2024 年河北省职业院校技能大赛

高职组

“区块链技术应用”赛项

样题三

任

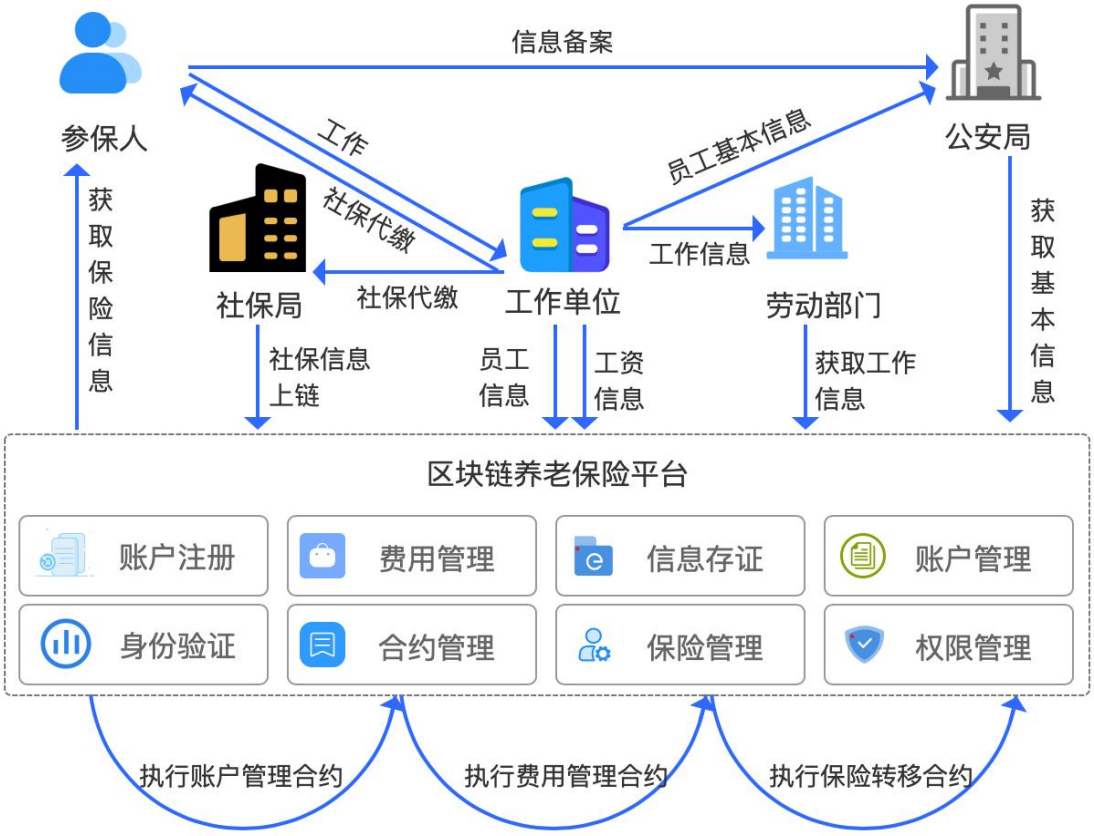
务

书

参赛队编号：_____

背景描述

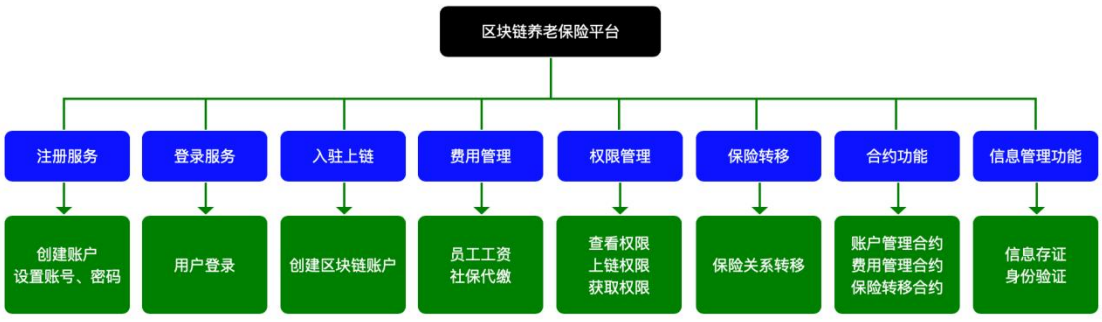
养老保险是对于老年人的最基本的生活保障。各种数据显示，当前的养老金市场规模庞大。2016 年美国的养老金资产总额为 19.1 万亿美元；而据估计，截至 2016 年，全球养老金资产总额约为 36 万亿美元。美国是最大的养老保险市场，其占全球份额为 62%；其次是英国(7.9%)和日本(7.7%)。然而，养老保险市场在各个方面存在诸多缺陷，往往会对个人造成负面影响。常见的问题包括治理不善和糟糕的绩效管理，以及在养老金缴纳过程中出现问题(因为没有标准的过程，其流程容易出现隐藏的成本、糟糕的数据管理，甚至是养老金欺诈)。



区块链养老保险平台业务流程图

现有一个基于区块链的职工养老保险平台，当工作单位 U 参保人 P 办理职工基本养老保险时，参保人所在的工作单位在养老保险管理平台 S 输入个人身份证号，S 从公安部门调取 P 的基本信息，从劳动

部门 L 调取 P 的工作单位信息、工作时间、工资，从城市 A 的社保局 N 上调取城市 A 的缴费基数，根据调取的信息生成养老保险账户。参保人 P 可以拥有多个工作单位，但 S 只允许一个工作单位为其缴纳养老保险费用，如果有其他工作单位为其缴纳，需要其他工作单位进行减员操作才可以进行缴费。城市 A 规定每月 P 的个人缴费比例为 8% ，称为个人账户，U 的缴费比例为 19%，称为统筹账户。城市 A 的缴费基数上限为 24000 元，下限为 3200 元。工作单位通过 S 将统筹账户的费用支付给 N，P 个人账户的缴费也由本公司代缴。若工作单位未在 30 天内缴纳费用，L 追究其公司的法律责任。P 月工资为 25000 元，则需要个人账户每月缴纳 2000 元，统筹账户缴纳 4750 元。当 P 在 S 中申请将养老保险关系从城市 A 转移到城市 B 时，S 将授权城市 B 的社保局 M 调取 P 的养老保险账户，M 依据 U 是否办理停缴手续来判断是否符合转移条件。如果 U 未办理停缴手续则复核失败。如果复核成功，则 N 将 P 的养老保险账户、全部个人账户基金和全部统筹账户基金转移到 M。M 建立 P 的参保档案，在 S 中接收个人账户基金、统筹账户基金、养老保险账户转移。最终将复核数据、转移数据和接收数据存储在 S 中，方便以后调用。



区块链养老保险平台系统架构图

模块一：区块链平台运维与系统测试（35 分）

任务 1-1：区块链产品需求分析与方案设计

【要求】

依据区块链养老保险应用的背景信息，编写该区块链产品的需求分析文档、绘制相关业务流程、用户用例图、业务系统功能结构图、列举功能清单，最后将完成的相关内容保存至作答区并提交。

【任务】

1.依据给定的背景信息，使用文字描述对区块链养老保险的应用进行需求分析，要求使用 Y 模型即需求场景、背后目标、产品功能进行分析；

2.依据给定的背景信息，对考题进行核心业务分析，使用 Visio 工具绘制系统业务流程图，包含投保人办理职工基本养老保险的核心业务流程、社保局查阅缴费信息生成养老保险账户的核心业务流程、工作单位为参保人缴纳养老保险的核心业务流程；

3.依据给定的背景信息，对考题进行核心业务分析，使用 Visio 工具绘制系统业务用例图，包含投保人办理职工基本养老保险的核心业务用例、社保局查阅缴费信息生成养老保险账户的核心业务用例、工作单位为财参保人缴纳养老保险的核心业务用例；

4.依据给定的背景信息，使用 Visio 工具绘制业务系统功能结构图；

5.依据给定的背景信息，使用表格编制业务系统功能清单。

任务 1-2：区块链平台部署与运维

围绕养老保险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。通过通过监控工具完成对网络、节点服务的监控。最终利用业务需求规范，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. 根据参数配置要求，安装并部署区块链系统控制台，检查部署控制台是否正常运行；

3. 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；

4. 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；

5. 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。

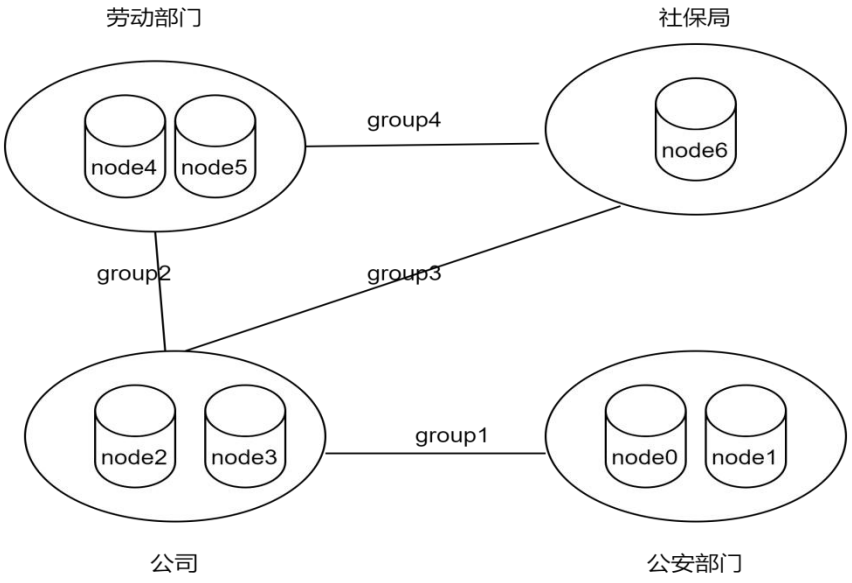
子任务 1-2-1： 按要求完成 FISCO BCOS 区块链系统部署并验证

【要求】

登陆 Linux 服务器，以普通搭链方式安装并部署如图所示的单机、四机构、四群组、八节点的星形组网拓扑区块链系统，在指定操作目录（/root/tools）中完成以下操作，并保证系统能正常运行。并将操作过程截图，保存至作答区并点击提交。

【任务】

登陆 Linux 服务器，以企业级部链方式安装并部署如图所示的单机、四机构、四群组、八节点的星形组网拓扑区块链系统。其中，四群组名称分别为 group1、group2、group3 和 group4，四个机构名称为公安部门(agencyPolice)、企业(agencyCompany)、劳动部门(agencyLabor)、社保局（agencySSB）。p2p_port、channel_port、jsonrpc_port 起始端口分别为 30330、20230、8545。确保搭建的区块链系统能正常运行，并将执行的命令和完整的命令执行结果截图保存至作答区并点击提交。具体任务如下：



（1）根据题目设置各项要求，完善 build_chain.sh 脚本内容，包括设置区

块链的网络过期时间为 90 天，设置 fisco bcos 链的共识算法为 rpbft,配置生成群组配置文件进行区块链系统的网络配置,并通过 build_chain 离线搭建区块链平台；

（2）通过命令启动所有节点，查看并验证节点进程状态；

（3）检查区块链节点 node0 的连接状态以及 node0 group1 和 node5 group2 的共识状态日志输出。

子任务 1-2-2：按要求完成区块链系统控制台 Console 的安装与运维

【要求】

登陆 Linux 服务器，安装、部署区块链系统控制台 Console，并完成节点的运维。同时，检查控制台是否能够正常运行。将操作过程截图，保存至作答区并点击提交。

【任务】

1. 登陆 linux 服务器，进入指定操作目录（/root/tools）按下列要求完成控制的安装与部署，并将安装过程和部署启动结果截图，保存至作答区并点击提交。具体操作任务如下：

（1）使用 tar 命令解压缩 console.tar.gz 文件,拷贝配置文件并查看拷贝是否成功；

（2）配置控制台链接 fisco 链的相关证书文件，并启动控制台；

（3）使用 get_account.sh 脚本生成账户 1、2、3，并分别测试以账户 1、2、3 登录控制台；

（4）将提供的三个账户设为三种角色,设定账户 1 为链管理员账户并验证，账户 2 为系统管理员账户，授权账户 2 可以部署合约和创建用户表并验证，账户 3 为普通账户并验证。

2. 基于已完成的区块链系统与控制台平台，开展区块链节点的授权管理与退出运维任务，并将运维过程及运行结果截图，保存至答题区。具体操作任务如下：

（1）完成授权管理节点权限：登录账户 1 的控制台，授权账户 2 拥有管理节点的权限，登录账户 2 的控制台，查看共识节点列表，将第一个 nodeID 对应

的节点设置为观察节点，登录账户 3 的控制台，将观察节点加入共识节点列表；

（2）完成修改系统参数权限：登录账户 1 的控制台，授权账户 2 拥有修改系统参数的权限，登录账户 2 的控制台，修改系统参数 tx_count_limit 的值为 2000，登录账户 3 的控制台，修改系统参数 tx_count_limit 的值为 3000；

（3）授权账户写用户表权限：通过账户 1 授权账户 3 可以写用户表 t_test 的权限；登录账户 3 的控制台，在用户表 t_test 插入一条记录，然后查询该表的记录；登录账户 2 的控制台，更新账户 3 插入的记录，并查询该表的记录；通过账户 1 撤销账户 3 写用户表 t_test 的权限。

子任务 1-2-3：按要求完成 WeBASE-Node-Manager 的安装与部署

【要求】

登录 Linux 服务器，部署节点管理平台，并将部署、启动、应用过程结果截图，保存至作答区并点击提交。

【任务】

登录 linux 服务器，进入指定操作目录（/root/tools）中完成区块链一体化管理平台的配置部署，并检查是否安装成功，具体操作任务如下：

（1）进入 WeBASE-Node-Manager 目录，完成数据库初始化操作；

（2）修改 application.yml 配置文件，进行 WeBASE-Node-Manager 的服务配置，包括数据库名称，数据库用户，数据库密码等；

（3）使用命令启动 WeBASE-Node-Manager 管理平台服务，并检查节点管理是否正常启动；

（4）进行节点管理服务的 API 接口测试。

子任务 1-2-4：区块链管理平台部署与验证

【要求】

登录 Linux 服务器，部署节点管理平台，并将部署、启动、应用过程结果截图，保存至作答区并点击提交。

【任务】

登录 linux 服务器，进入指定操作目录（/root/tools）中完成区块链一体化管理平台的配置部署，并检查是否安装成功，具体操作任务如下：

（1）进入 WeBASE-Web 目录，修改 docs 目录下配置文件 nginx.conf，设置

服务器 ip 地址、WeBASE-web 服务端口、静态文件路径，节点管理服务 ip 和端口；并将配置文件 nginx.conf 到默认配置目录/etc/nginx/conf.d 中；

（2）使用命令启动 nginx，并检查管理平台是否正常启动；

（3）通过浏览器访问管理平台页面，默认账号密码：admin/Abcd1234 。

任务 1-3：区块链系统测试

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。

子任务 1-3-1：按要求完成区块链系统的压力测试

【要求】

登录 Linux 服务器，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，并将测试设计与执行过程结果截图，保存至作答区并点击提交。

【任务】

登录 linux 服务器，使用 Caliper 测试工具对 CrowFund.sol 中 newNeeder 和 print 功能进行压力测试，具体操作任务如下：

（1）修改 4nodes1group/fisco-bcos.json 文件，进行压力测试 CrowFund.sol 合约文件配置；

（2）配置进行压测的 js 信息；

（3）提供 newNeeder 功能核心测试代码；

（4）提供 print 功能核心测试代码；

（5）设置 txNumber=100，tps=1，所有测试通过率为 100%。

子任务 1-3-2：按要求完成智能合约的安全漏洞测试

【要求】

根据给定的 Wallet 与 Attack 智能合约，基于已有的 truffle、ganache 环境和工程文件，使用 truffle 工具、JavaScript 语言完成智能合约的部署、测试用例代码编写，完成该智能合约的安全漏洞测试，并对其进行修复后重新测试验证。最后将完成的代码及测试结果截图，保存至作答区并点击提交。

【任务】

有如下问题智能合约，使用 truffle 工具完成对该智能合约的漏洞测试与修复。

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.11;

contract Wallet {
    address public owner;

    constructor() payable {
        owner = msg.sender;
    }

    function transfer(address payable _to, uint _amount) public {
        require(tx.origin == owner, "Not owner");
        (bool sent, ) = _to.call{value: _amount}("");
        require(sent, "Failed to send Ether");
    }
}

contract Attack {
    address payable public owner;
    Wallet wallet;

    constructor(Wallet _wallet) {
        wallet = Wallet(_wallet);
        owner = payable(msg.sender);
    }

    function attack() public {
        wallet.transfer(owner, address(wallet).balance);
```

```
    }
}
```

（1）分析以上智能合约中存在的漏洞，并说明其可能造成的危害；

（2）编写测试用例，复现智能合约中存在的漏洞；

（3）创建修复后的智能合约，命名为 setupRepair.sol，并编写测试用例修复结果，并说明修复内容。

模块二：智能合约开发与测试（30 分）

任务 2-1：智能合约设计

【要求】

根据区块链养老保险应用需求分析和方案设计文档的描述，编写该区块链产品的智能合约功能需求以及设计该智能合约的结构图，最后将完成的功能需求分析内容和结构图保存至作答区并点击提交。

【任务】

1. 根据区块链养老保险产品项目背景和需求分析，编写该区块链养老保险产品的智能合约功能需求文档；
2. 设计智能合约中各角色应具备的功能；
3. 完成区块链养老保险智能合约的时序图；

任务 2-2：按要求完成智能合约开发

【要求】

使用 Solidity 语言完成“养老保险”系统中智能合约功能代码。使用开发工具打开/home/admin1/PensionInsurance/contracts 目录下的相应合约，找到“**选手填写部分**”，并使用正确代码替换，最后将完成的代码结果截图保存至作答区并点击提交。

【任务】

1.养老保险智能合约（PensionInsurance）接口编码。

（1）编写养老保险智能合约的实体接口，完成实体通用数据的初始化，实现员工和账户实体信息上链的功能；

```
//员工身份信息
struct Person {
        //选手填写部分
        // 姓名
        // 年龄
       // 身份证号码
       // 雇主
```

```
            // 缴费开始日期
            // 工资
            // 缴费基数
            //员工地址
        }
       //养老保险账户信息
    struct InsuranceAccount {
             //选手填写部分
            // 个人账户余额
            // 总账户余额
            // 雇主是否为职工的管理员
            //缴费时间
        }
```

（2）基于给定的智能合约代码以及注释，完善员工信息上链的接口。

```
    //构造函数
        function addPerson(string name, uint age, string memory idNumber, string employer, uint startDate, uint salary, address employeeAddress ) public {
            // 选手填写部分
            // 身份证号码不能为空且账户不能已存在

            uint paymentBase= salary > MAX_INSURANCE_BASE ? MAX_INSURANCE_BASE : (salary < MIN_INSURANCE_BASE ? MIN_INSURANCE_BASE : salary);
            // 选手填写部分
            // 创建新账户并存储信息


        }
```

（3）基于给定的智能合约代码以及注释，完善新职工账户上链的接口。

```
    function addEmployee(address employee, uint256 salary, address _sponsor) public {
            // 选手填写部分
    // 只有管理员可以添加新职工账户

            require(employee!= address(0) && insuranceAccounts[employee].personalBalance == 0, "Invalid employee address");
            //选手填写部分
            // 创建新职工账户

            uint256 insuranceBase = salary > MAX_INSURANCE_BASE ? MAX_INSURANCE_BASE : (salary < MIN_INSURANCE_BASE ? MIN_INSURANCE_BASE : salary);
            uint256 personalAmount = insuranceBase * PERSONAL_RATE / 100;
```

```
        uint256 overallAmount = insuranceBase * OVERALL_RATE / 100;

        insuranceAccounts[_sponsor].overallBalance  选手填写部分 overallAmount;
        insuranceAccounts[employee].personalBalance  选手填写部分 personalAmount;
        insuranceAccounts[_sponsor].personalBalance = 选手填写部分;


    }
```

（4）基于给定的智能合约代码以及注释，完善新雇主上链的接口。

```
/********** 添加新雇主账户接口 **********/
    function addSponsorAccount(address _sponsor, string sponsorName) public {
        // 只有管理员可以添加新雇主账户
        require(admin == msg.sender, "Only admin can add new sponsors");
        //选手填写部分
        // 雇主地址不能为空且雇主账户不能已存在
        // 创建新雇主账户
        sponsor[_sponsor] = 选手填写部分;
        sponsors.push(_sponsor);


    }
```

2. 养老保险主合约（PensionInsuranceTransfer）接口编码。

（1）编写养老保险主合约的实体接口，完成实体通用数据的初始化，实现申请人和社保局实体信息上链的功能；

```
    struct Application {
        // 申请人地址
        //申请人当前社保局地址
        // 原城市
        // 目标城市
        // 是否停缴   true === 停止缴费
        // 是否批准
    }
    // 定义社保局结构体
    struct SocialSecurityBureau {
        // 社保局城市
        // 社保局地址
        // 是否允许接收社保转移授权
        // 是否已接收
    }
```

（2）基于给定的智能合约代码以及注释，完善授权功能。

```
        //   需要接收城市社保局被授权
        function applyForTransfer(string memory _fromCity, string memory _toCity,
address applicant,address fromBureauAddress,    address toBureauAddress) public {
            accounts[applicant].personalFund =选手填写部分;
            //选手填写部分
            // 申请人地址不能为空且必须有账户
            // 目标城市社保局必须被授权
            // 创建申请
            currentApplication[applicant] = newApplication;
            applicationAddress.push(选手填写部分);
        }
```

（3）基于给定的智能合约代码以及注释，完善 PensionInsuranceTransfer 主合约社保局查询调用申请函数。

```
function authorizeTransfer(address _applicantAddress, address fromBureauAddress)
public {
            //选手填写部分
            // 判断社保局权限，只有当前社保局可以调用申请
            // 转移账户
            // 申请人账户清零
            // 当前社保局接收通过申请
            // 更新申请状态
        }
```

任务 2-3：按要求完成智能合约的编译、部署与调用

【要求】

登陆 linux 服务器，进入/fisco 目录，基于已部署好的 FISCO BCOS 区块链网络平台，启动 WeBASE-Front 环境，通过浏览器访问 WeBASE-Front 服务。根据上述已完成的智能合约，使用 WeBASE-Front 的合约管理模块完成智能合约的编译、部署与调用。最后将完成的代码结果截图保存至作答区并提交。

【任务】

启动 WeBASE-Front 服务，使用浏览器登陆 WeBASE-Front 管理平台；

正确编译并部署上题中的智能合约,调用相关接口验证智能合约的业务流程。要求如下：

使用 WeBASE-Front 管理平台编译、部署“PensionInsuranceTransfer.sol”合约，获取合约的 abi；

（2)测试 PensionInsuranceTransfer.sol 合约,先调用 addSponsorAccount ()添加雇主账户，再调用 getSponsor ()查看雇主账户信息；

（3）测试 PensionInsuranceTransfer.sol 合约，先调用 depositSposor() 给雇主账户充值；

（4）测试 PensionInsuranceTransfer.sol 合约，调用 addPerson()添加员工，并通过 addEmployee()给员工开设养老保险账户；

（5）测试 PensionInsuranceTransfer.sol 合约，调用 makePayment()给雇员缴纳社保；

（6）测试 PensionInsuranceTransfer.sol 合约，调用 addBureaus()创建社保局，并通过 changeBureausAuthorized()修改社保局状态，再通过 applyForTransfer()完成提交社保转移申请,最后通过 authorizeTransfer()、receiveTransfer()审批申请。

任务 2-4：按要求完成智能合约的功能测试

【要求】

根据上述已完成的智能合约，补充添加雇主账户并验证雇主账户数量、添加个人签章账户以及验证签章账户数量测试用例计划，基于已有的 truffle、ganache-cli 环境，已有的养老保险基础测试项目 PensionInsuranceTransfer，使用 VSCode 开发工具、truffle 工具、JavaScript 语言完善智能合约的部署、测试用例代码编写，并执行测试。最后将完成的代码及测试结果截图保存至作答区并提交。

【任务】

1. 补充添加印章账户并验证印章账户数量、添加个人签章账户以及验证签章账户数量测试用例计划，具体要求如下：

（1）补全以下添加雇主账户操作的测试用例设计，根据测试结果，描述测试结论和添加雇主账户测试功能存在的问题；

测试用例编号	PensionInsuranceTransfer_001
测试项目	养老保险添加雇主账户
测试标题	账户数量不对
重要级别	高

预置条件	/* 选手填写部分*/
输入	/* 选手填写部分*/
操作步骤	①存入 5 个雇主账户；②读取雇主账户数量 ③账户数量为 6 比较
预期输出	/* 选手填写部分*/

（2）补全以下添加养老保险转移操作的测试用例设计，根据测试结果，描述测试结论和添加申请功能存在的问题。

测试用例编号	PensionInsuranceTransfer _002
测试项目	养老保险转移模块中添加申请
测试标题	正常提交申请
重要级别	高
预置条件	/* 选手填写部分*/
输入	/* 选手填写部分*/
操作步骤	①创建 2 个社保局；②创建申请 ③查询申请人列表是否存在
预期输出	/* 选手填写部分*/

2. 在/home/admin1/PensionInsuranceTransfer 目录下完善测试脚本与测试。具体要求如下：

（1）启动 ganache-cli 环境，要求配置监听端口为 7545，网络 ID 为 5777。使用 truffle 完善合约 PensionInsuranceTransfer 的部署脚本并测试 PensionInsuranceTransfer.sol 合约，部署脚本参考如下结构，完善部署脚“1_init_migrations.js”文件，补充“选手填写部分”，并使用正确代码替换；

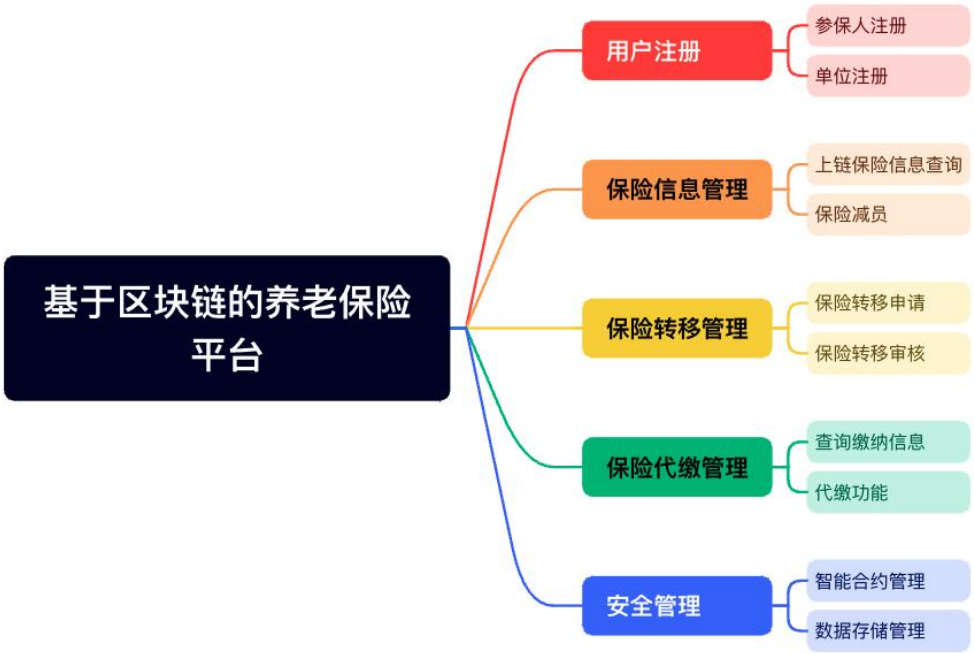
（2）在 test 目录中完善 test.js 文件，补充测试函数测试实现添加 6 个雇主账户以及验证雇主账户数量的函数；

（3）在 test 目录中完善 test.js 文件，完善测试函数测试实现添加转移申请和验证是否正常提交申请的函数。

模块三：区块链应用系统开发（30 分）

养老保险是对于老年人的最基本的生活保障。各种数据显示，当前的养老金市场规模庞大。2016 年美国的养老金资产总额为 19.1 万亿美元；而据估计，截至 2016 年，全球养老金资产总额约为 36 万亿美元。美国是最大的养老保险市场，其占全球份额为 62%；其次是英国(7.9%)和日本(7.7%)。然而，养老保险市场在各个方面存在诸多缺陷，往往会对个人造成负面影响。常见的问题包括治理不善和糟糕的绩效管理，以及在养老金缴纳过程中出现问题（因为没有标准的过程，其流程容易出现隐藏的成本、糟糕的数据管理，甚至是养老金欺诈 ）。

区块链是去中心化信任网络，适合作为此类分布式应用的底层架构和基础工具。本应用基于 FISCO BCOS 开发可信养老保险管理系统，主要涉及用户管理（参保人和公司），保险信息管理、保险转移管理功能模块。系统采用前后端分离架构，前端使用 Vue.js 和 JavaScript，后端开发使用 Java。完成的区块链系统所记录的信息更加真实可靠，可以帮助解决信任危机。



任务 3-1：按要求完成应用后端开发

【要求】

后端系统需要与区块链平台进行交互，并完成复杂的业务逻辑。本题需要根据具体需求完成养老保险管理模块相关智能合约的部署。同时，使用 IntelliJ IDEA 开发工具打开/home/admin1/PensionInsurance/backend 代码，基于后端系统的开发模板，代码文件中的“选手填写部分”注释，补充缺失的代码段，完成后将所有补充的代码截图，保存至作答区并点击提交。

【任务】

1. 基于提供的服务器环境，利用已安装部署好的 FISCO BCOS 区块链底层平台，使用 ADMIN 账户完成/home/admin1/PensionInsurance/contracts 目录下两个智能合约的上传、编译和部署，并获取合约地址；

（1） 启动 FISCO BCOS 区块链平台（/fisco），使用浏览器打开 WeBASE-Front；

（2） 上传 PensionInsurance.sol 和 PensionInsuranceTransfer.sol 智能合约，编译并部署合约；

（3） 获取智能合约的 abi 及合约地址。

2. 使用 Java 语言，完成房屋租赁管理模块的后端功能开发。使用 IntelliJ IDEA 开发工具打开/home/admin1/PensionInsurance/backend 代码，基于后端系统的开发模板，找到**“选手填写部分”**的注释，补充缺失的代码段。

（1）设置应用程序相关配置，填写所有者地址、合约地址、webase-url 地址；

（2）打开 OrgServiceImpl.java 文件，完成注册服务类的业务逻辑；

代码片段 1：

```
/**
 * 注册用户
 * @param personBo
 * @return
 */
@Override
public Result<String> addPerson(PersonBo personBo) {
```

```
        if (StrUtil.isEmpty(personBo.getEmployeeAddress())
        ) {
            return Result.error(ResultVO.PARAM_EMPTY);
        }
        List funcParam = new ArrayList();
        //选手填写部分
        //封装参数

        String _result =  选手填写部分;
        JSONObject obj=选手填写部分;
        String statusOK=obj.get("statusOK").toString();
        if (选手填写部分){
            return Result.error(ResultVO.CONTRACT_ERROR);
        }
        return Result.success("ok");
    }
```

代码片段 2：

```
    /**
     * 注册雇主
     * @param insuranceAccountBo
     * @return
     */
    @Override
    public Result<String> addSponsorAccount(InsuranceAccountBo insuranceAccountBo) {
        if (StrUtil.isEmpty(insuranceAccountBo.getSponsorAddress())
        ) {
            return Result.error(ResultVO.PARAM_EMPTY);
        }
        List funcParam = new ArrayList();
        //选手填写部分

        String _result =选手填写部分
        JSONObject obj=选手填写部分;
        String statusOK=选手填写部分;
        if (选手填写部分){
            return Result.error(ResultVO.CONTRACT_ERROR);
        }
        return Result.success("ok");
    }
```

代码片段 3：

```
    /**
     * 注册雇员账户
     * @param insuranceAccountBo
```

```
     * @return
     */
    @Override
    public Result<String> addEmployee(InsuranceAccountBo insuranceAccountBo) {
        if (StrUtil.isEmpty(insuranceAccountBo.getAddress())
        ) {
            return Result.error(ResultVO.PARAM_EMPTY);
        }
        List funcParam = new ArrayList();
        //选手填写部分

        String _result =选手填写部分
        JSONObject obj=选手填写部分;
        String statusOK=选手填写部分);
        if (选手填写部分){
            return Result.error(ResultVO.CONTRACT_ERROR);
        }
        return Result.success("ok");
    }
```

代码片段 4：

```
/**
     * 注册社保局
     * @param socialSecurityBureauBo
     * @return
     */
    @Override
    public Result<String> addBureaus(SocialSecurityBureauBo socialSecurityBureauBo) {
        if (选手填写部分)
        ) {
            return Result.error(ResultVO.PARAM_EMPTY);
        }
        List funcParam = new ArrayList();
        //选手填写部分

        String _result =选手填写部分
        JSONObject obj=选手填写部分;
        String statusOK=选手填写部分;
        if (选手填写部分){
            return Result.error(ResultVO.CONTRACT_ERROR);
        }
        return Result.success("ok");
    }
```

（3）打开 QueryServiceImpl.java 文件，实现相关查询业务逻辑；

代码片段 1：

```
/**
 * 获取所有的雇员列表
 * @return
 */
@Override
public Result listAllEmployees(String userAddress) {
    //获取所有的职工地址
    String _result = 选手填写部分;
    选手填写部分
    //根据地址获取职工详情
    List<Person> personList = new ArrayList<>();
    for(选手填写部分) {
        Person person =选手填写部分;
        选手填写部分
    }
    return Result.success(personList);
}
```

代码片段 2：

```
/**
 * 根据员工地址查询账户详情
 * @param userAddress
 * @param
 * @return
 */
private InsuranceAccount getInsuranceAccountDetail(String userAddress,String accountAddress){
    List funcParam = new ArrayList();
    //选手填写部分
    InsuranceAccount insuranceAccount = new InsuranceAccount();
    try{
        String _result = 选手填写部分;
        JSONArray employeeArray = JSONUtil.parseArray(_result);
        String _result2 = 选手填写部分;
        JSONArray sponsor = JSONUtil.parseArray(_result2);
        //封装员工信息
        选手填写部分
    }catch (Exception ex){
        System.out.println(ex.getMessage());
    }
    return insuranceAccount;
}
```

（4）打开 TransactionServiceImpl.java 文件，实现相关社保转移交易逻辑；

代码片段 1：

```
/**
 * 雇主账户充值
 * @param insuranceAccountBo
 * @return
 */
@Override
public Result<String> depositSposor(InsuranceAccountBo insuranceAccountBo) {
    if (选手填写部分)
    ) {
        return Result.error(ResultVO.PARAM_EMPTY);
    }
    List funcParam = new ArrayList();
    //调用合约实现充值
    选手填写部分

    JSONObject obj=JSONUtil.parseObj(_result);
    String statusOK=选手填写部分;
    if (选手填写部分){
        return Result.error(ResultVO.CONTRACT_ERROR);
    }
    return Result.success("ok");
}
```

代码片段 2：

```
/**
 * 社保缴费
 * @param insuranceAccountBo
 * @return
 */
@Override
public Result<String> makePayment(InsuranceAccountBo insuranceAccountBo) {

    if (选手填写部分) {
        return Result.error(ResultVO.PARAM_EMPTY);
    }
    List funcParam = new ArrayList();
    选手填写部分

    String _result = 选手填写部分
```

```
        JSONObject obj=选手填写部分;
        String statusOK=选手填写部分;
        if (选手填写部分){
            return Result.error(ResultVO.CONTRACT_ERROR);
        }
        return Result.success("ok");
    }
```

（5）打开 TransactionController.java 文件，实现相关交易接口逻辑；

代码片段 1：

```
@ApiOperation(value = "给雇主充值", notes = "给雇主充值")
@RequestMapping(value = "depositSposor", method = RequestMethod.POST)
public Result<String> depositSposor(选手填写部分) {
    return  选手填写部分;
}
```

代码片段 2：

```
@ApiOperation(value = "缴纳养老保险", notes = "缴纳养老保险")
@RequestMapping(value = "makePayment", method = RequestMethod.POST)
public Result<String> makePayment(选手填写部分) {
    return  选手填写部分;
}
```

代码片段 3：

```
@ApiOperation(value = "申请养老保险转移", notes = "申请养老保险转移")
@RequestMapping(value = "applyForTransfer", method = RequestMethod.POST)
public Result<String> applyForTransfer(选手填写部分) {
    return 选手填写部分;
}
```

代码片段 4：

```
@ApiOperation(value = "社保局审核申请", notes = "fromCity 社保局审核申请")
@RequestMapping(value = "authorizeTransfer", method = RequestMethod.POST)
public Result<String> authorizeTransfer(选手填写部分) {
    return 选手填写部分;
}
```

代码片段 5：

```
@ApiOperation(value = "新的社保局接受申请", notes = "toCity  社保局接收账户转移接口  ")
@RequestMapping(value = "receiveTransfer", method = RequestMethod.POST)
public Result<String> receiveTransfer(选手填写部分) {
    return 选手填写部分;
```

```
}
```

（6）打开 OrgController.java 文件，实现相关注册接口逻辑；

代码片段 1：

```
@ApiOperation(value = "登录", notes = "登录")
@RequestMapping(value = "login", method =选手填写部分)
public Result<String> login(@RequestBody 选手填写部分) {
    String userAddress =选手填写部分;
    if (选手填写部分) {
        return Result.error(ResultVO.PARAM_EMPTY);
    }
  if (选手填写部分){
        return Result.error(new ResultVO(500,"登录失败，账号错误",null));
  }
  return Result.success("ok");
}
```

代码片段 2：

```
@ApiOperation(value = "注册用户", notes = "注册用户")
@RequestMapping(value = "addPerson", method =选手填写部分)
public Result<String> addPerson(@RequestBody 选手填写部分) {
    return  选手填写部分;
}
```

代码片段 3：

```
@ApiOperation(value = "注册雇主", notes = "注册雇主")
@RequestMapping(value = "addSponsorAccount", method =  选手填写部分)
public Result<String> addSponsorAccount(@RequestBody 选手填写部分) {
    return 选手填写部分;
}
```

代码片段 4：

```
@ApiOperation(value = "注册员工", notes = "注册接口")
@RequestMapping(value = "addEmployee", method = RequestMethod.POST)
public Result<String> addEmployee(@RequestBody 选手填写部分) {
    return  选手填写部分;
}
```

代码片段 5：

```
@ApiOperation(value = "注册社保局", notes = "注册接口")
@RequestMapping(value = "addBureaus", method =选手填写部分)
```

```
public Result<String> addBureaus(@RequestBody 选手填写部分) {
    return 选手填写部分;
}
```

任务 3-2：按要求完成应用前端开发

【要求】

使用 Vue.js 和 JavaScript，完成养老保险管理模块的前端功能。使用 VSCode 开发工具打开/home/admin1/PensionInsurance/front 代码，基于前端系统的开发模板，找到 views 目录下 vue 文件中的“选手填写部分”注释，补充缺失的代码段。完成后，将所有补充的代码截图，保存至作答区并点击提交。

【任务】

1.请基于前端系统的开发模板，在登录页面 Login.vue 文件中添加对应的代码逻辑，实现对前端系统的登录功能，并测试功能完整性，示例页面如下；

题目的具体要求如下：

（1）界面有明确的登录相关提示语；

（2）需要填写的项有管理员账户地址；

（3）点击“登录”按钮时需要检查管理员账户地址是否为空；

（4）登录成功后跳转至相应管理页面；

代码片段 1：

```
<template>
  <el-row style="height: 100%;">
    <el-col :span="10" :offset="7" style="border-style: solid;border-color: #e6e6e6;margin-top: 5%">
      <el-row >
        <el-col :span="16" :offset="4">
          <el-form label-width="120px">
```

```
                <h1>养老保险政务管理平台</h1>
                <h3>登录界面</h3>
                <el-form-item label="管理员地址录入:">
                  <el-input type="primary" v-model="选手填写部分"></el-input>
                </el-form-item>
              </el-form>
            </el-col>

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

代码片段 2：

```
methods: {
    login: function () {
      if (选手填写部分 == 选手填写部分) {
        alert("区块链地址不能为空！")
      }else {
        let postData = {
          userAddress: this.address
        }
        this.axios.post('/org/login', postData).then((response) => {
          if (response.data.code == 选手填写部分) {
            this.$cookies.set('senderAddress',选手填写部分)
            alert('登录成功')
            // 跳转雇主信息页面
            this.$router.push('选手填写部分')
          }else {
            alert(`登录失败,'选手填写部分'`)
          }
        })
      }
    }
  }
```

2. 请基于前端系统的开发模板，在雇主公司列表页面 Sponsor. vue 文件中

添加对应的代码逻辑，实现对前端系统的雇主账户管理功能，并测试功能完整性,示例页面如下：

本题目的具体要求如下：

（1）界面有明确的相关雇主账户信息提示语；

（2）页面需要有“申请雇主社保账户”按钮，可以打开申请信息填写对话框；

（3）点击“添加”按钮可以添加雇主信息，并且有相关提示信息；

（4）创建房产成功后刷新列表数据；

（5）针对雇主账户能够进行账户充值操作；

代码片段 1：

```
<el-main style="padding-top: 10px">
            <el-row>
              <el-col :span="23" :offset="1" style="text-align: left">
                <el-button @click="选手填写部分" type="primary">申请雇主社保账户</el-button>
                <el-table :data="companyList" class="content">
                  <el-table-column prop="选手填写部分" label="公司名称" min-width="10%"></el-table-column>
                  <el-table-column prop="选手填写部分" label="公司地址" min-width="20%"></el-table-column>
                  <el-table-column prop="选手填写部分" label="公司账户余额" min-width="10%"></el-table-column>
                  <el-table-column label="操作" min-width="10%">
                    <template slot-scope="选手填写部分">
                      <el-button size="mini" type="primary"
                        @click="depositBalance(选手填写部分)">账户充值</el-button>
                    </template>
                  </el-table-column>
                </el-table>
              </el-col>
            </el-row>
          </el-main>
```

代码片段 2：

```
<el-dialog title="雇主公司信息" :visible.sync="dialogVisible">
        <el-form label-width="100px" ref="companyDetail">
          <el-form-item label="雇主公司地址:">
            <el-input v-model="选手填写部分"></el-input>
          </el-form-item>
          <el-form-item label="雇主公司名称:">
            <el-input v-model="选手填写部分"></el-input>
          </el-form-item>
          <el-form-item>
            <el-button type="primary" @click="onSubmit(选手填写部分)">添加</el-button>
            <el-button @click="closeDialog">取消</el-button>
          </el-form-item>
        </el-form>
      </el-dialog>
```

代码片段 3：

```
<el-dialog title="雇主公司信息" :visible.sync="depositBalanceDialog">
        <el-form label-width="100px" ref="companyDetail">
          <el-form-item label="公司地址:">
            <el-input v-model="选手填写部分"></el-input>
          </el-form-item>
          <el-form-item label="充值金额:">
            <el-input v-model="选手填写部分"></el-input>
          </el-form-item>
          <el-form-item>
            <el-button type="primary" @click="SubmitDeposit(选手填写部分)">充值</el-button>
            <el-button @click="closeDialog">取消</el-button>
          </el-form-item>
        </el-form>
      </el-dialog>
```

代码片段 4：

```
onSubmit: function (from) {
        let postData = {
          选手填写部分
        }
        this.axios
          .post("/org/addSponsorAccount", 选手填写部分)
          .then((response) => {
            console.log(response);
            if (response.data.code == 200) {
              选手填写部分
              this.closeDialog()
```

```
                } else {
                    alert(`请求内容有误,${response.data.msg}`);
                }
            });
        },
```

3. 请基于前端系统的开发模板，在用户账户列表 Person.vue 文件中添加对应的逻辑代码，实现对员工用户管理的相关业务功能，示例页面如下：

具体要求如下：

（1）界面有明确的相关合同信息提示语；

（2）页面需要有“申请用户社保账户”按钮，可以弹出用户信息输入框；

（3）点击“购买保险”按钮时，自动填写表单，需要确认金额是否正确；

（4）创建用户社保账户成功后刷新列表数据。

代码片段 1：

```
  <el-main style="padding-top: 10px">
            <el-row>
              <el-col :span="23" :offset="1" style="text-align: left">
                <el-button @click="dialogVisible = true" type="primary">申请用户社保账户</el-button>
                <el-table :data="personList" class="content">
                  <el-table-column prop="选手填写部分" label="姓名" min-width="10%"></el-table-column>
                  <el-table-column prop="选手填写部分" label="年龄" min-width="10%"></el-table-column>
                  <el-table-column prop="选手填写部分" label="身份证号码" min-width="20%"></el-table-column>
                  <el-table-column prop="选手填写部分" label="雇主" min-width="10%"></el-table-column>
                  <el-table-column prop="选手填写部分" label="缴费基数" min-width="10%"></el-table-column>
                  <el-table-column prop="选手填写部分" label="账户余额" min-width="10%"></el-table-column>
                  <el-table-column label="操作" min-width="10%">
                    <template slot-scope="scope">
```

```
                        <el-button size="mini" type="primary" @click="buyInsurance(选手填写部分)">购买保险</el-button>
                    </template>
                </el-table-column>
            </el-table>
        </el-col>
    </el-row>
</el-main>
```

代码片段 2：

```
<el-dialog title="员工信息" :visible.sync="dialogVisible">
    <el-form label-width="100px" ref="personDetail">
        <el-form-item label="员工姓名:">
            <el-input v-model="选手填写部分"></el-input>
        </el-form-item>
        <el-form-item label="员工年龄:">
            <el-input v-model="选手填写部分"></el-input>
        </el-form-item>
        <el-form-item label="身份证号码:">
            <el-input v-model="选手填写部分"></el-input>
        </el-form-item>
        <el-form-item label="公司名称:">
            <el-input v-model="选手填写部分"></el-input>
        </el-form-item>
        <el-form-item label="员工地址:">
            <el-input v-model="选手填写部分"></el-input>
        </el-form-item>
        <el-form-item label="工资:">
            <el-input v-model="选手填写部分"></el-input>
        </el-form-item>
        <el-form-item label="缴费起始日期:">
            <el-date-picker v-model="选手填写部分" type="date" placeholder="选择日期">
            </el-date-picker>
        </el-form-item>
        <el-form-item>
            <el-button type="primary" @click="onSubmit(选手填写部分)">立即创建</el-button>
            <el-button @click="closeDialog">取消</el-button>
        </el-form-item>
    </el-form>
</el-dialog>
<!-- 购买保险 -->
<el-dialog title="购买保险" :visible.sync="buyDialog">
    <el-form label-width="100px" ref="personDetail">
```

```
                        <el-form-item label="员工地址:">
                          <el-input v-model="选手填写部分"></el-input>
                        </el-form-item>
                        <el-form-item label="工资:">
                          <el-input v-model="选手填写部分"></el-input>
                        </el-form-item>
                        <el-form-item label="公司名称:">
                          <el-input v-model="选手填写部分"></el-input>
                        </el-form-item>
                        <el-form-item>
                          <el-button type="primary" @click="buySubmit(选手填写部分)">确认购买
</el-button>
                          <el-button @click="closeDialog">取消</el-button>
                        </el-form-item>
                      </el-form>
                    </el-dialog>
```

4.请基于前端系统的开发模板，在用户社保转移申请组件Application.vue文件中添加对应的逻辑代码，实现对后端系统的用户提交申请及审批业务功能，示例页面如下：

具体要求如下：

（1）界面有明确的相关合同信息提示语；

（2）页面需要有“停缴”按钮，可以实现对申请停止缴费功能；

（3）页面需要有“批准”按钮，可以实现对申请批准功能；

（4）点击“添加社保转移申请”按钮后给出相应的操作结果提示，刷新列表。

代码片段1：

```
<el-main style="padding-top: 10px">
          <el-row>
            <el-col :span="20" :offset="2" style="text-align: left">
              <el-button @click="dialogVisible = true" type="primary">添加社保转移申请</el-button>
              <el-table :data="applicationList">
                <el-table-column prop="选手填写部分" label="申请人地址"
```

```
min-width="120px"></el-table-column>
                    <el-table-column prop="选手填写部分" label="原城市"></el-table-column>
                    <el-table-column prop="选手填写部分" label="转入城市"></el-table-column>
                    <el-table-column prop="选手填写部分" label="是否停缴">
                    </el-table-column>
                    <el-table-column prop="选手填写部分" label="通过申请">
                    </el-table-column>
                    <el-table-column label="操作">
                      <template slot-scope="scope">
                        <el-button type="primary" @click="onDialogStopped(选手填写部分)">停缴
</el-button>

                        <el-button type="primary" @click="onDialogApproved(选手填写部分)">批准
</el-button>
                      </template>
                    </el-table-column>
                  </el-table>
                </el-col>
              </el-row>
              <el-row>
                <!-- 添加申请 -->
                <el-dialog title="申请信息" :visible.sync="dialogVisible">
                  <el-form label-width="100px" ref="applicentDetail">
                    <el-form-item label="申请人地址:">
                      <el-input v-model="选手填写部分"></el-input>
                    </el-form-item>
                    <el-form-item label="当前社保局地址:">
                      <el-input v-model="选手填写部分"></el-input>
                    </el-form-item>
                    <el-form-item label="当前城市:">
                      <!-- <el-input v-model="选手填写部分"></el-input> -->
                      <el-select v-model="选手填写部分" placeholder="请选择转出省份">
                        <el-option v-for="item in provinces" :label="选手填写部分" :value="选手填写部分"
                          :key="item.code"></el-option>

                      </el-select>
                    </el-form-item>
                    <el-form-item label="转入社保局地址:">
                      <el-input v-model="选手填写部分"></el-input>
                    </el-form-item>
                    <el-form-item label="转入城市:">
                      <el-select v-model="选手填写部分" placeholder="请选择转入省份">
                        <el-option v-for="选手填写部分" :label="选手填写部分" :value="选手填写部分"
                          :key="item.code"></el-option>
```

```
            </el-select>
          </el-form-item>

          <el-form-item>
            <el-button type="primary" @click="onSubmit(选手填写部分)">立即创建</el-button>
            <el-button @click="选手填写部分">取消</el-button>
          </el-form-item>
        </el-form>
      </el-dialog>
```

5. 请基于前端系统的开发模板，在社保局列表组件 Bureaus.vue 文件中添加对应的页面代码逻辑，实现对前端系统的社保局信息查看功能，并测试功能完整性，示例页面如下：

具体要求如下：

（1）界面有明确的相关合同信息提示语；

（2）页面需要有“社保局注册”按钮，可以实现社保户添加注册功能，操作之后更新合同列表数据。

代码片段 1：

```
<el-main style="padding-top: 10px">
        <el-row>
          <el-col :span="20" :offset="2" style="text-align: left">
            <el-button @click="选手填写部分" type="primary">社保局注册</el-button>
            <el-table :data="bureausList">
              <el-table-column min-width="20%"></el-table-column>
              <el-table-column prop="选手填写部分" label="社保局"></el-table-column>
              <el-table-column prop="选手填写部分" label="社保局地址"></el-table-column>
            </el-table>
          </el-col>
        </el-row>
        <el-row>
          <el-dialog title="申请信息" :visible.sync="dialogVisible">
            <el-form label-width="100px" ref="bureausDetail">
```

```
                    <el-form-item label="行政区域:">
                      <!-- <el-input v-model="选手填写部分"></el-input> -->
                      <el-select v-model="选手填写部分" placeholder="请选择注册省份">
                        <el-option v-for="item in provinces" :label="选手填写部分" :value="选手填写部分"
                          :key="选手填写部分"></el-option>
                      </el-select>
                    </el-form-item>
                    <el-form-item label="社保局地址:">
                      <el-input v-model="选手填写部分"></el-input>
                    </el-form-item>

                    <el-form-item>
                      <el-button type="primary" @click="onSubmit(选手填写部分)">立即创建</el-button>
                      <el-button @click="closeDialog">取消</el-button>
                    </el-form-item>
                  </el-form>
                </el-dialog>
              </el-row>
            </el-main>
```

代码片段 2：

```
onSubmit: function (from) {
      let postData = {
        选手填写部分
      }
      console.log(postData);
      this.axios
        .post("/org/addBureaus", postData)
        .then((response) => {
          console.log(response);
          if (response.data.code == 200) {
            选手填写部分 = 选手填写部分;

            this.closeDialog()
            选手填写部分
          } else {
            alert(`请求内容有误, ${response.data.msg}`);
          }
        });
    },
    closeDialog: function () {
      (选手填写部分),
        (选手填写部分),
        (选手填写部分);
    },
```

任务 3-3：按要求完成区块链应用联调测试

【要求】

运行区块链应用的前后端程序，测试养老保险管理模块的相关功能。

【任务】

1. 测试养老保险管理模块相关功能。

（1）以合约所有者账户登录，分别测试创建雇主、员工、给员工开设账户、缴纳社保等操作。

（2）以合约所有者角色登录，分别测试养老保险转移，社保局审批等相关功能。