2023年全国职业院校技能大赛

高职组

“区块链技术应用”

赛项赛卷（10卷）

任

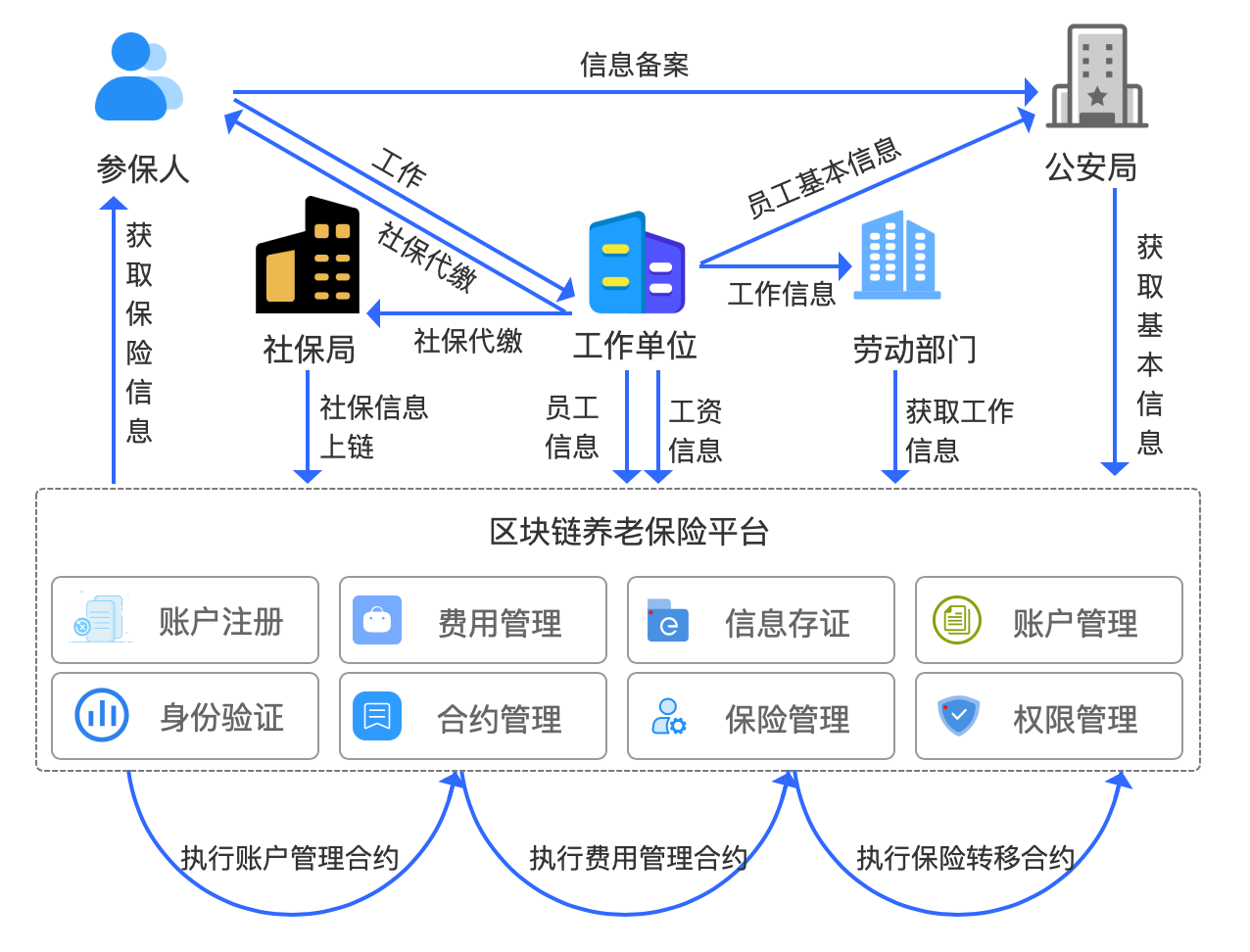
务

书

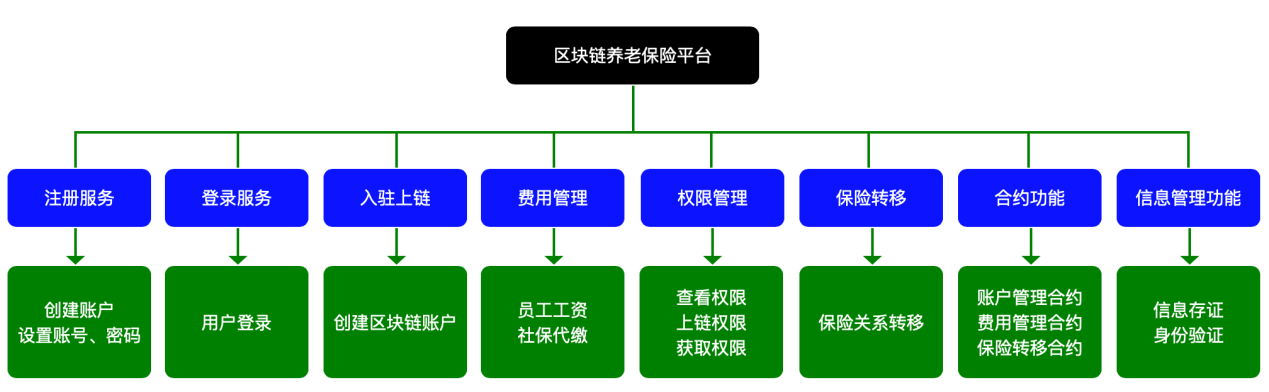
参赛队编号：

**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：区块链产品方案设计及系统运维（35分）

选手完成本模块的任务后，将任务中设计结果、运行代码、运行结果等截图粘贴至客户端桌面【区块链技术应用赛\重命名为工位号\模块一提交结果.docx】中对应的任务序号下。

### 任务1-1：区块链产品需求分析与方案设计

养老保险平台中涉及到参保人、社保局、公安局、工作单位等参与方，他们需要在区块链养老保险平台中完成账户注册、身份上链、社保代缴、信息核查等多种业务活动。通过对业务活动的功能分析，可以更好的服务系统的开发流程。基于养老保险平台系统架构，以区块链养老保险平台为背景，结合账户注册、登录服务、入驻上链、保险转移、社保代缴等核心功能描述，撰写流程图/功能图、用例图等概要设计。

本任务需要依据项目背景完成需求分析与方案设计，具体要求如下:

1. 依据给定区块链养老保险平台的业务流程图以及用例表，编制系统业务用例图，用例图中包含系统参与角色以及用例。

表1-1-1 用例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **用例编号** | **用例标题** | **优先级** | **测试步骤** | **预期结果** | **测试结果** |
| HK-TEST-01 | 工作单位在区块链养老保险平台上参保 | 高 | 在参保界面，工作单位可以输入员工的身份证号码，为员工办理参保 | 申请成功 | 申请成功 |
| HK-TEST-02 | 区块链养老保险平台生成保险账户 | 高 | 区块链养老保险平台自动拉取参保人的基本信息、工作信息等进行核对，核对通过后，在保险账户界面为该参保人成功创建养老保险账户 | 创建成功 | 创建成功 |
| HK-TEST-03 | 工作单位在区块链养老保险平台进行减员 | 高 | 工作单位在保险缴纳页面，点击减员，弹窗确认员工信息，确认无误后减员成功 | 减员成功 | 减员成功 |
| HK-TEST-04 | 工作单位在区块链养老保险平台进行养老保险缴纳 | 高 | 在缴费界面，工作单位确认员工工资，点击缴费，系统判定工资与缴费基数的关系，确定缴费额度，由工作单位确认缴纳 | 缴纳成功 | 缴纳成功 |
| HK-TEST-05 | 社保局在区块链养老保险平台进行保险关系转移 | 高 | 在保险管理界面，展示当前参保人的保险数据，点击转出，判断社保局的身份和权限，将参保人的保险关系转移出去 | 转移成功 | 转移成功 |
| HK-TEST-06 | 社保局在区块链养老保险平台进行保险关系接收 | 高 | 在保险管理界面，展示当前参保人的转入申请数据，点击接收，为该参保人创建参保档案，将参保人的保险关系进行接收 | 获取申请信息  参保档案创建成功  接收成功 | 获取申请信息  参保档案创建成功  接收成功 |

2. 依据给定的背景信息、区块链养老保险业务流程图以及区块链养老保险平台架构图以及给出的养老保险业务的核心流程，使用思维导图工具编制业务系统功能图；

表1-1-2 养老保险平台的核心流程

|  |  |
| --- | --- |
| 缴纳养老保险流程 | 区块链养老保险平台根据缴费标准生成缴费协议，对协议合约进行平台签名，将协议代码广播到区块链。获取参保人工资数据，判断费用额度。 |
| 保险关系转移流程 | 区块链养老保险平台将参保人身份进行验证、参保关系、停缴状态进行核查，符合则进入保险关系转移流程，向参保人保险账户、账户基金等进行转移 |

3. 按照基础层、合约层、接口层以及应用层来设计区块链系统的架构，画出系统架构图，其中在基础层需指明需要的节点、名称、协议、存储等信息；

4. 结合案例背景将区块链养老保险平台核心功能进行划分，完成下方表格中各个主要模块要实现的功能；

在空白处填写主要模块要实现的功能：

|  |  |
| --- | --- |
| 参保模块 | 请输入要实现的功能 |
| 缴费模块 | 请输入要实现的功能 |
| 保险转移模块 | 请输入要实现的功能 |
| 合约功能模块 | 请输入要实现的功能 |

5. 根据用例设计以及下方提供好的承诺集，使用Visio工具完成区块链应用系统业务流程图；

表1-1-3 承诺集

|  |  |
| --- | --- |
| 参保承诺 | 判断工作单位是否在平台输入正确的身份信息完成参保 |
| 生成保险账户承诺 | 判断参保人基本信息、工作信息是否符合缴费基数规定，是则生成养老保险账户，否则结束 |
| 减员承诺 | 判断参保人是否存在多家缴费单位 |
| 缴费超时承诺 | 判断工作单位是否在规定时间内未参保人缴纳养老保险费用 |
| 缴费承诺 | 判断参保人的工资是否超出了缴纳金额的上下限 |
| 转移保险关系承诺 | 判断参保人是否办理了停缴 |

整合所有内容，模块调用接口编写并形成对应的需求用例文档至【区块链技术应用赛\重命名为工位号】下。

### 任务1-2：区块链系统部署与运维

**子任务1-2-1：区块链网络环境搭建**

操作任务如下：

（1）打开区块链源码文件，检查区块链网络基础运行环境，通过输入命令，检查Docker环境，并确定Docker版本；

（2）打开区块链源码文件，通过输入命令，make编译区块链网络，生成区块链网络编译执行文件，请把编译执行文件截图；

（3）成功编译区块链网络后，通过输入命令，启动区块链服务;

（4）启动区块链网络后，通过输入命令，检查区块链网络状态。

**子任务1-2-2：区块链运维管理**

（1）通过命令行方式，使用Docker启动管理台，启动成功后，使用IP在浏览器中，浏览管理台系统；

（2）使用超级管理员账号（admin）登录，申请链账户的证书文件；

（3）快速生成证书按钮，系统讲快速自动生成四个组织证书、四个节点证书、四个用户证书；

（4）查看链参数配置基础信息、共识信息、节点信息。

**子任务1-2-3：区块链浏览器部署合约**

（1）进行区块链管理，部署智能合约；

（2）发起上链操作；

（3）调用成功后，查看交易详情。

**子任务1-2-4：搭建区块链网络监控系统**

请按如下步骤，完成对系统的搭建，具体操作任务如下：

（1）通过命令行方式，在节点 env.yaml 配置中打开 metricSwitch；

（2）配置每个 target 对应区块链节点的 server.yaml 中的metricPort端口；

（3）进入到对应目录下，通过命令行启动启动prometheus，打开区块链监控系统项目网址。

### 任务1-3：区块链系统测试

（1）从源代码构建Hyperbench，并开启Hyperbench网络；

（2）在区块链上运行HyperBench，并开启区块链网络；

（3）通过SQL手工注入方法分析区块链系统漏洞，寻找环境注入点，判断注入类型并获取数据库中的信息。

## 模块二：智能合约开发与测试（30分）

选手完成本模块的任务后，将任务中设计结果、运行代码、运行结果等截图粘贴至客户端桌面【区块链技术应用赛\重命名为工位号\模块二提交结果.docx】中对应的任务序号下。

### 任务2-1：智能合约设计

根据职工养老保险系统需求用例文档，设计合约接口，画出各需求用例的时序图。

### 任务2-2：智能合约开发

使用Solidity语言进行智能合约开发，根据需求用例文档在待补充源码中完成程序接口功能的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行合约进行业务功能的验证，成功获取合约的abi，将合约部署至区块链，获取部署的合约信息，将任务中编写代码、运行截图、部署至区块链截图保存至指定位置。

**子任务2-2-1：账户管理合约编码**

根据需求用例文档在待补充源码中完成账户管理合约的编码，解决代码错误和警告，正确编译合约，功能调试正确。需要编写生成账户接口，完成从外部部门检索姓名、年龄、雇主、开始日期、工资、缴费基数，将人员信息进行综合存储功能。

待补充源码：

contract PensionAccount {

// 定义用于存储人员信息的结构体

struct Person {

string name; // 姓名

uint age; // 年龄

string idNumber; // 身份证号码

string employer; // 雇主

uint startDate; // 开始日期

uint salary; // 工资

uint paymentBase; // 缴费基数

}

// 定义映射，使用身份证号码作为键存储人员信息

mapping(string => Person) public people;

/\*\*\*\*\*\*\*\*\*\*\* 生成账户接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 生成账户接口开发 \*\*\*\*\*\*\*\*\*\*\*/

// 从外部部门检索信息的函数

function retrieveName(string memory) private pure returns (string memory) {

// 从公安部门检索姓名的代码

return "John Doe";

}

function retrieveAge(string memory) private pure returns (uint) {

// 从公安部门检索年龄的代码

return 35;

}

function retrieveEmployer(string memory) private pure returns (string memory) {

// 从劳动部门检索雇主的代码

return "ABC Company";

}

function retrieveStartDate(string memory) private pure returns (uint) {

// 从劳动部门检索开始日期的代码

return 1625097600; // 2021年6月30日

}

function retrieveSalary(string memory) private pure returns (uint) {

// 从劳动部门检索工资的代码

return 5000;

}

function retrievePaymentBase() private pure returns (uint) {

// 从社会保障局检索缴费基数的代码

return 5000;

}

}

**子任务2-2-2：费用管理合约编码**

根据需求用例文档在待补充源码中完成费用管理合约的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行添加新职工账户、添加新雇主账户。

1．编写添加新职工账户接口，实现当账户不存在，只有管理员可以添加职工账户，职工新账户中账户余额为0，未被赞助的功能；

2．编写添加新雇主账户接口，实现当账户不存在，只有管理员可以添加雇主账户，雇主新账户中账户余额为0，已被赞助的功能；

待补充源码：

contract PensionInsurance {

struct Account {

uint256 personalBalance; // 个人账户余额

uint256 overallBalance; // 总账户余额

bool isSponsor; // 雇主是否为职工的管理员

uint paymentTimestamp;//缴费时间

}

// 定义合约的变量

uint256 constant MAX\_INSURANCE\_BASE = 24000; // 最大保险基数

uint256 constant MIN\_INSURANCE\_BASE = 3200; // 最小保险基数

uint256 constant PERSONAL\_RATE = 8; // 个人缴费率

uint256 constant OVERALL\_RATE = 19; // 总缴费率

uint256 constant SPONSOR\_RATE = OVERALL\_RATE - PERSONAL\_RATE; // 雇主缴费率

uint256 constant PERSONAL\_AMOUNT = 2000; // 每月个人缴费金额

uint256 constant OVERALL\_AMOUNT = 4750; // 每月总缴费金额

uint256 constant PAYMENT\_DEADLINE = 30 days; // 缴费截止日期

address public sponsor = msg.sender; // 管理员的地址

// 定义账户的映射

mapping(address => Account) public accounts;

/\*\*\*\*\*\*\*\*\*\*\* 添加新职工账户接口 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 添加新职工账户接口 \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* 添加新雇主账户接口 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 添加新雇主账户接口 \*\*\*\*\*\*\*\*\*\*\*/

// 定义设置管理员的函数

function setSponsor(address newSponsor) public {

// 只有当前管理员可以设置新的管理员

require(msg.sender == sponsor, "1");

// 新的管理员不能与当前管理员相同

require(newSponsor != sponsor, "1");

// 新的管理员必须有雇主账户

require(accounts[newSponsor].overallBalance > 0, "1");

// 设置新的管理员并更新账户的管理员状态

address oldSponsor = sponsor;

sponsor = newSponsor;

accounts[oldSponsor].isSponsor = false;

accounts[newSponsor].isSponsor = true;

}

// 定义进行缴费的函数

function makePayment(address employee) public {

// 职工账户必须存在

require(accounts[employee].personalBalance > 0 || accounts[employee].overallBalance > 0, "1");

// 雇主账户必须存在且已被赞助

require(accounts[sponsor].overallBalance > 0 && accounts[sponsor].isSponsor, "1");

// 雇主账户必须有足够的余额进行总缴费

require(accounts[sponsor].overallBalance >= OVERALL\_AMOUNT, "1");

// 缴费必须在截止日期之前进行

require(block.timestamp <= accounts[sponsor].paymentTimestamp + PAYMENT\_DEADLINE, "1");

// 根据职工的工资计算个人和总缴费金额

uint256 salary = 25000; // 职工的月薪

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;

// 更新职工和雇主账户的余额

accounts[employee].personalBalance += personalAmount;

accounts[sponsor].personalBalance += personalAmount;

accounts[sponsor].overallBalance -= overallAmount;

accounts[sponsor].paymentTimestamp = block.timestamp;

}

// 定义删除职工账户的函数

function removeEmployee(address employee) public {

// 只有管理员可以删除职工账户

require(msg.sender == sponsor, "1");

// 职工账户必须存在且未被赞助

require(accounts[employee].personalBalance > 0 && !accounts[employee].isSponsor, "2");

// 职工账户必须余额为零

require(accounts[employee].personalBalance == 0 && accounts[employee].overallBalance == 0, "3");

// 删除职工账户

delete accounts[employee];

}

}

**子任务2-2-3：保险转移合约编码**

根据需求用例文档在待补充源码中完成保险转移合约的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行合约中的申请转移关系、接收账户转移接口。

1．编写申请转移关系接口，实现创建申请、添加到申请列表功能，其中创建申请需要设置申请人地址、原城市、目标城市、停缴状态、批准状态；

2．编写接收账户转移接口，实现获取账户，进行账户授权状态、接收状态、个人账户基金、统筹账户基金、养老保险账户的信息设置；

待补充源码：

contract PensionInsurance {

// 定义申请结构体

struct Application {

address applicant; // 申请人地址

string fromCity; // 原城市

string toCity; // 目标城市

bool isStopped; // 是否停缴

bool isApproved; // 是否批准

}

// 定义账户结构体

struct Account {

uint personalFund; // 个人账户基金

uint overallFund; // 统筹账户基金

uint pensionInsurance; // 养老保险账户

}

// 定义社保局结构体

struct SocialSecurityBureau {

address bureauAddress; // 社保局地址

bool isAuthorized; // 是否被授权

bool hasReceived; // 是否已接收

}

// 定义申请列表

Application[] public applications;

// 定义账户映射

mapping(address => Account) public accounts;

// 定义社保局映射

mapping(address => SocialSecurityBureau) public socialSecurityBureaus;

/\*\*\*\*\*\*\*\*\*\*\* 申请转移关系接口 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 申请转移关系接口 \*\*\*\*\*\*\*\*\*\*\*/

// 社保局调用申请

function authorizeTransfer(address applicantAddress, bool isStopped) public {

// 获取申请

Application memory currentApplication;

for (uint i = 0; i < applications.length; i++) {

if (applications[i].applicant == applicantAddress) {

currentApplication = applications[i];

break;

}

}

// 判断是否停缴

if (isStopped) {

currentApplication.isApproved = false;

return;

}

// 转移账户

Account storage currentAccount = accounts[applicantAddress];

SocialSecurityBureau storage currentBureau = socialSecurityBureaus[msg.sender];

currentAccount.personalFund = 0;

currentAccount.overallFund = 0;

currentAccount.pensionInsurance = 0;

currentBureau.hasReceived = true;

// 更新申请状态

currentApplication.isStopped = false;

currentApplication.isApproved = true;

}

/\*\*\*\*\*\*\*\*\*\*\* 接收账户转移接口 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 接收账户转移接口 \*\*\*\*\*\*\*\*\*\*\*/

}

**子任务2-2-4：合约部署和调用**

1．解决代码错误和警告，正确编译合约，成功获取三个合约的abi；

2．将三个合约形成一个文件部署至区块链，获取部署合约的地址信息。

### 任务2-3：智能合约测试

写功能测试用例，实现费用管理功能的业务测试，使用性能测试工具，对申请转移账户接口进行性能测试。

1.根据需求用例文档在以下测试用例表格中编写费用管理合约中的测试用例，依据费用管理中缴费单位的个数、与单位缴费、个人缴费的三种状态，分析在前置条件为参保人已参保、已生成养老账户的情况下，参保人有多个工作单位、有固定单位为其缴费、与工作单位不按时缴费时，不同缴费时间大于、小于规定缴费时间的情况下，产生的养老金账户和养老保险管理平台的归属结果。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 用例编号 | 用例标题 | 优先级 | 前置条件 | 测试步骤 | 测试数据 | 预期结果 |
| PI-AM001 | 费用管理（缴费单位个数） | 高 | 请填写 | 请填写 | 请填写 | 请填写 |
| 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 |
| 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 |
| 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 |

2.利用测试工具，模拟测试申请账户转移接口发送100次请求，将所需HTTP请求、协议、服务器名称或IP以及路径填到工具Web服务器中，连同通请求一起发送的参数，运行成功后将汇总报告提交到指定位置。

## 模块三：区块链应用系统开发（30分）

选手完成本模块的任务后，将任务中设计结果、运行代码、运行结果等截图粘贴至客户端桌面【区块链技术应用赛\重命名为工位号\模块三提交结果.docx】中对应的任务序号下。

### 任务3-1：区块链应用前端功能开发

完成区块链应用系统中的职工养老保险申请页面渲染，要求如下：

（1）按照生成养老保险账户原型图的长度、宽度、行高、间距、文字样式、颜色等，完成订单详情页面的样式开发，将结果截图保存；

（2）完成Vue调用添加养老保险账户API，将养老保险账户信息提交至API接口中，设计图截图保存。



养老保险账户页面的样式开发补充源码：

|  |
| --- |
| .content{  width: 100%;  height: 100%;  background: #EEEEEE;  .contentModel{  position: absolute;  top: 50%;  left: 50%;  box-sizing: border-box;  width: 600px;  padding: 41px 40px 40px;  background: #fff;  border-radius: 8px;  box-shadow: 0 3px 12px 0 #c0c4cc;  transform: translate(-50%, -50%);  -ms-transform: translate(-50%, -50%);  -moz-transform: translate(-50%, -50%);  -webkit-transform: translate(-50%, -50%);  -o-transform:translate(-50%, -50%);  .payment\_title{  font-size: 21px;  font-weight: normal;  text-align: center;  margin-bottom: 20px;  }  //此处代码补全:按原型图样式，进行长度、宽度、行高、间距、文字样式、颜色等样式设置；  }  } |

养老保险账户页面的Html开发补充源码：

|  |
| --- |
| <template>  <div class="content">  <div class='contentModel'>  <h3 class='payment\_title'>生成养老保险账户</h3>  <!-- 此处代码补全:按照原型图格式，画出参保人姓名、参保人省份证、参保人工作单位、工作时长、工资、缴费基数信息并提交 -->  </div>  </div>  </template> |

### 任务3-2：区块链应用后端功能开发

**子任务3-2-1： 获取区块链交易信息**

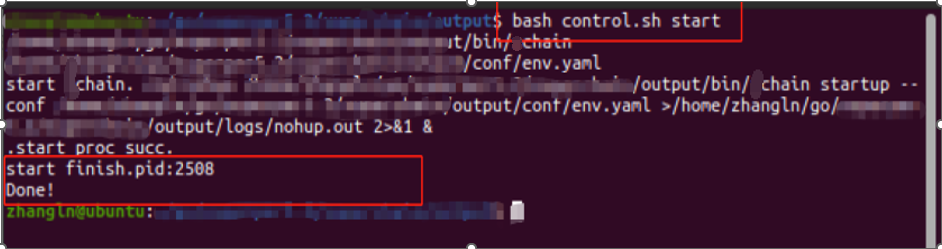
区块链应用系统开发需要区块链底层网络进行支撑，完成和链上数据进行交互，并完成链上信息查询。

要求如下：

（1）打开区块链网络文件目录，通过区块链启动脚本、区块链网络；

（2）打开（src/controller/BlockController.java）文件，查询区块链信息接口。使用Java-SDK获取区块链的最新高度和最新交易Hash，并将结果按十进制的整数和字符串类型返回；

启动区块链源码截图：



区块链的最新高度和最新交易Hash接口补充源码：

|  |
| --- |
| */\*\*获取最新区块高度和最新交易Hash\*/*  @RequestMapping(value = **"/getBlockInformation"**)  @ResponseBody  **public** Map<String,Object> getBlockInformation(FlightInformation flightInformation){  Map<String,Object> resMap = **new** HashMap<>();  **try**{    *// 获取最新区块高度*  **long** blockHeight = block.getLong(**"height"**);  System.***out***.println(**"最新区块高度："** + blockHeight);  *// 获取最新区块交易Hash*  String txHashes = block.getJSONArray(**"txHashes"**);  System.***out***.println(**"最新区块交易Hash："** + txHash);    resMap.put(**"code"**,200);  resMap.put(**"message"**,**"请求成功"**);  resMap.put(**"data"**,**null**);  } **catch** (ClientProtocolException e) {  log.error(**"操作失败—log："**,e);  resMap.put(**"code"**,500);  resMap.put(**"message"**,**"请求失败"**);  } **catch** (IOException e) {  log.error(**"操作失败—log："**,e);  resMap.put(**"code"**,500);  resMap.put(**"message"**,**"请求失败"**);  }**catch** (Exception e){  log.error(**"操作失败—log："**,e);  resMap.put(**"code"**,500);  resMap.put(**"message"**,**"请求失败"**);  }  **return** resMap;  } |

**子任务3-2-2： 声明生成养老保险账户实体类和数据库设计**

使用Java语言，完成区块链系统数据的存储。要求如下：

（1）根据“账户管理合约”中变量的字段，在Java项目中声明实体类（PensionAccount），将声明代码结果截图保存。（2分）

包含字段：钱包地址（walletAddress）、个人账户余额（personalBalance）、总账户余额（overallBalance）、雇主是否为职工的赞助商（isSponsor）、缴费时间（paymentTimestamp）、参保人姓名（insuredPersonName）、参保人身份证（insuredPersonIdentityCard）、参保人工作单位（insuredPersonWorkUnit）、工作年限（workingYear）、薪资（salary）、缴费基数（paymentBase）

（2）请打开Mysql命令行，连接数据库，根据第1步声明的实体类，使用Mysql命令创建数据库表（pension\_account）与表结构字段。

签署房屋租赁合同合约实体字段补充源码：

|  |
| --- |
| */\*\**  *\* 养老账户实体类*  *\*/*  **public class** PensionAccount {  **private static final long *serialVersionUID*** = 1L;  //在此处进行代码补全，声明养老账户实体字段，并添加Get和Set方法  } |

创建pension\_account表补充源码：

|  |
| --- |
| **CREATE** **TABLE** `pension\_account` (  `id` **INT** **NOT** **NULL** **AUTO\_INCREMENT** **COMMENT** '主键id',  //在此处进行建表sql语句补全或使用工具创建表字段  `create\_time` **DATETIME** **NOT** **NULL** **COMMENT** '创建时间',  `update\_time` **DATETIME** **NOT** **NULL** **COMMENT** '修改时间',  **PRIMARY** **KEY** (`id`) **USING** **BTREE**  )  **COLLATE**='utf8\_general\_ci'  **ENGINE**=**InnoDB**  **ROW\_FORMAT**=**COMPACT**  ; |

**子任务3-2-3： 编写调用账户管理合约添加养老保险账户接口**

请使用Java语言，通过Java-SDK完成调用账户管理合约接口，添加养老保险账户。要求如下：

（1）接收从Web端对应各种参数（包括参保人姓名、身份证、工作单位、工作年限、工资、缴费基数）将代码截图保存；

（2）调用Java-SDK，运行调用智能合约API，接收养老保险账户的钱包地址、账户余额等信息传递给前端页面，将代码截图保存；

（3）养老保险账户创建成功后，将获取到的钱包信息进行解析，并通过数据库依赖包（mysql-connector-java-bin.jar）存储到数据库中。

|  |
| --- |
| */\*\*添加养老保险账户\*/*  @RequestMapping(value = **"/addPensionAccount"**)  @ResponseBody  //此处代码补全:接收从Web端传输的参数（包括参保人姓名、身份证、工作单位、工作年限、工资、缴费基数）  **public** Map<String,Object> **addPensionAccount** (…){  Map<String,Object> resMap = **new** HashMap<>();  **try**{  //此处代码补全:调用Java-SDK，运行调用智能合约API，创建养老保险账户的钱包地址、账户余额传递给前端页面  //此处代码补全：创建养老保险账户成功后，将获取到的钱包信息进行解析，并通过数据库依赖包（mysql-connector-java-bin.jar）存储到数据库中  resMap.put(**"code"**,200);  resMap.put(**"message"**,**"请求成功"**);  resMap.put(**"data"**,depositInfo);  } **catch** (Exception e){  log.error(**"操作失败—log："**,e);  resMap.put(**"code"**,200);  resMap.put(**"message"**,**"请求失败"**);  }  **return** resMap;  } |