
function companyToBankReceipt(address senderAddress, ① , uint amount, uint8 receiptType) returns(uint) {
            ②

            Bank memory bank = bankMap[senderAddress];
            Company memory accepterCompany = companyMap[③];
```

```
        //确认发送公司存在
        if (keccak256(bank.bankName) == ④) {
            return 404001;
        }

        //确认接收公司存在
        if        (keccak256(accepterCompany.companyName)        ==
keccak256("")) {
            return 404002;
        }

        //如果存证接收的公司资产小于存证数额,那么就不能交易发送存
证
        if (accepterCompany.creditAsset < amount) {
            return 500001;
        }
```

（2）编写创建存证的接口，实现创建存证的功能；（1 分）

```
    //创建存证
        Receipt   memory   newReceipt   =   Receipt(senderAddress,
accepterAddress, ①, 3, amount);
        receiptIndex ② 1;
        receiptMap[③] = newReceipt;

bankMap[senderAddress].sendReceiptIndex.push(receiptIndex);
        companyMap[accepterAddress].④;
```

（3）编写交易金额数量变化的接口，实现凭证交易双方资金的变化功能；（1 分）

```
    bankMap[senderAddress].① ② amount;
```

companyMap[accepterAddress].③ ④ amount;

return 200;

}

4. 合约编译、部署和调用（4 分）

（1）解决代码错误和警告，正确编译并部署合约，成功获取部署的合约地址和 abi。（1 分）

（2）调用食品溯源智能合约的接口，完整验证业务流程。（3 分）

任务 2-2：智能合约测试（10 分）

编写智能合约单元测试代码并完成合约功能测试、性能测试，具体要求如下：

1.配置区块链网络（2 分）

启动区块链网络，创建新的 Workspace，配置对外访问的 RPC 接口为 7545，配置项目的配置文件 config.js 实现与新建 Workspace 的连接。

2.补充给定基础代码中注释提示的部署逻辑（2 分）

基于 VSCODE 加载的测试项目，补全位于 test 文件夹中 HelloWorld.js 文件预操作的方法。在测试文件中添加预定义的方法（在其他方法启动前使用）。

3.补充代码中注释提示的测试逻辑（2 分）

基于 VSCODE 加载的测试项目，补全位于 test 文件夹中 HelloWorld.js 文件，添加测试用例，测试智能合约的 get 方法。

4.测试 hello.get()方法（2 分）

基于 VSCODE 加载的测试项目，补全位于 test 文件夹中 HelloWorld.js 文件，添加测试用例，测试智能合约的 hello.get()方法。

5.测试.should.equal 进行对比判断（2 分）

基于 VSCODE 加载的测试项目，补全位于 test 文件夹中 HelloWorld.js 文件，添加测试用例，测试智能合约的 equal 字符串比较方法。

□

模块三：区块链应用系统开发（30 分）

任务 3-1：区块链应用前端功能开发（10 分）

1.请基于前端系统的开发模板，在注册组件 Register.vue 文件中添加对应的注册逻辑代码，实现对后端系统的注册功能，并测试功能完整性（3 分）：

本题目的具体要求如下：

(1) 界面有明确的注册相关提示语

(2) 需要填写的部分有组织名称、区块链地址、组织类型

(3) 页面需要有“返回”按钮，可以跳转到登录页面

(4) 点击“注册”按钮时需要检查区块链地址是否已输入

(5) 注册成功后跳转登录页面

Register.vue:

代码片段 1：

```
<el-row>
        <el-col :span="16" :offset="4">
          <el-form label-width="100px">
            <h3>选手填写部分</h3>
            <el-form-item label="组织名称:">
              <el-input  type="primary"  v-model="选手填写部分"></el-input>
            </el-form-item>
            <el-form-item label="区块链地址:">
              <el-input  type="primary"  v-model="选手填写部分"></el-input>
            </el-form-item>
            <el-form-item label="组织类型:">
              <el-radio-group v-model="orgType">
              <el-radio :label="1">公司</el-radio>
```

```
            <el-radio :label="2">银行</el-radio>
            </el-radio-group>
          </el-form-item>
        </el-form>
      </el-col>
    </el-row>
    <el-row style="padding-bottom:20px">
      <el-button type="primary" 选手填写部分>注册</el-button>
      <el-button type="primary" 选手填写部分>返回</el-button>
    </el-row>
```

代码片段 2：

```
register: function () {
    if (this.address == "") {
      alert(选手填写部分)
    }else {
      let postData = {
        orgType: 选手填写部分,
        username: 选手填写部分,
        address:选手填写部分
      }
// 和后端交互
      选手填写部分
    }
  },
```

代码片段 3：

```
goback: function () {
    this.orgType = ''
    this.username =  ''
```

```
        this.address = ''
        选手填写部分
      }
```

2.请基于前端系统的开发模板，在登录组件Login.vue文件中添加对应的登录逻辑代码，实现对后端系统的登录功能，并测试功能完整性（3分）：

本题目的具体要求如下：

(1) 界面有明确的登录相关提示语

(2) 需要填写的项有用户地址、组织类型

(3) 页面需要有“注册”按钮，可以跳转注册页面

(4) 点击“登录”按钮时需要检查各个表项是否已输入

(5) 登录成功后跳转首页，路径为“/home”

Login.vue:

代码片段1:

```
<el-col :span="16" :offset="4">
          <el-form label-width="80px">
            <h1>供应链金融应用</h1>
            <h3>选手填写部分</h3>
            <el-form-item label="用户地址:">
              <el-input  type="primary"  v-model="选手填写部分
"></el-input>
            </el-form-item>
            <el-form-item label="组织类型:">
              <el-radio-group v-model="选手填写部分">
              <el-radio :label="1">公司</el-radio>
              <el-radio :label="2">银行</el-radio>
              </el-radio-group>
            </el-form-item>
          </el-form>
        </el-col>
```

```
      </el-row>
      <el-row style="margin-bottom: 20px">
        <el-button type="primary" 选手填写部分>登录</el-button>
        <el-button type="primary" 选手填写部分>注册</el-button>
      </el-row>      </el-row>
```

代码片段 2：

```
login: function () {
      if (this.address == "") {
        alert("选手填写部分")
      }else if (this.orgType == "") {
        alert("选手填写部分")
      }else {
        let postData = {
          orgType: 选手填写部分,
          address: 选手填写部分
        }
        // 与后端交互
选手填写部分
      }
},
```

代码片段 3：

```
register: function () {
          选手填写部分
      },
```

3.请基于前端系统的开发模板，在公司组件 Company.vue 文件中添加对应的逻辑代码，实现对后端系统的公司相关业务功能，并测试功能完整性（2 分）：

Company.vue:

代码片段 1：

```
<el-row>
            <el-col :span="20" :offset="2">
            <el-table :data="companyList" style="font-size: 20px">
              <el-table-column  prop="address"  label="账户地址
"></el-table-column>
              <el-table-column  prop="name"  label="公司名称
"></el-table-column>
              <el-table-column  prop="amount"  label="账户总额
"></el-table-column>
              <el-table-column label="查看详情">
                <template slot-scope="scope">
                  <el-button type="primary" @click="选手填写部分">
查询</el-button>
                </template>
              </el-table-column>
              <el-table-column prop="receiptType" label="转账">
                <template slot-scope="scope">
                  <el-button                    type="primary"
@click="transfer(scope.row)">操作</el-button>
                </template>
              </el-table-column>
            </el-table>
          </el-col>
          </el-row>
          <el-row>
```

代码片段 2：

```
<el-dialog title="公司详情" :visible.sync="dialogVisible">
```

```
<el-form label-width="100px">
  <el-form-item label="账户地址:">
    {{选手填写部分 }}
  </el-form-item>
  <el-form-item label="公司名称:">
    {{ 选手填写部分 }}
  </el-form-item>
  <el-form-item label="账户总额:">
    {{ 选手填写部分 }}
  </el-form-item>
</el-form>
```

代码片段 3：

```
detail: function (queryAddress) {
      this.dialogVisible = true
      let address = 选手填写部分
      this.axios.get(`      选      手      填      写      部
分?address=${address}&queryAddress=${queryAddress}`)
        .then((response) => {
          console.log(response)
          if (response.data.code == 200) {
            let                    inAddress                    =
response.data.data.companyVO.address;
            let inName = 选手填写部分;
            let inAmount = response.data.data.companyVO.amount;
            this.companyDetail = {
              address: 选手填写部分,
              name: 选手填写部分,
              amount: 选手填写部分,
              senderReceiptList:
```

```
response.data.data.senderReceiptList,
                    accepterReceiptList:
response.data.data.accepterReceiptList
                  }
                } else {
                  alert(`请求内容有误，${response.data.data}`)
                }
              })
        },
```

4.请基于前端系统的开发模板，在银行组件 Bank.vue 文件中添加对应的逻辑代码，实现对后端系统的银行相关业务功能，并测试功能完整性（2 分）：

Bank.vue:

代码片段 1：

```
<el-row>
      <el-dialog title="交易（发送凭证）页" :visible.sync="transDialogVisible" width="30%">
        <el-form label-width="100px">
          <el-form-item label="发送账户地址:">
            {{选手填写部分}}
          </el-form-item>
          <el-form-item label="接收账户地址:">
            {{选手填写部分}}
          </el-form-item>
          <el-form-item label="交易额:">
            <el-col :span="16" :offset="4">
              <el-input type="primary" v-model="选手填写部分"></el-input>
```

```
              </el-col>
            </el-form-item>
            <el-form-item label="凭证类型:">
              <el-select v-model="选手填写部分" placeholder="请选择
">
                <el-option
                    v-for="item in options"
                    :key="item.value"
                    :label="item.label"
                    :value="item.value">
                </el-option>
              </el-select>
            </el-form-item>
          </el-form>
          <el-row>
            <el-button type="primary" size="medium" @click="选手填写
部分">确定</el-button>
          </el-row>
        </el-dialog>
      </el-row>
```

代码片段 2:

```
executeTransaction: function() {
      let funcName = "companyToBankReceipt";
      if (this.transDetail.amount ==选手填写部分) {
        alert('交易额不能为空！')
        return
      }
      if (this.$cookies.get('orgType') == 选手填写部分) {
        alert('银行不能给银行发送凭证！')
```

```
        return
      }
      if (选手填写部分) {
        alert("凭证发送账户和接收账户不能相一致！")
        return
      }
      this.axios.post(`/finance/transaction/${funcName}`, 选手填写部分).then((response) => {
        if (response.data.code == 200) {
          alert('凭证发送成功')
          this.query()
          this.transDialogVisible = false
        }else {
          alert(`凭证发送失败, ${response.data.data}`)
        }
      })
    },
```

任务 3-2：区块链应用后端功能开发（20 分）

1.开发区块链供应链金融应用后端系统中用户功能模块对应的用户注册功能，根据前后代码补充最合适的代码，并测试功能完整性。(4 分)

OrgServiceImpl.java:

```
 /**
     * 注册 Service
     * RegisterBO registerBO
     * */
    @Override
    public Result<String> register(RegisterBO registerBO) {
```

```
        if (StrUtil.isEmpty(选手填写部分) ||
                StrUtil.isEmpty(选手填写部分) ||
                registerBO.getOrgType() == 选手填写部分
        ) {
            return Result.error(ResultVO.PARAM_EMPTY);
        }

        List funcParam = new ArrayList();
        funcParam.add(选手填写部分);
        funcParam.add(选手填写部分);
        if(registerBO.getOrgType() == 2){
            funcParam.add(BigInteger.valueOf(1000));
        }

        String funcName;
        if(registerBO.getOrgType() == 2){
            funcName =选手填写部分;
        }else{
            funcName =选手填写部分;
        }

        String _result = weBASEUtils.funcPost(OWNER_ADDRESS,funcName,funcParam);
        JSONObject _resultJson = JSONUtil.parseObj(_result);
        if (_resultJson.containsKey("statusOK") == false || _resultJson.getBool("statusOK") != true){ // _resultJson.getInt("code") > 0
            return Result.error(ResultVO.选手填写部分);
        }
```

```
        return Result.success("ok");
    }
```

2.开发区块链供应链金融应用中后端系统中用户功能模块对应的用户登录功能，根据前后代码补充最合适的代码，并测试功能完整性。(4 分)

OrgServiceImpl.java:

```
/**
     * 登录 Service
     * LoginBO loginBO
     **/
    @Override
    public Result<String> login(@RequestBody LoginBO loginBO) {
        if (StrUtil.isEmpty(loginBO.getAddress())
        ) {
            return Result.error(ResultVO.PARAM_EMPTY);
        }

        List funcParam = new ArrayList();
        funcParam.add(选手填写部分);

        String funcName;
        if(loginBO.getOrgType() == 2){
            funcName =选手填写部分;
        }else{
            funcName =选手填写部分;
        }
```

```
            String _result = weBASEUtils.funcPost(选手填写部分,funcName,funcParam);
            JSONArray _resultJson = JSONUtil.parseArray(_result);
            if (StrUtil.isEmpty(_resultJson.get(0).toString())){
                return Result.error(ResultVO.选手填写部分);
            }

            return Result.success("ok");
    }
```

3.开发区块链供应链金融应用后端系统中查询功能模块对应的查询所有公司信息功能，根据前后代码补充最合适的代码，并测试功能完整性。(4 分)

QueryServiceImpl.java:

```
    /**
         * 获取所有公司数据，不包含存证详细信息
         * */
        @Override
        public Result listCompany(String userAddress){

            String _result = weBASEUtils.funcPost(userAddress,"getAllCompanyAddress",new ArrayList());

            try {
                JSONArray respArray = 选手填写部分;
                String a = respArray.get(0).toString();
                JSONArray addressArray = 选手填写部分;
                List<CompanyVO> companyList = new ArrayList<>();
```

```
            for(Object obj: addressArray) {
                CompanyVO companyVO = 选手填写部分;
                companyList.add(companyVO);
            }
            return Result.success(选手填写部分);
        }catch (Exception e) {
            ResultVO resultVO = ResultVO.CONTRACT_ERROR;
            resultVO.setData(选手填写部分);
            return Result.error(选手填写部分);
        }
    }
```

4.开发区块链供应链金融应用的后端系统中查询功能模块对应的查询银行凭证列表功能根据前后代码补充最合适的代码，并测试功能完整性。(4 分)

QueryServiceImpl.java:

```
/**
     * 获取包括银行详情，银行发送凭证列表，银行接收凭证列表
     * */
    @Override
    public Result getBankEntity(String userAddress, String queryAddress){
        BankVO bankVO = 选手填写部分;
        List<ReceiptVO> senderReceiptList = 选手填写部分;
        List<ReceiptVO> accepterReceiptList = 选手填写部分;
        BankEntityVO bankEntityVO = new BankEntityVO();
        bankEntityVO.setBankVO(选手填写部分);
        bankEntityVO.setSenderReceiptList(选手填写部分);
        bankEntityVO.setAccepterReceiptList(选手填写部分);
        return Result.success(bankEntityVO);
```

```
    }
```

5.开发区块链供应链金融应用的后端系统中交易功能模块对应的银行向公司交易凭证功能，根据前后代码补充最合适的代码，并测试功能完整性。(2分)

TransactionServiceImpl.java:

```
/**
     * 公司向银行发送凭证（银行转账给公司）
     * */
    public Result<String> bankToCompanyReceipt(TransactionBO transactionBO) {
        选手填写部分;
}
```

6.开发区块链供应链金融应用的后端系统中交易功能模块对应的公司向公司交易凭证功能，根据前后代码补充最合适的代码，并测试功能完整性。(2分)

TransactionServiceImpl.java:

```
 /**
     * 公司向公司的凭证发送
     * */
    public Result<String> companyToCompanyReceipt(TransactionBO transactionBO) {
        选手填写部分;
    }
```