**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **《区块链技术与应用》** |
| **学生姓名：** | **石夏源** |
| **学生学号：** | **202230091036** |
| **学生专业：** | **软件工程** |
| **授课教师：** | **许可** |
| **开课学期：** | **2024-2025-1** |

**华南理工大学软件学院**

**2024 年 9月**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验地点：** | B7后座 1楼 | 138房 | | |
| **实验日期与时间：** | 2024年9月28日1-4节 | | | |
| **实验教师：** | 郭 锐、许可 | |  |  |
| **总体批改意见：** |  | | | |

**实验得分：**

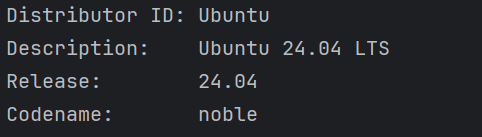
|  |  |  |  |
| --- | --- | --- | --- |
| 实验1评分 | 实验2评分 | 实验3评分 | 总分 |
| /40 | /40 | /20 | /100 |

(\*\*\*\*\*\*注意：本页内容学生请勿改动\*\*\*\*\*\*\*\*)

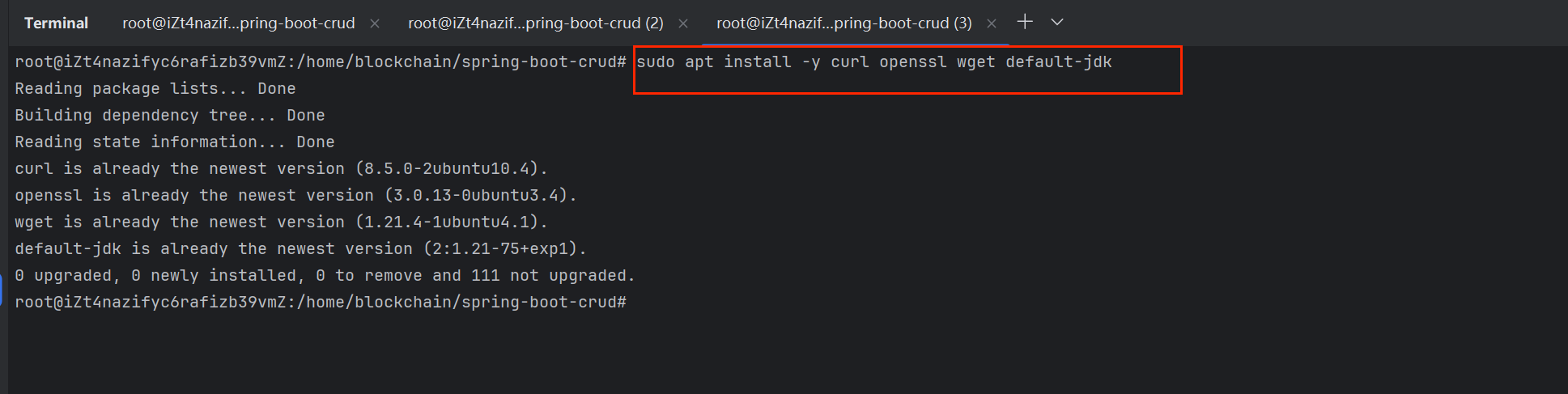
**实验一 区块链理解与实践（40分）**

**准备实验环境（8分）**

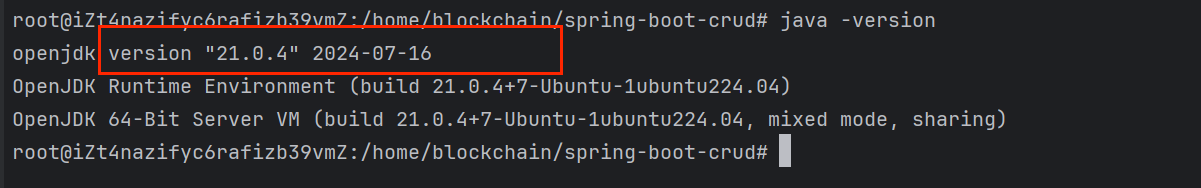
1. 你实验用的操作系统是 Ubuntu 24.04 LTS 。（2分）



1. 你的实验环境中是否已经安装必须使用的依赖（如curl、openssl、wget、openjdk8）？若无，请给出你的安装步骤。（2分）

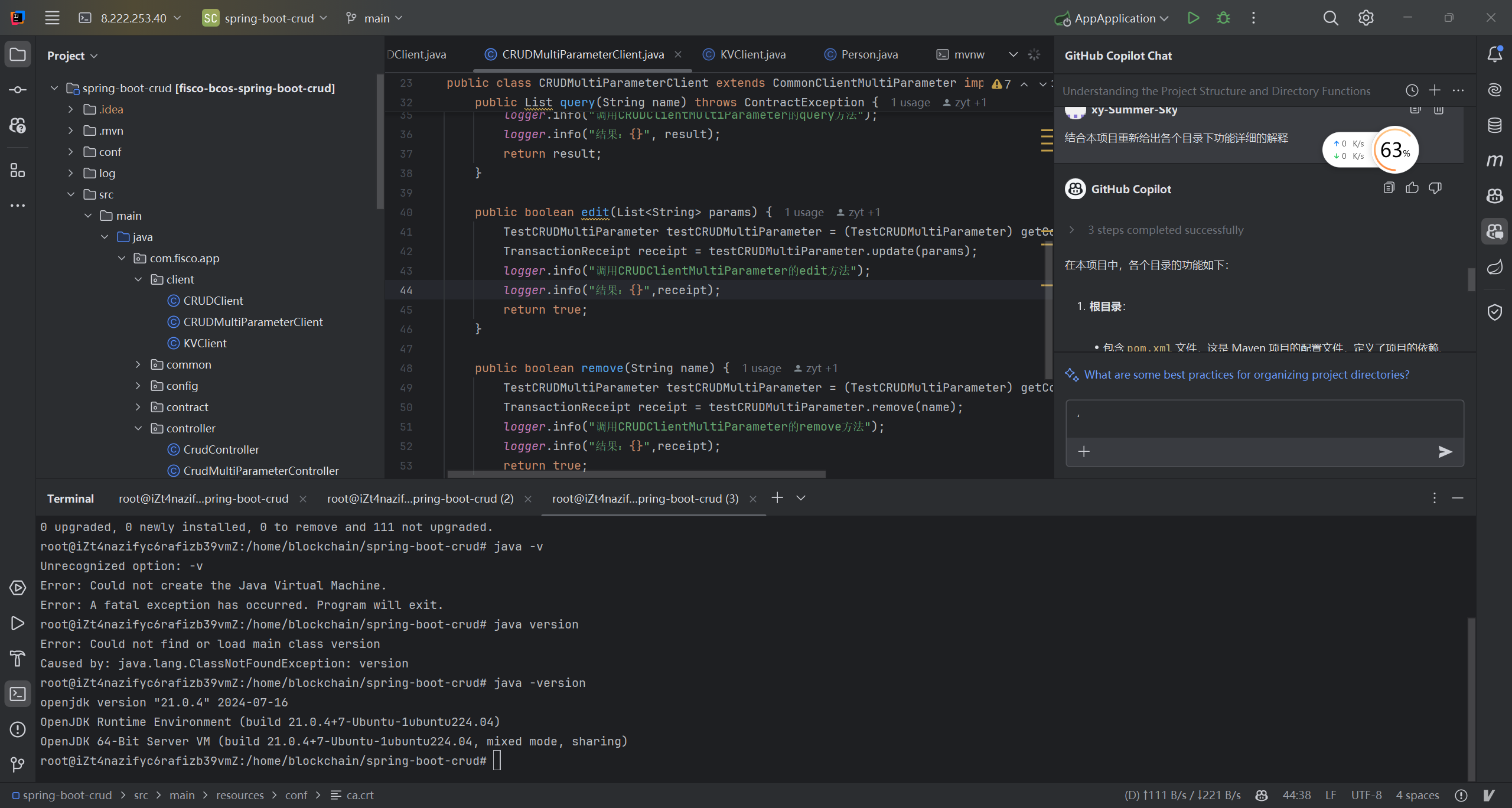


1. 所安装的Java版本是： 21.0.4 （请提供java -version的运行截图）（2分）



1. 你的实验环境中是否已经安装了VS Code或者IntelliJ IDEA等必要的开发IDE？若无，请给出你的安装步骤。（2分）

已经安装IDEA



**在命令行搭建FISCO BCOS区块链（15分）**

1. 你创建的工作目录是： /home/blockchain (/home/blockchain/fisco) （请填写绝对路径），创建并进入该目录需要哪些命令？（2分）

Cd /home/blockchain/fisco

1. 你是如何下载build\_chain.sh脚本的？使用了哪些命令？（5分）

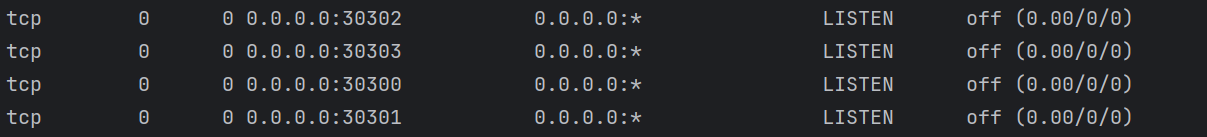
使用命令curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v3.11.0/build\_chain.sh && chmod u+x build\_chain.sh



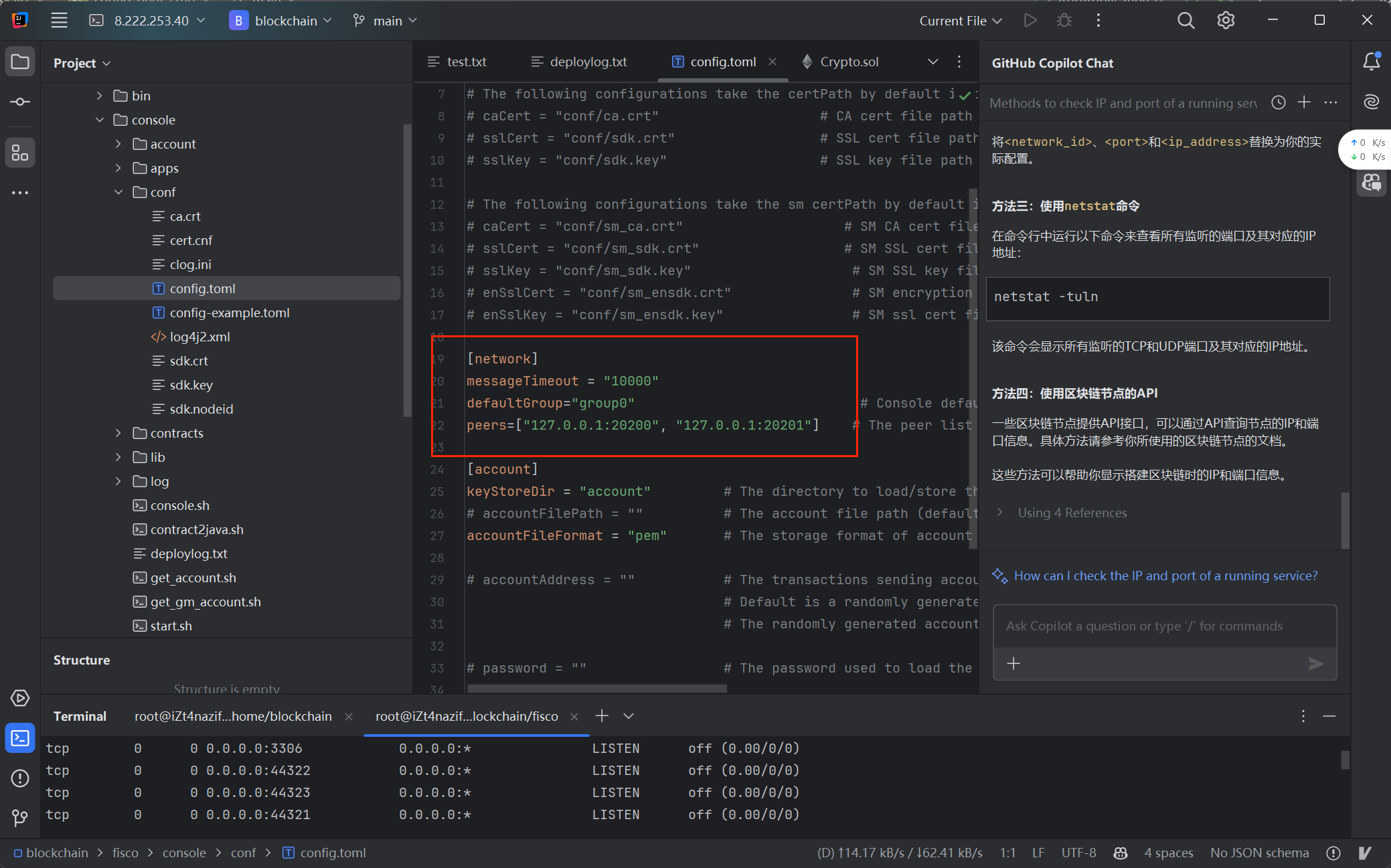
1. 你指定搭建区块链时的IP端口是什么？（3分）

bash build\_chain.sh -l 127.0.0.1:4 -p 30300,20200

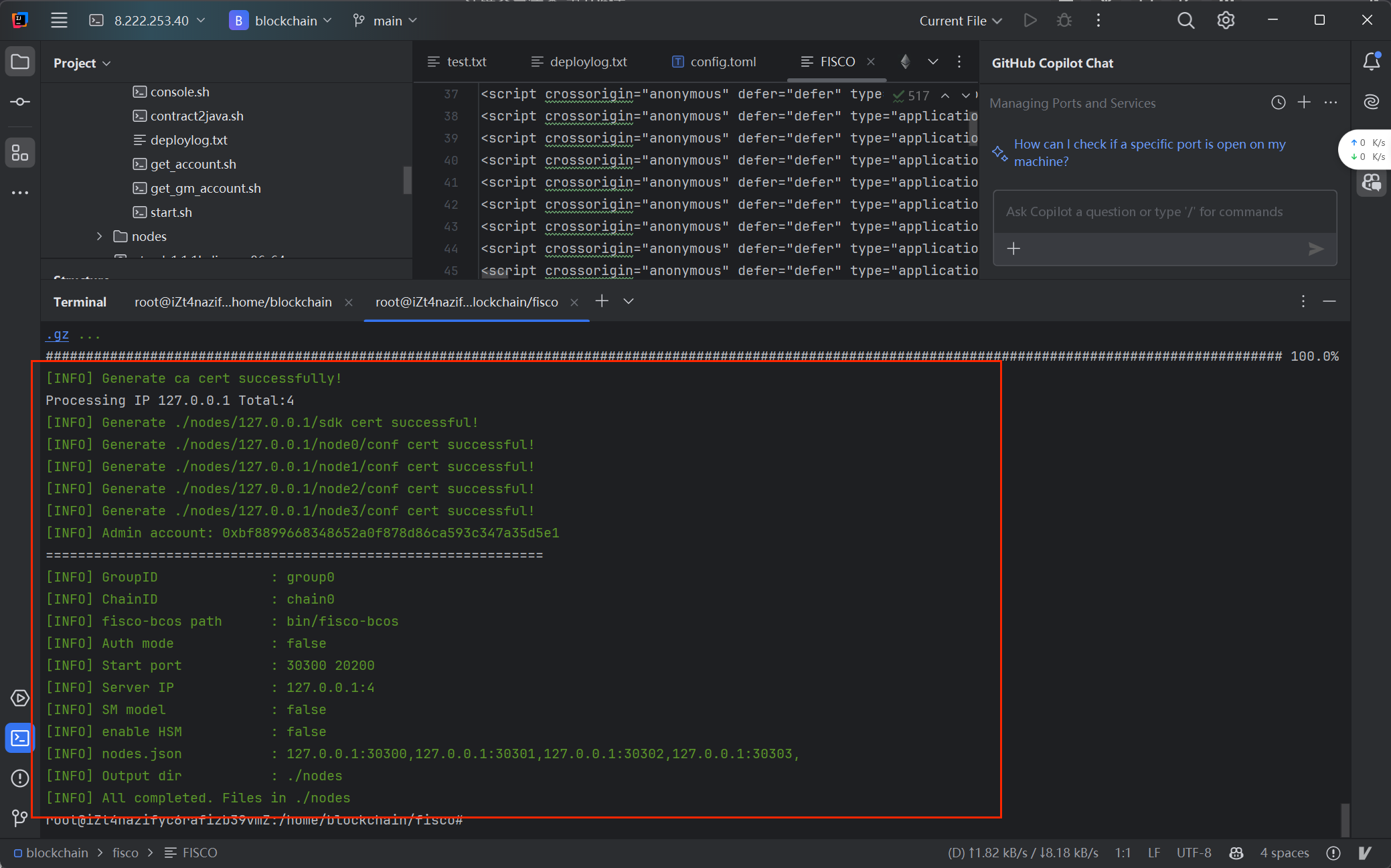
127.0.0.1的IP上构建4个节点，P2P端口以30300开始，RPC端口以20200开始







1. 使用build\_chain.sh脚本以后，命令行输出了什么？请简述。（3分）



1. 证书生成：成功生成了多个节点的证书，路径为`./nodes/127.0.0.1/sdk`和`./nodes/127.0.0.1/nodeX/conf`。

2. 管理员账户：管理员账户地址为`0xbf8899668348652a0f878d86ca593c347a35d5e1`。

3. 组ID：`group0`

4. 链ID：`chain0`

5. FISCO-BCOS路径：`bin/fisco-bcos`

6. 认证模式：`false`

7. 起始端口：`30300`和`20200`

8. 服务器IP：`127.0.0.1:4`

9. SM模式：`false`

10. 启用HSM：`false`

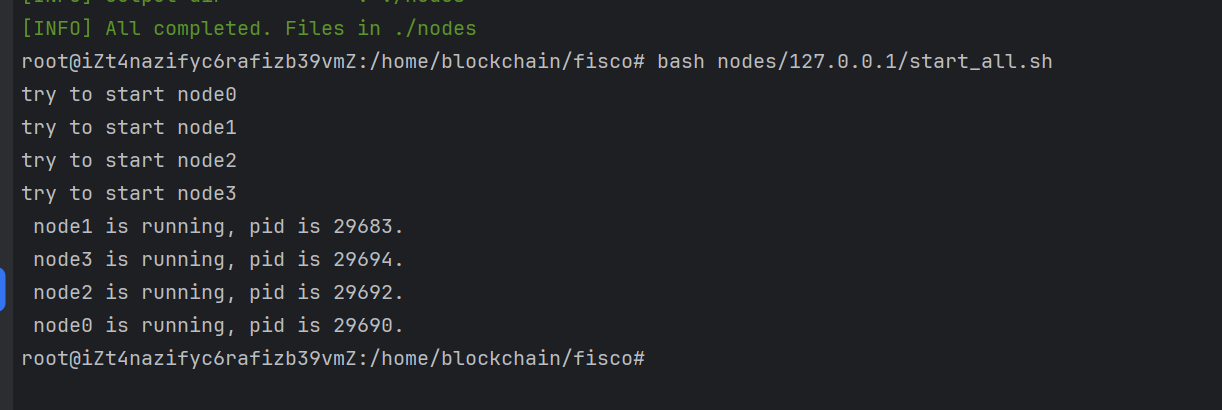
11. 节点配置文件：`127.0.0.1:30300,127.0.0.1:30301,127.0.0.1:30302,127.0.0.1:30303`

12. 输出目录：`./nodes`

13. 完成状态：所有文件已生成并保存在`./nodes`目录中。

1. 你使用了哪些命令启动了区块链？请简述你是如何检查区块链启动正常的。（2分）

使用命令：bash nodes/127.0.0.1/start\_all.sh



通过ps aux |grep -v grep |grep fisco-bcos检查进程是否启动

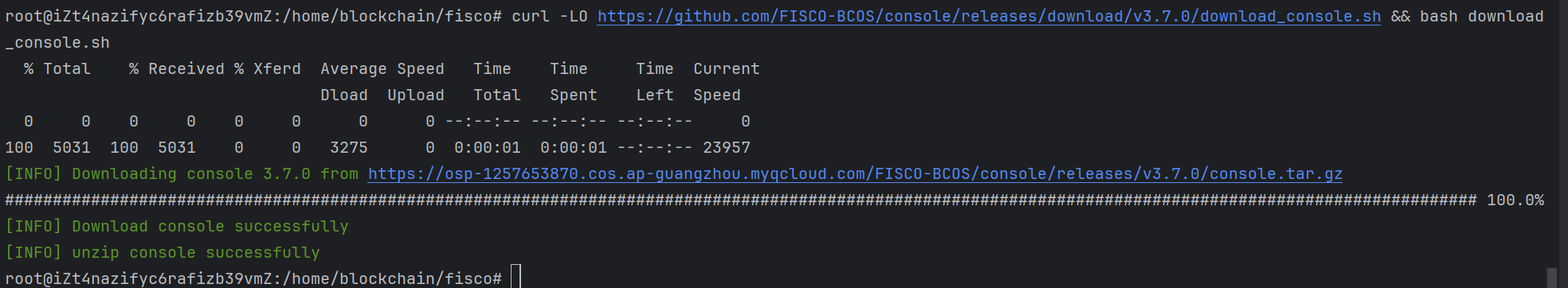


**使用控制台部署调用HelloWorld合约（12分）**

1. 使用哪些命令下载控制台？（2分）

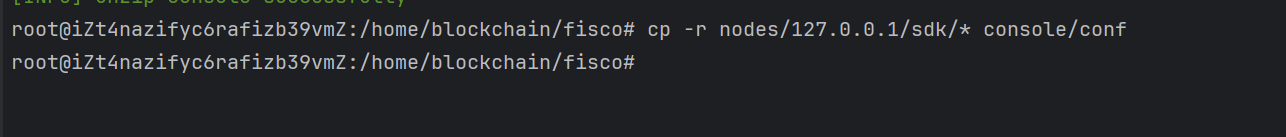
使用命令

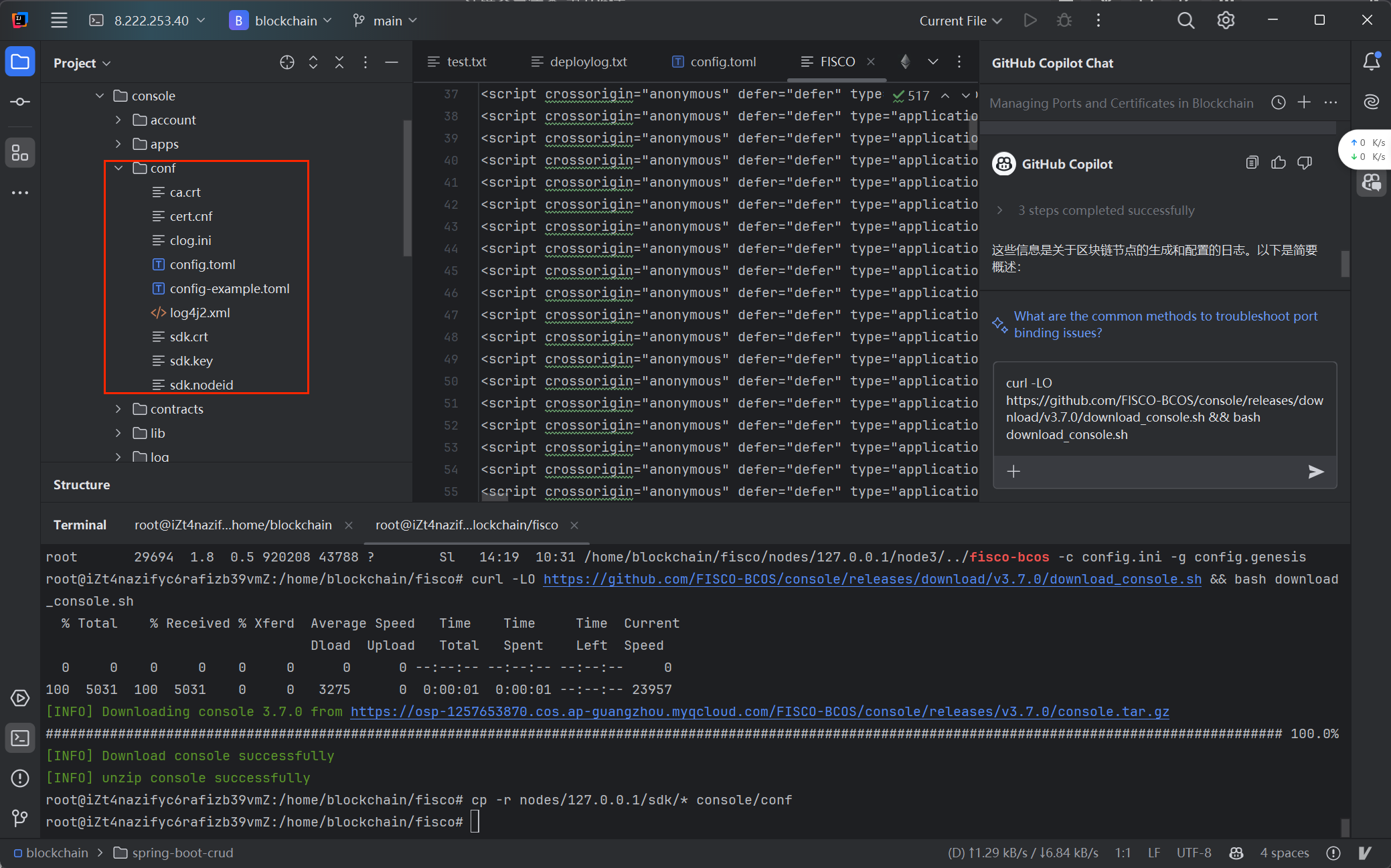
curl-LO [https://github.com/FISCO-BCOS/console/releases/download/v3.7.0/download\_console.sh && bash download\_console.sh](https://github.com/FISCO-BCOS/console/releases/download/v3.7.0/download_console.sh%20&&%20bash%20download_console.sh)下载



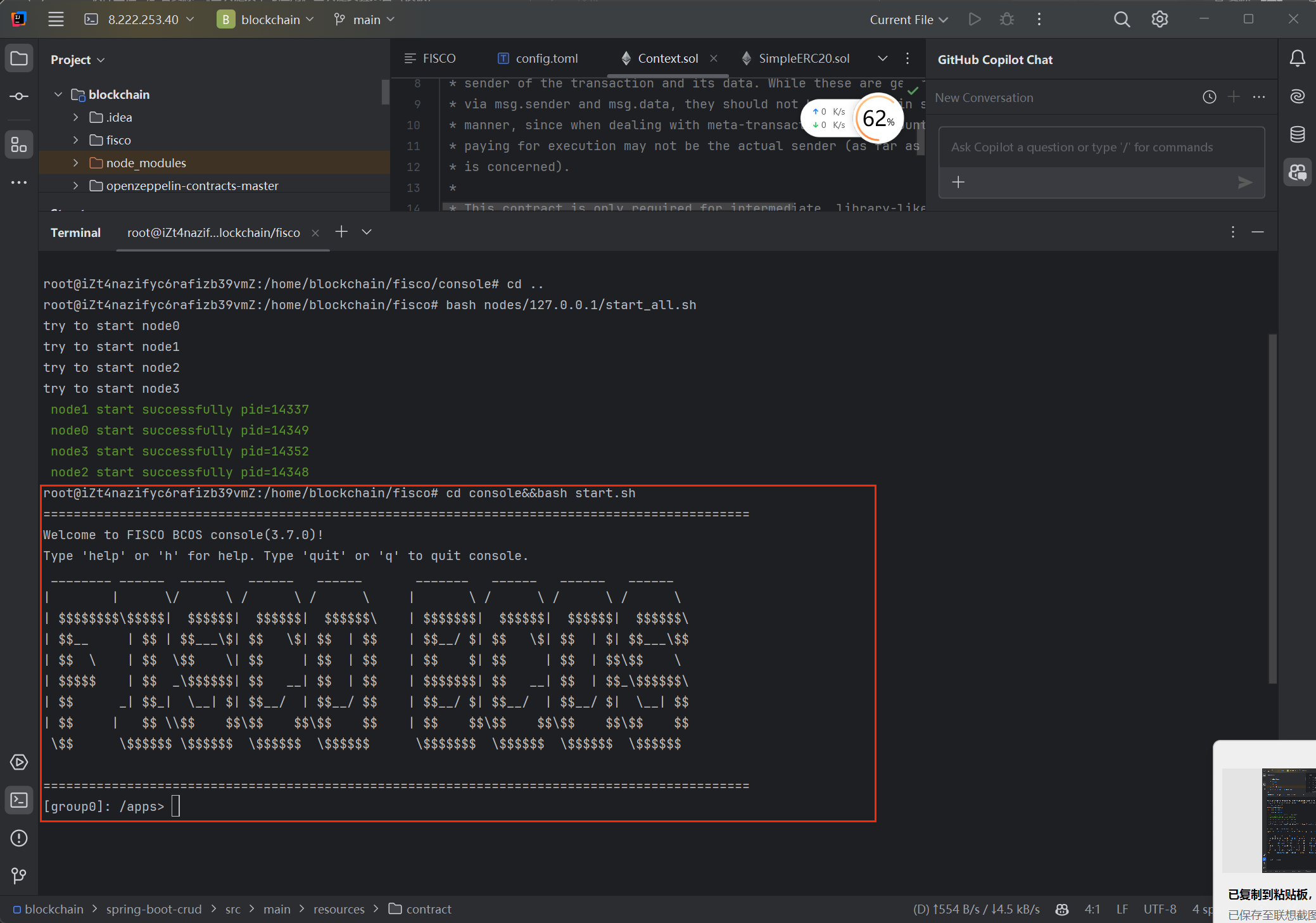
1. 你是怎么配置控制台的？请截图并描述。（3分）

控制台与节点之间默认开启SSL连接，控制台需要配置证书才可连接节点。开发建链脚本在生成节点的同时，生成了SDK证书，可直接拷贝生成的证书供控制台使用





使用cd console &&bash start.sh命令进行启动



1. 请尝试描述以下HelloWorld合约所实现的功能。（3分）

构建了一个可以有默认/自定义设置名称并获取名称的合约

pragma solidity >=0.6.10 <=0.8.26;

contract HelloWorld {

string name;

constructor() public {

name = "Hello, World!";

}

function get() public view returns (string memory) {

return name;

}

function set(string memory n) public {

name = n;

}

}

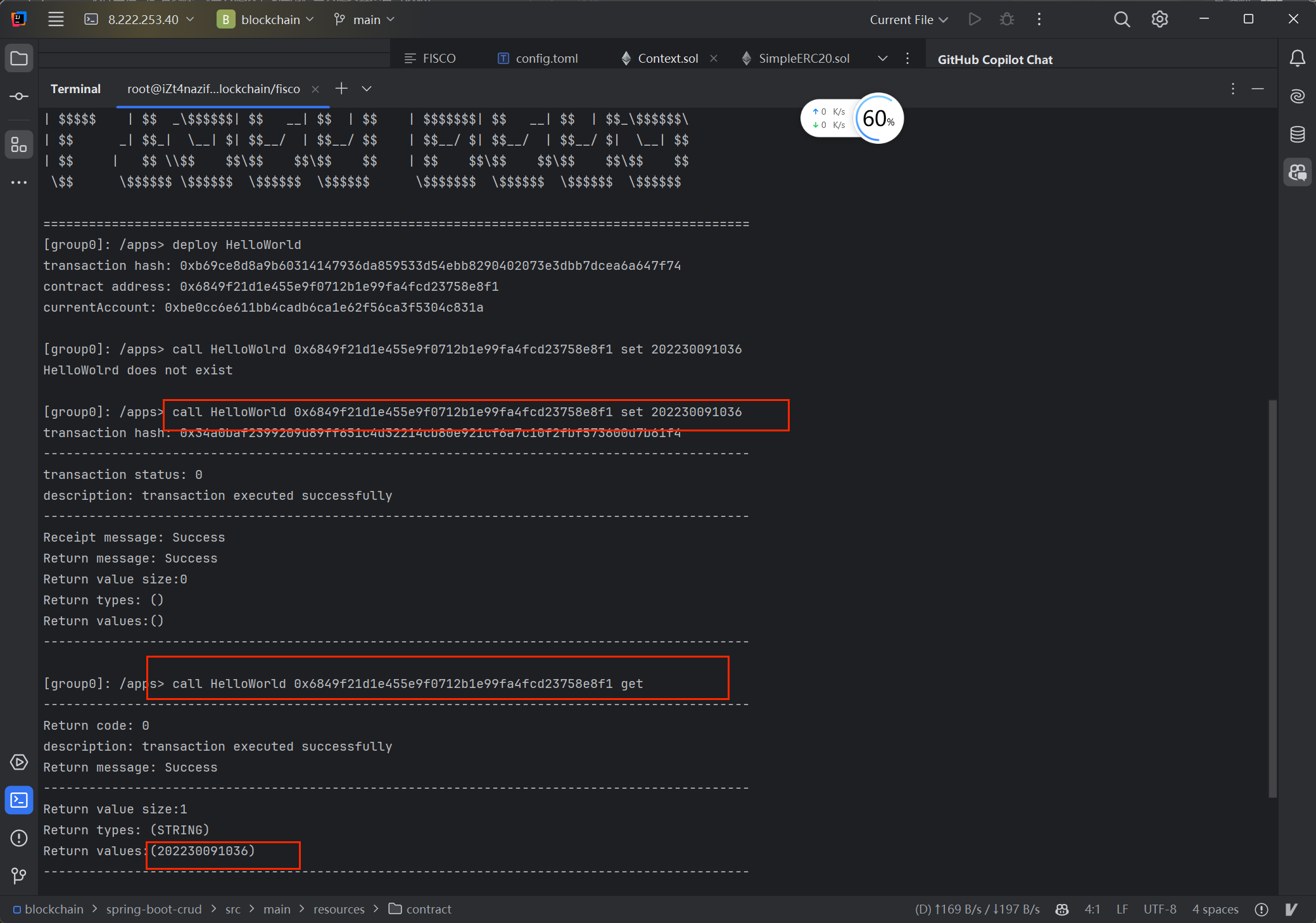
1. 请提供步骤截图，描述如何使用控制台部署HelloWorld合约、调用HelloWorld合约set接口。在调用set接口时，要求填入自己的学号。（4分）

启动控制台后,

使用命令：deploy HelloWorld部署合约

call HelloWorld 0x6849f21d1e455e9f0712b1e99fa4fcd23758e8f1 set 202230091036调用set接口

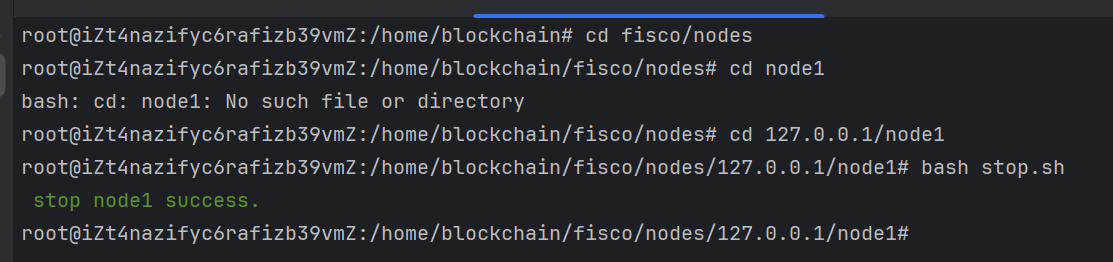
call HelloWorld 0x6849f21d1e455e9f0712b1e99fa4fcd23758e8f1 get获取name，使用get接口



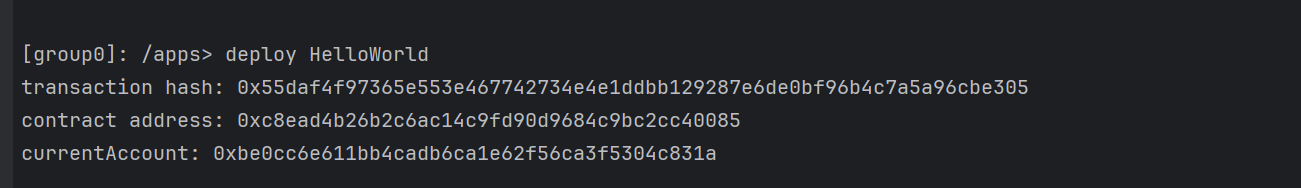
**PBFT共识算法容错模拟实践（5分）**

1. 通过逐渐停掉其中1个节点、2个节点，观察是否还能成功部署HelloWorld合约的截图，在控制台中输入getSyncStatus命令查看节点数量。（3分）

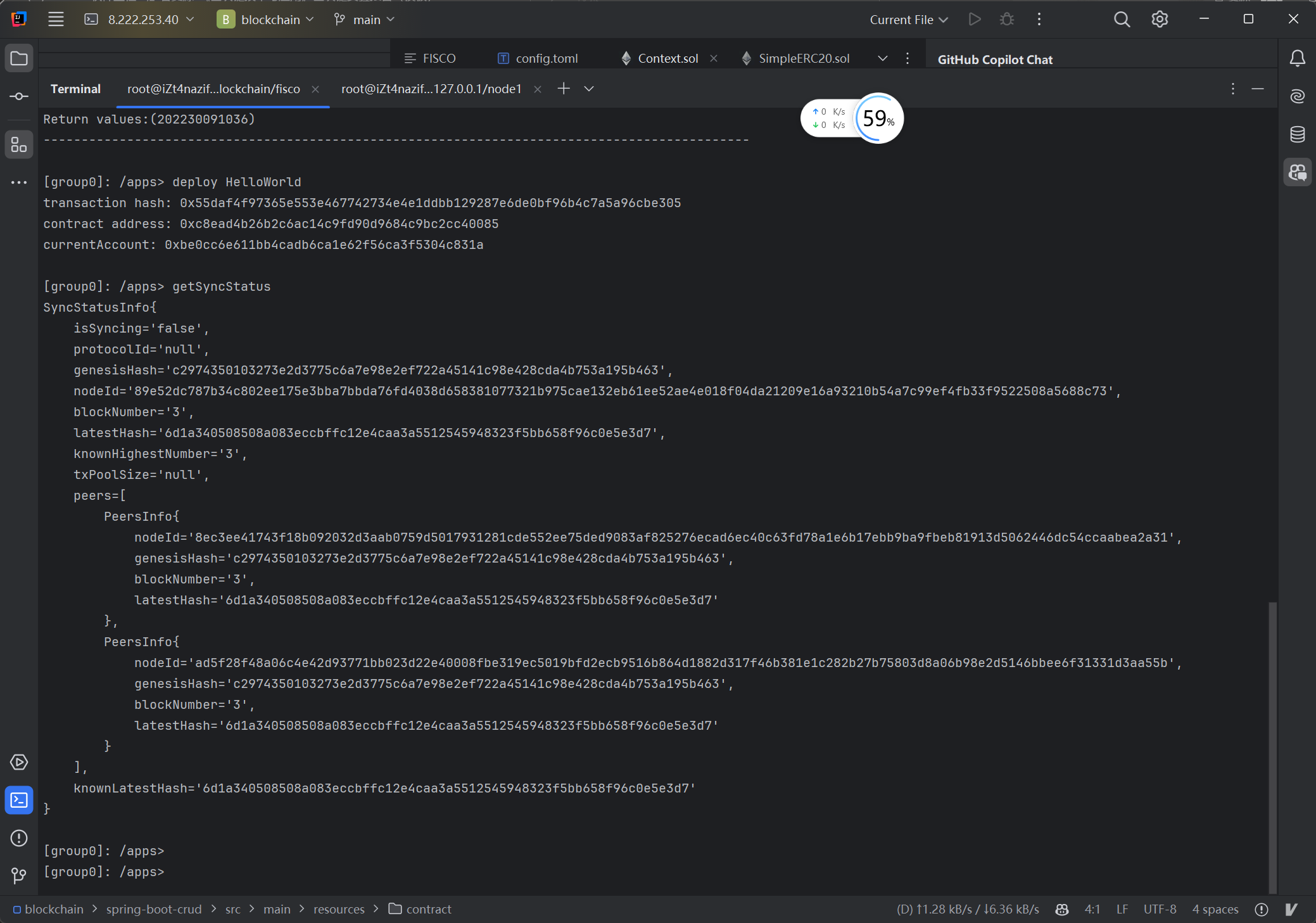
停用一个节点：



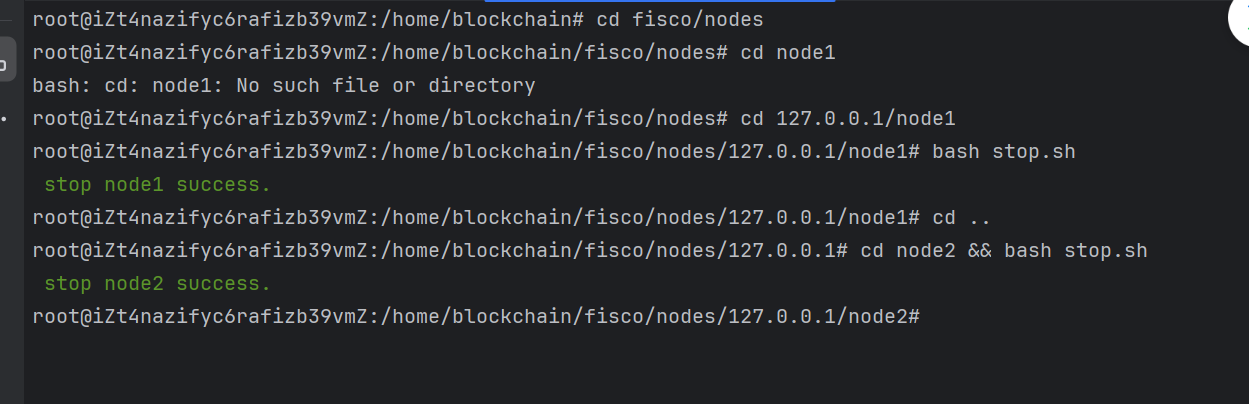
停用一个依然部署成功：



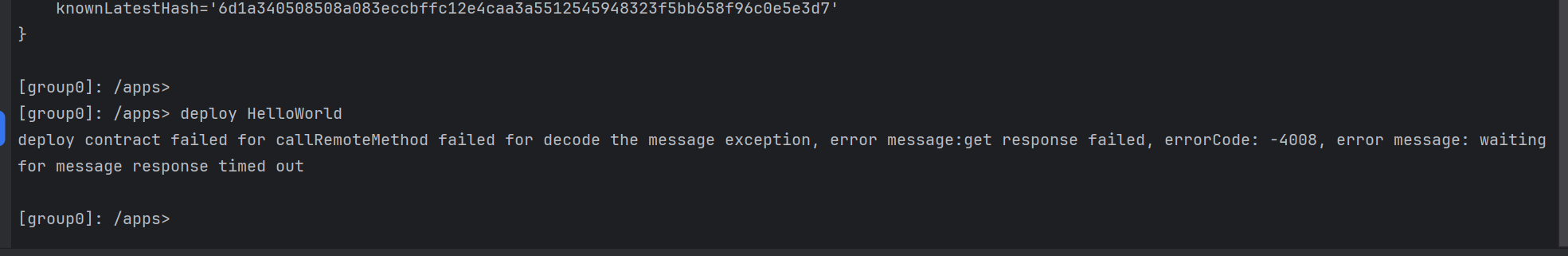
getSyncStatus命令查看节点数量：



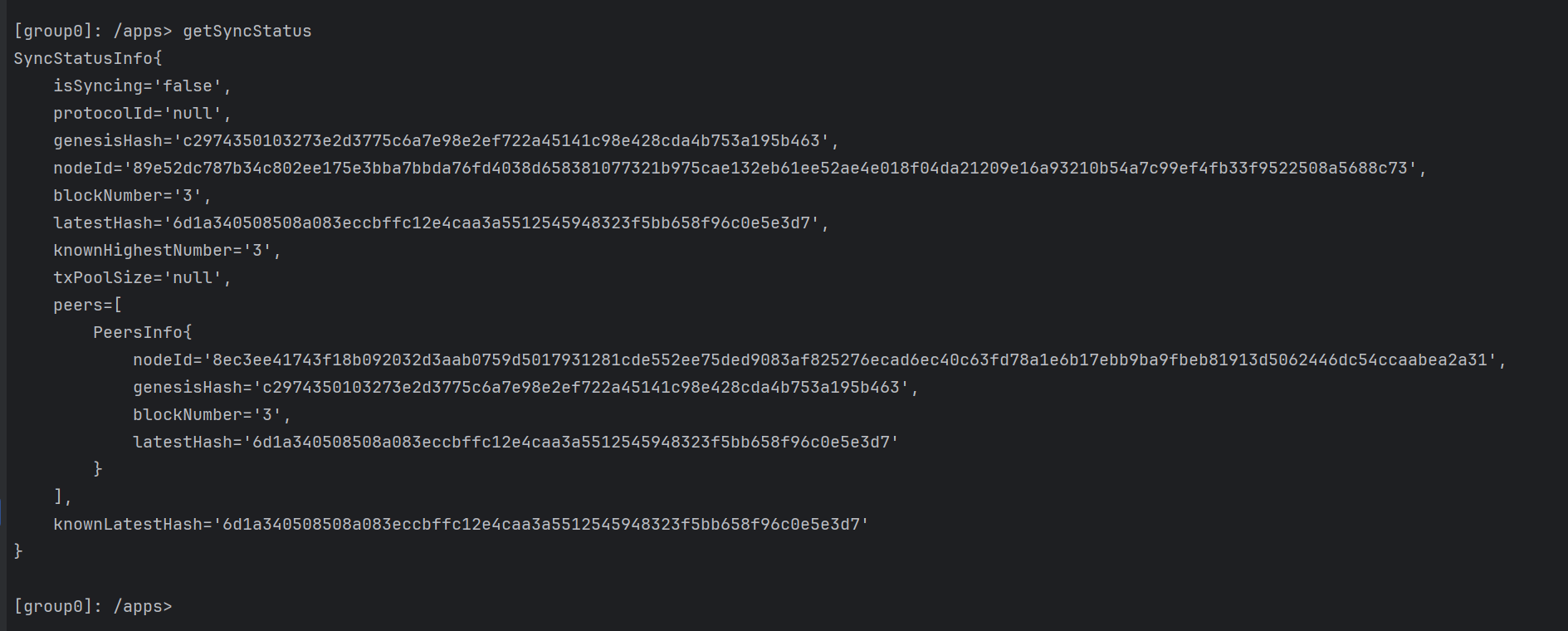
停止节点2：



再次部署合约失败：



getSyncStatus命令查看节点数量：



1. 尝试分析上一步停掉部分节点后，操作成功/失败的原因（2分）

**PBFT**(Practical Byzantine Fault Tolerance)共识算法可以在少数节点作恶(如伪造消息)场景中达成共识，它采用签名、签名验证、哈希等密码学算法确保消息传递过程中的防篡改性、防伪造性、不可抵赖性，并优化了前人工作，将拜占庭容错算法复杂度从指数级降低到多项式级别，在一个由(3\*f+1)个节点构成的系统中，只要有不少于(2\*f+1)个非恶意节点正常工作，该系统就能达成一致性，如：7个节点的系统中允许2个节点出现拜占庭错误。

作恶节点数量为f（注意：作恶节点可能不发送任何消息，也可能发送错误消息）为了保证一致性达成，系统内节点数量必须大于3。

由于不知道那f个节点是恶意节点还是故障节点，这f个节点可以不发送消息，也可以发送错误的消息，所以在设计阈值的时候，我们要保证必须在n-f个状态复制机的沟通内，就要做出决定，因为如果阈值设置为需要n-f+1个消息，那么如果这f个作恶节点全部不回应，那这个系统根本无法运作下去）。

在n-f个状态复制机的沟通内，要做出决定，但是我们无法预测这f个作恶节点做了什么（错误消息/不发送），所以我们并不知道，这n-f个里面有几个是作恶节点，我们必须保证正常的节点大于作恶节点数。所以有 n-f-f > f，从而得出了n > 3f,，则n最小为3f+1.

本实验中共四个节点，为了达成一致性，只能容忍1个节点出现错误。当错误节点为2时，不满足“n最小为3f+1.”的条件

**实验二 使用智能合约与区块链交互（40分）**

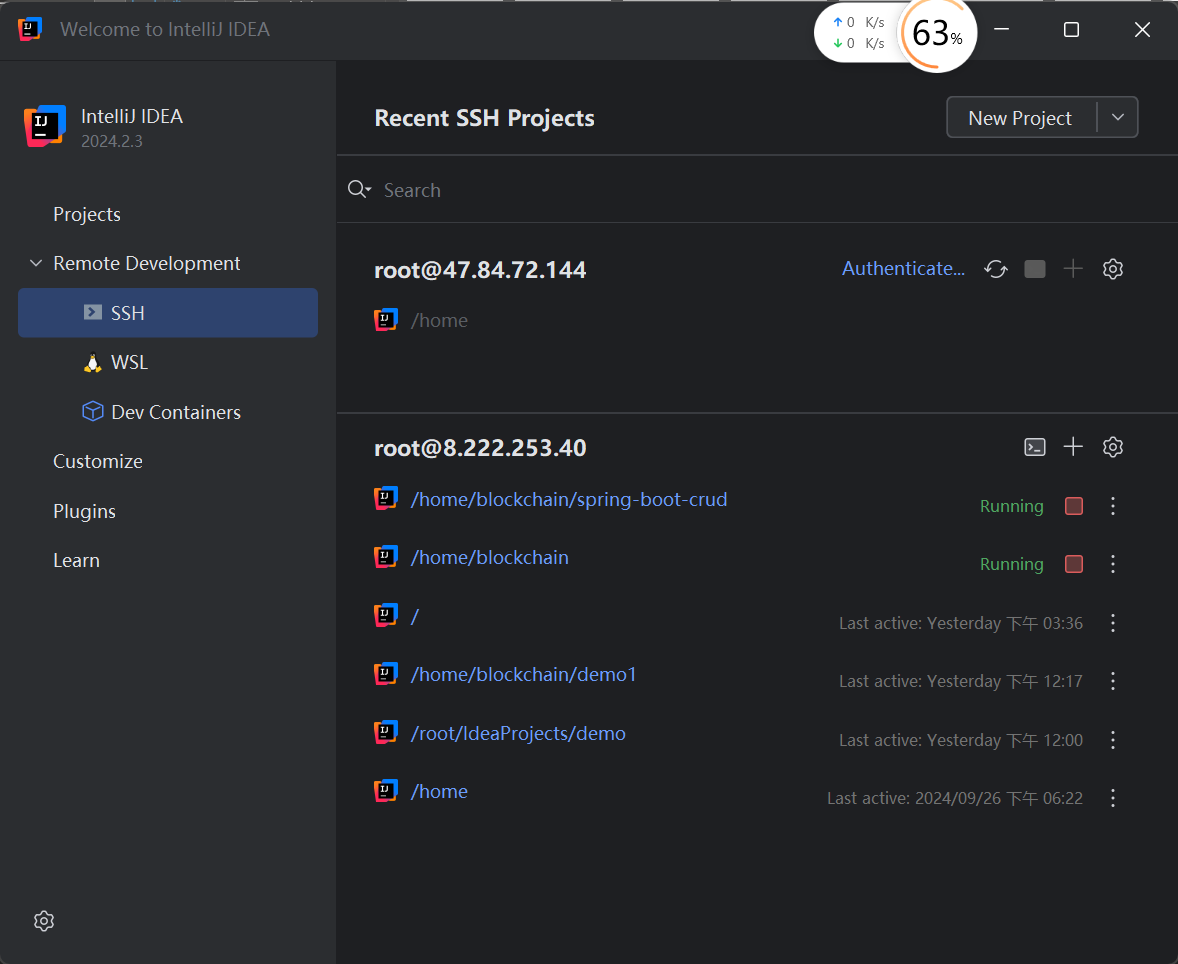
**准备Solidity开发环境（10分）**

搭建Solidity开发环境有以下几种方式：

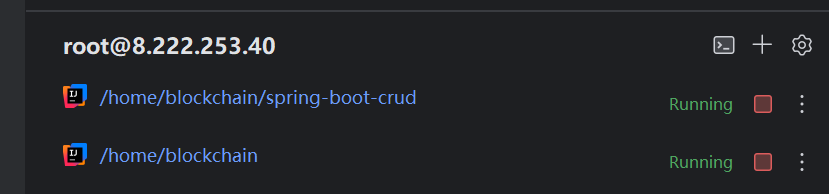
* 使用Remix 在线/本地搭建的方式，链接：[在线](https://remix.ethereum.org/)、[本地](https://github.com/ethereum/remix-project)
* 使用VSCode + 插件的方式
* 使用IntelliJ IDEA + 插件的方式

1. 你使用的是什么方式搭建的？请截图描述步骤。（8分）

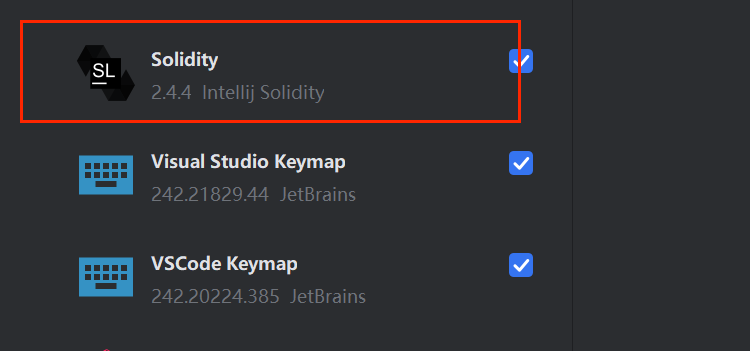
远程ssh连接：



指定项目目录（输入授权密码）:



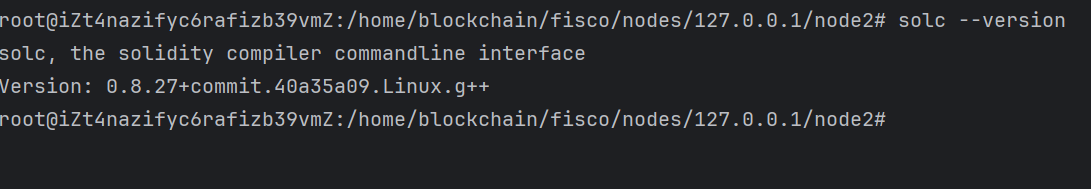
远程主机安装idea并且安装插件：



使用IntelliJ IDEA + 插件的方式

1. 你使用的Solidity版本是什么版本？（2分）

0.8.27



**实现ERC20标准的合约ERC20Test（25分）**

1. 根据ERC20标准的接口定义，编写智能合约，合约名为ERC20Test，要求实现mint、transfer、approve、transferFrom、balanceOf接口（15分）

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.4;

contract ERC20Test {

string public \_name = "SimpleToken";

string public \_symbol = "STK";

uint8 public \_decimals = 0;

uint256 public \_totalSupply;

address public owner;

struct User {

uint256 balance;

mapping(address => uint256) allowance;

}

mapping(address => User) private users;

event Transfer(address indexed from, address indexed to, uint256 value);

event Approval(address indexed owner, address indexed spender, uint256 value);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can call this function");

\_;

}

constructor(uint256 initialSupply) {

owner = msg.sender;

\_totalSupply = initialSupply \* 10 \*\* uint256(\_decimals);

users[msg.sender].balance = \_totalSupply;

emit Transfer(address(0), msg.sender, \_totalSupply);

}

function name() public view returns (string memory) {

return \_name;

}

function symbol() public view returns (string memory) {

return \_symbol;

}

function decimals() public view returns (uint8) {

return \_decimals;

}

function totalSupply() public view returns (uint256) {

return \_totalSupply;

}

function balanceOf(address \_owner) public view returns (uint256 balance) {

return users[\_owner].balance;

}

function transfer(address \_to, uint256 \_value) public returns (bool success) {

require(\_to != address(0), "ERC20: transfer to the zero address");

require(users[msg.sender].balance >= \_value, "ERC20: transfer amount exceeds balance");

users[msg.sender].balance -= \_value;

users[\_to].balance += \_value;

emit Transfer(msg.sender, \_to, \_value);

return true;

}

function transferFrom(address \_from, address \_to, uint256 \_value) public returns (bool success) {

require(\_from != address(0), "ERC20: transfer from the zero address");

require(\_to != address(0), "ERC20: transfer to the zero address");

require(users[\_from].balance >= \_value, "ERC20: transfer amount exceeds balance");

require(users[\_from].allowance[msg.sender] >= \_value, "ERC20: transfer amount exceeds allowance");

users[\_from].balance -= \_value;

users[\_to].balance += \_value;

users[\_from].allowance[msg.sender] -= \_value;

emit Transfer(\_from, \_to, \_value);

return true;

}

function approve(address \_spender, uint256 \_value) public returns (bool success) {

require(\_spender != address(0), "ERC20: approve to the zero address");

users[msg.sender].allowance[\_spender] = \_value;

emit Approval(msg.sender, \_spender, \_value);

return true;

}

function allowance(address \_owner, address \_spender) public view returns (uint256 remaining) {

return users[\_owner].allowance[\_spender];

}

function mint(address account, uint256 amount) public onlyOwner returns (bool) {

require(account != address(0), "ERC20: mint to the zero address");

\_totalSupply += amount;

users[account].balance += amount;

emit Transfer(address(0), account, amount);

return true;

}

}

参考内容链接：[https://ethereum.org/zh/developers/docs/standards/tokens/erc-20/#body](https://ethereum.org/zh/developers/docs/standards/tokens/erc-20/" \l "body)

|  |
| --- |
| ERC-20（以太坊意见征求 20）由 Fabian Vogelsteller 提出于 2015 年 11 月。这是一个能实现智能合约中代币的应用程序接口标准。  ERC-20 的功能示例包括：   * 将代币从一个帐户转到另一个帐户 * 获取帐户的当前代币余额 * 获取网络上可用代币的总供应量 * 批准一个帐户中一定的代币金额由第三方帐户使用 |

如果智能合约实施了下列方法和事件，它可以被称为 ERC-20 代币合约，一旦部署，将负责跟踪以太坊上创建的代币。

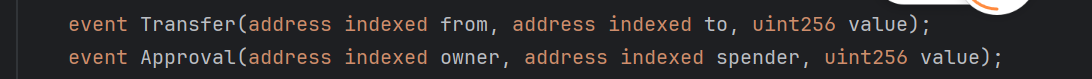
|  |
| --- |
| // 接口定义  function name() public view returns (string)  function symbol() public view returns (string)  function decimals() public view returns (uint8)  function totalSupply() public view returns (uint256)  function balanceOf(address \_owner) public view returns (uint256 balance)  function transfer(address \_to, uint256 \_value) public returns (bool success)  function transferFrom(address \_from, address \_to, uint256 \_value) public returns (bool success)  function approve(address \_spender, uint256 \_value) public returns (bool success)  function allowance(address \_owner, address \_spender) public view returns (uint256 remaining)  // 事件  event Transfer(address indexed \_from, address indexed \_to, uint256 \_value)  event Approval(address indexed \_owner, address indexed \_spender, uint256 \_value) |

1. 在合约中使用event、struct、mapping、require、modifier等语言特性。（10分，每个特性2分）

Event特性

event Transfer(address indexed from, address indexed to, uint256 value);

event Approval(address indexed owner, address indexed spender, uint256 value);



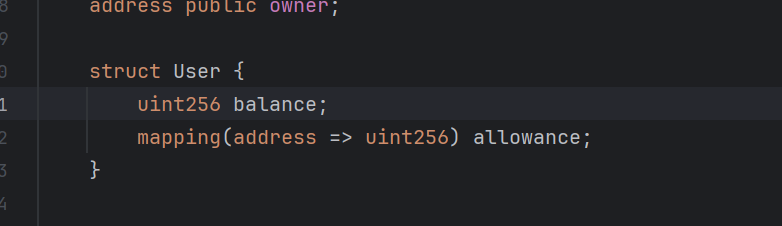
struct特性

struct User {

uint256 balance;

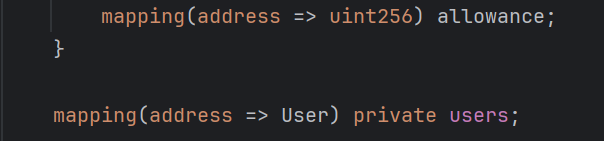
mapping(address => uint256) allowance;

}



mapping特性

mapping(address => User) private users;



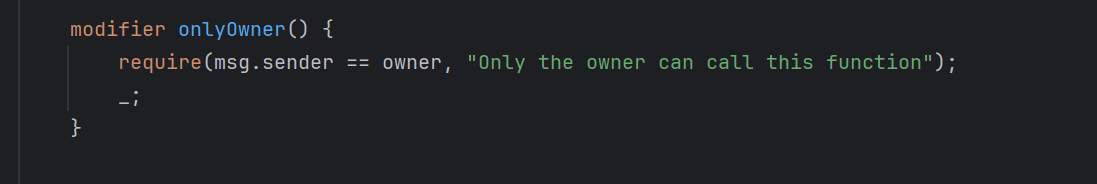
modifier特性：

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can call this function");

\_;

}



Require特性

function transfer(address \_to, uint256 \_value) public returns (bool success) {

require(\_to != address(0), "ERC20: transfer to the zero address");

require(users[msg.sender].balance >= \_value, "ERC20: transfer amount exceeds balance");

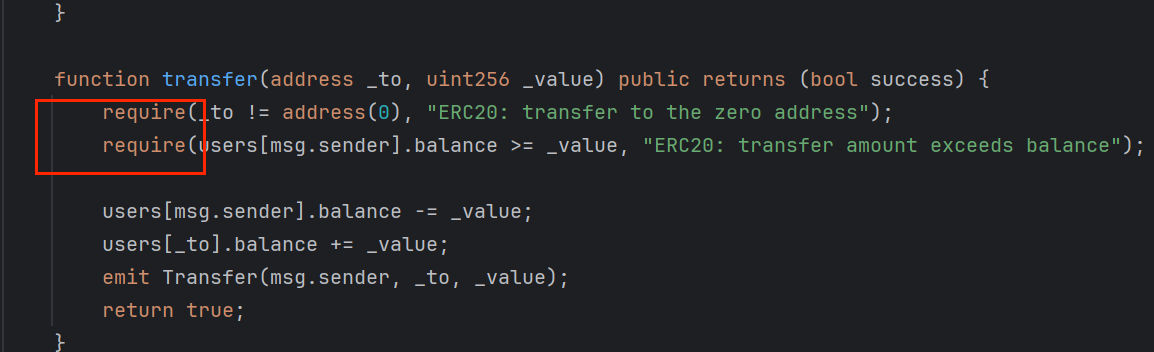
users[msg.sender].balance -= \_value;

users[\_to].balance += \_value;

emit Transfer(msg.sender, \_to, \_value);

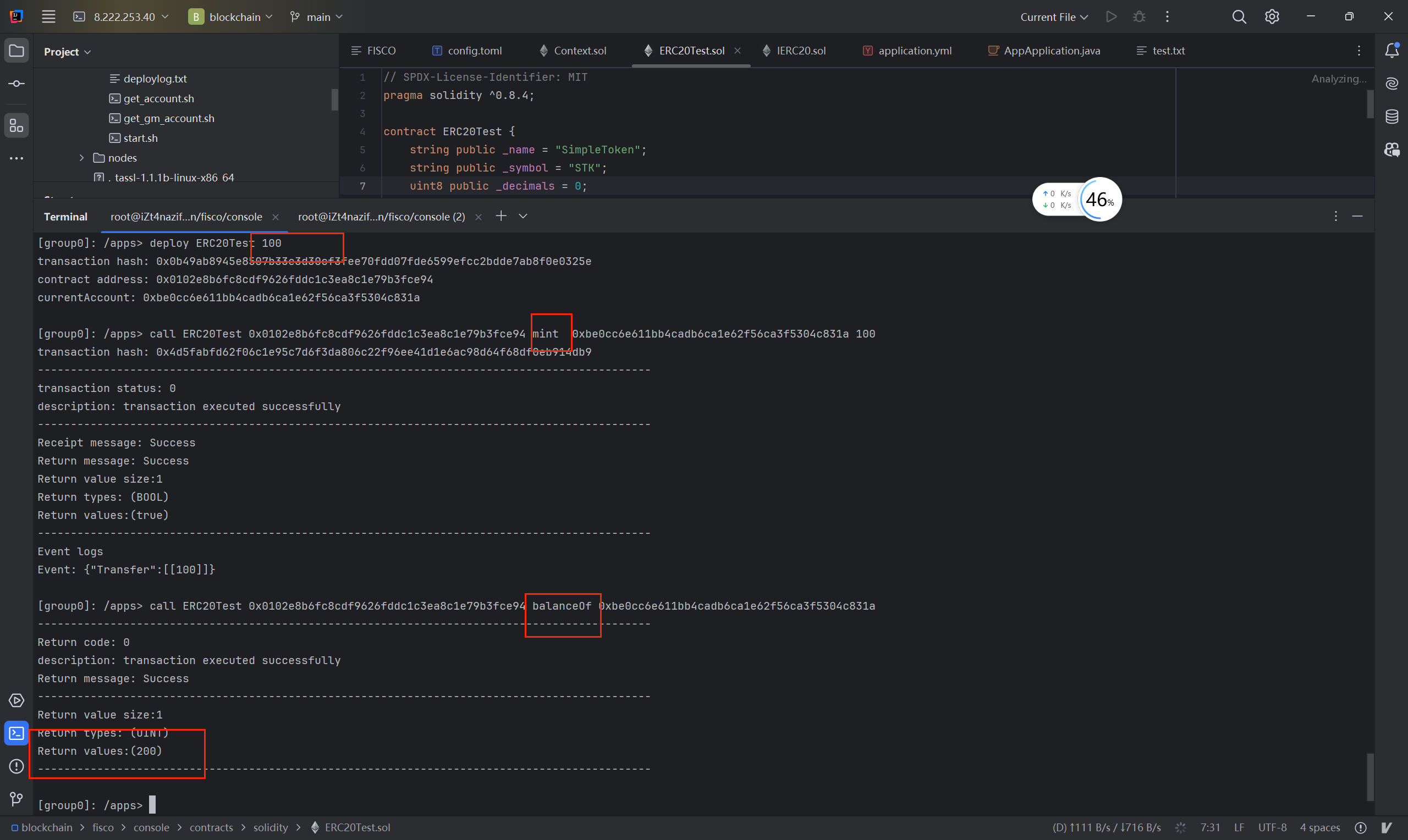
return true;

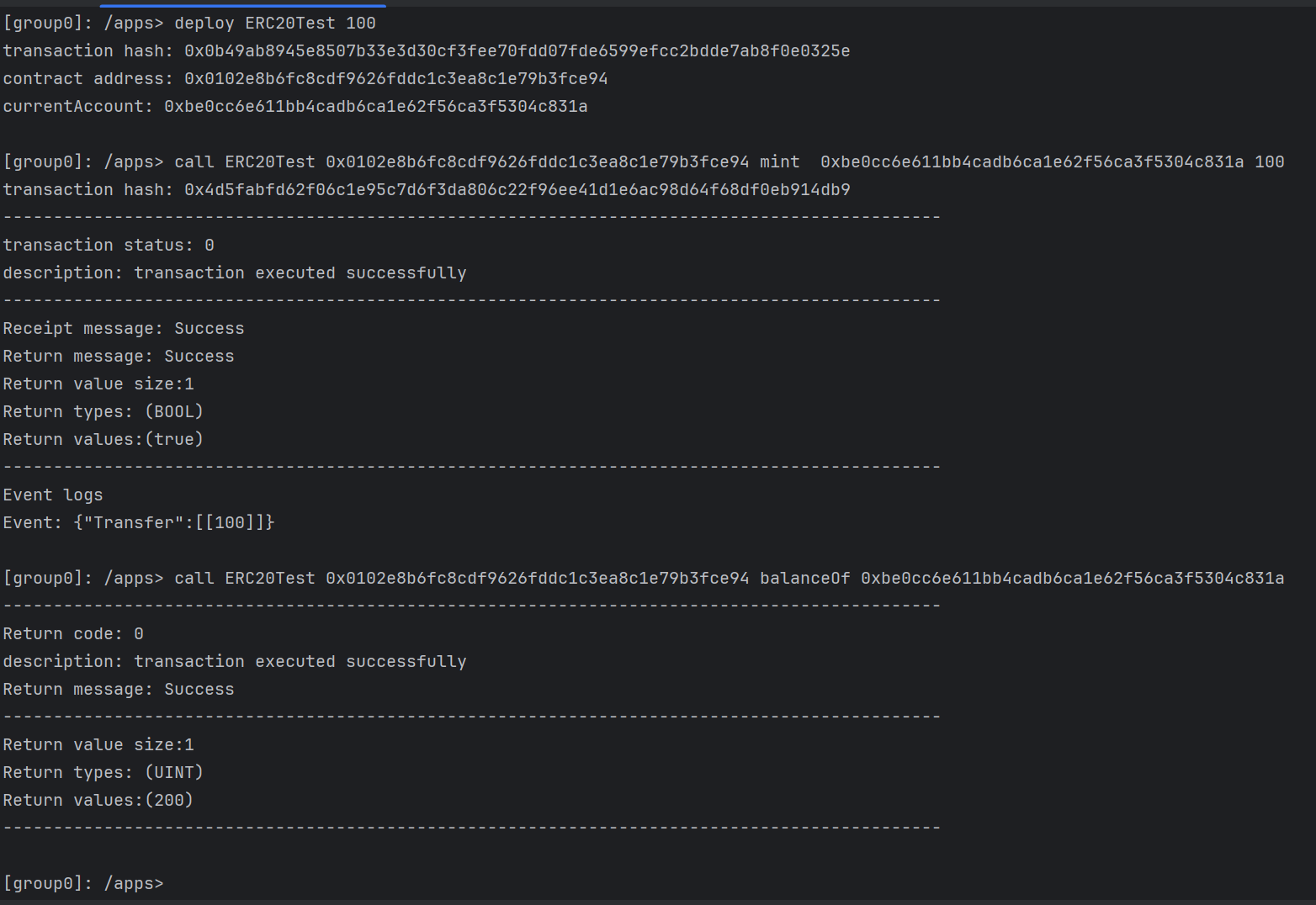
}



**在FISCO BCOS上部署调用合约ERC20Test（5分）**

在控制台中部署ERC20Test合约，调用合约的mint接口，给自己账号中发100个资产，并使用balanceOf接口查看结果（5分）





最终账户里有200个货币资产

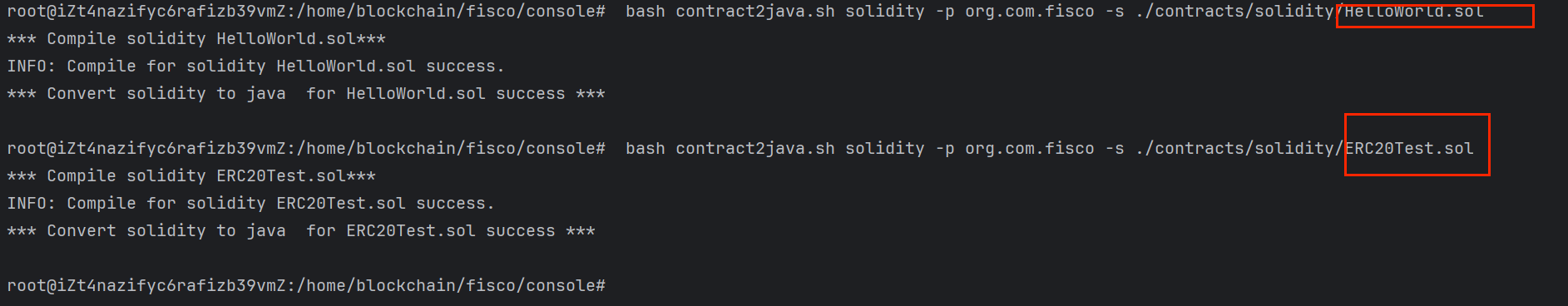
小数位设定为0位

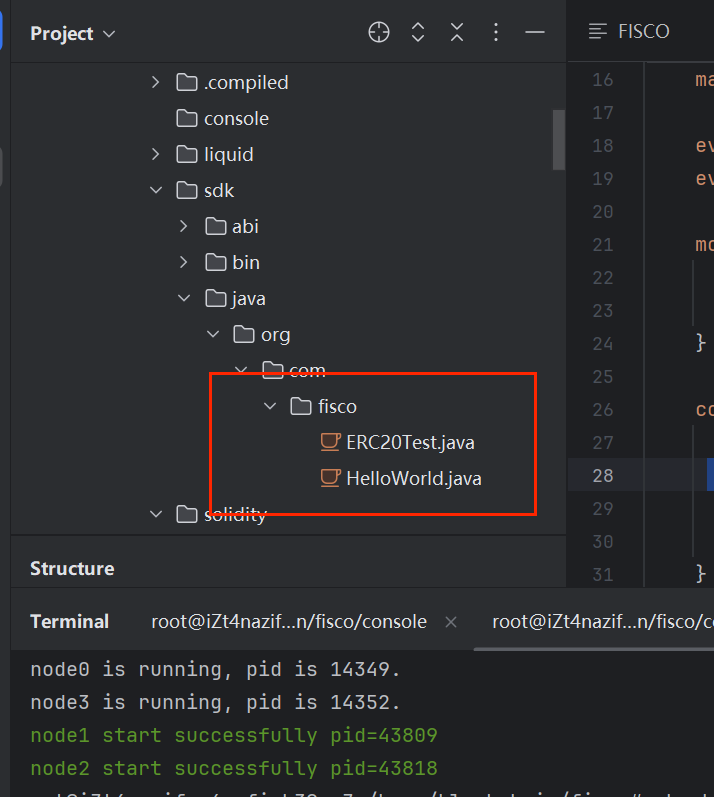
**实验三 在区块链上构建应用系统（20分）**

**将实验2的智能合约编译成Java接口文件（10分）**

参考文档链接，将实验2的智能合约编译成Java接口文件。（10分）

依次转化HelloWorld和ERC20Test合约为java文件



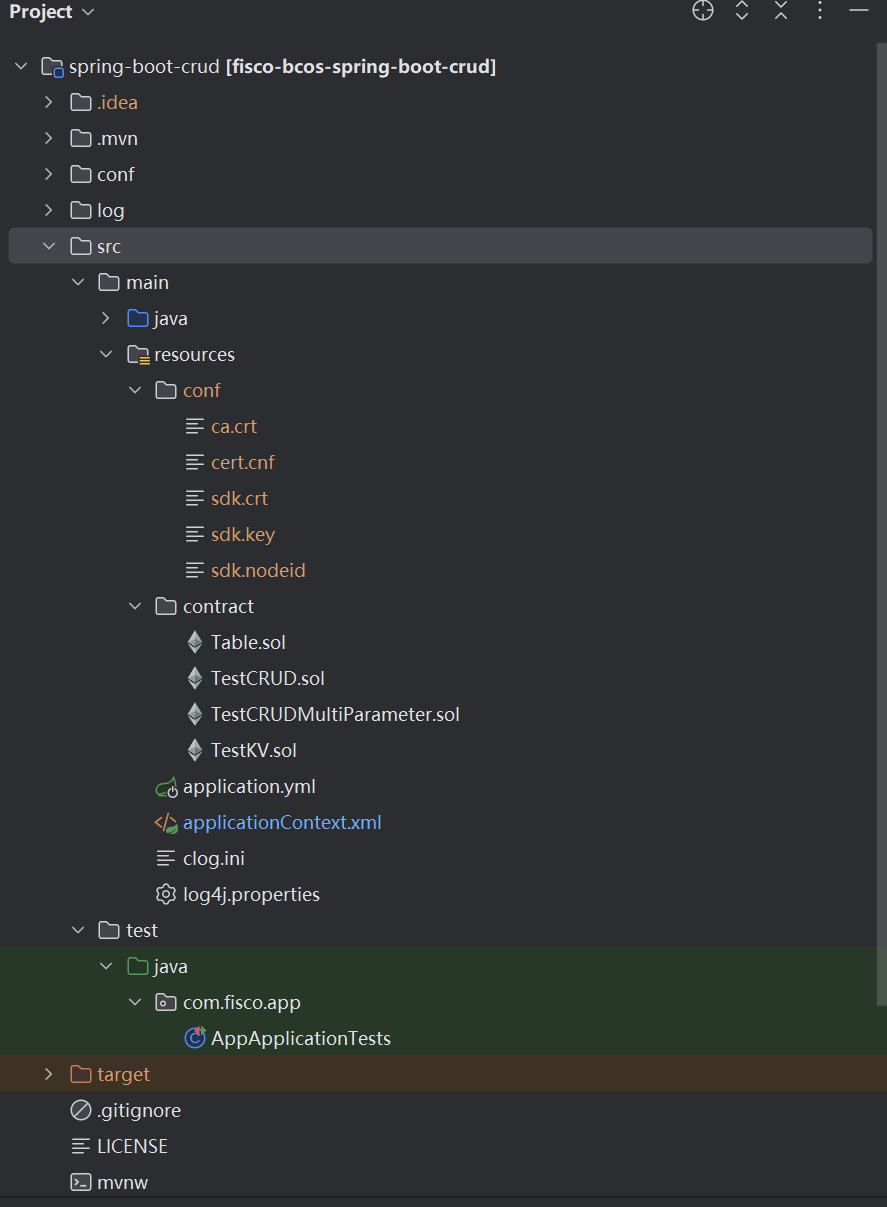
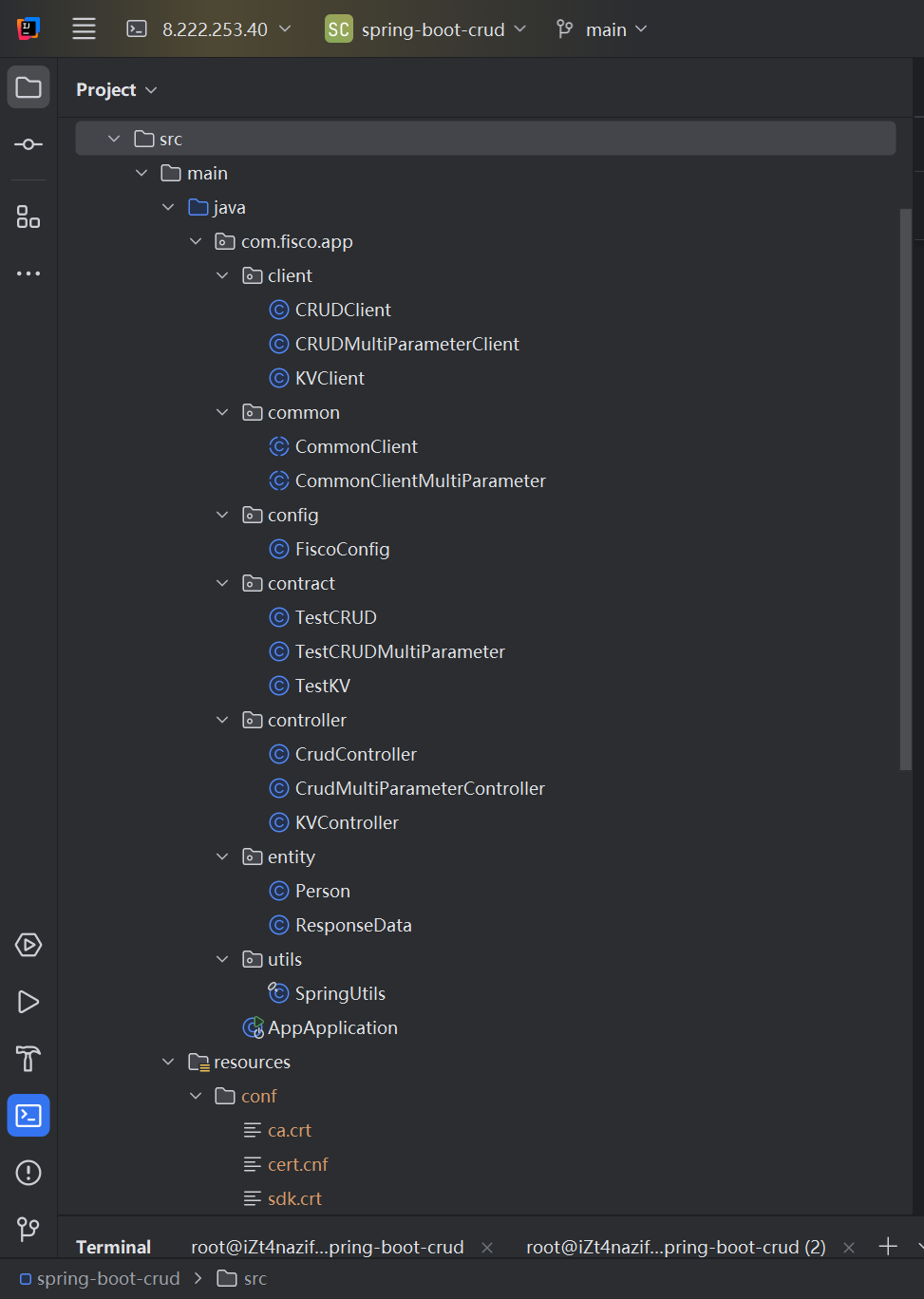


参考链接：[生成智能合约的Java接口文件](https://fisco-bcos-doc.readthedocs.io/zh-cn/latest/docs/sdk/java_sdk/contracts_to_java.html)

**使用Spring-boot-crud框架构建服务调用区块链（8分）**

参考demo项目：[FISCO-BCOS/spring-boot-crud](https://github.com/FISCO-BCOS/spring-boot-crud)

1. 根据demo教程指导，编译构建智能合约项目。请给出智能合约项目创建完成后，项目结构的截图，要求展开其中的src目录。（4分）



1. 简要说明项目中的代码项目结构，说明每一级目录代表的功能。（4分）

在本项目中，各个目录的功能如下：

1. **src/main/java**：

包含项目的主要 Java 源代码文件。

1. **src/main/resources**：

包含项目的资源文件，如证书、合约文件

**com.fisco.app 目录下的子目录**

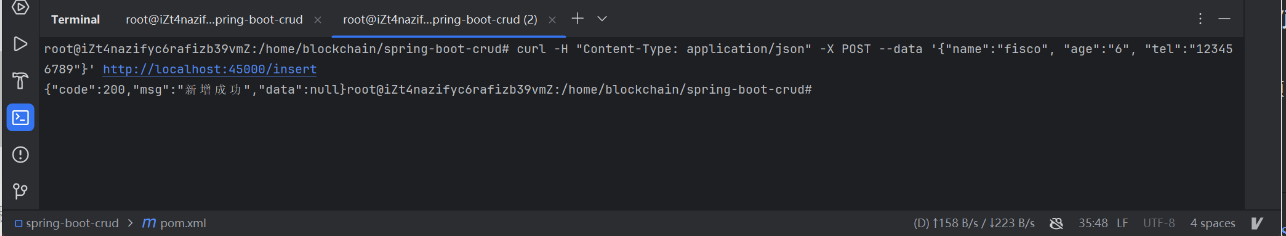
1. **com.fisco.app.client**：存放客户端相关的代码，提供服务接口，供Controller层调用
2. **com.fisco.app.common**：

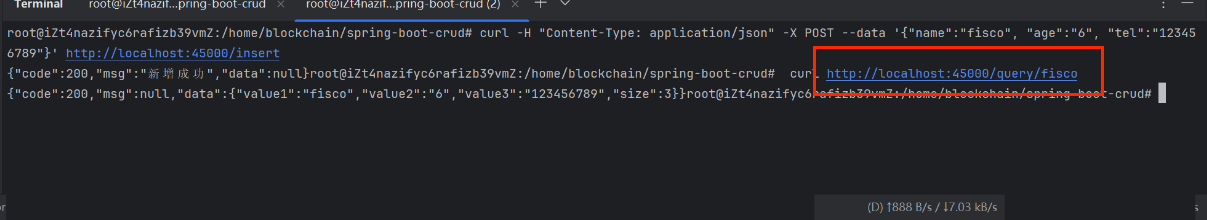
存放通用抽象类， 本项目中是两个抽象类，供其它具体类进行继承实现。

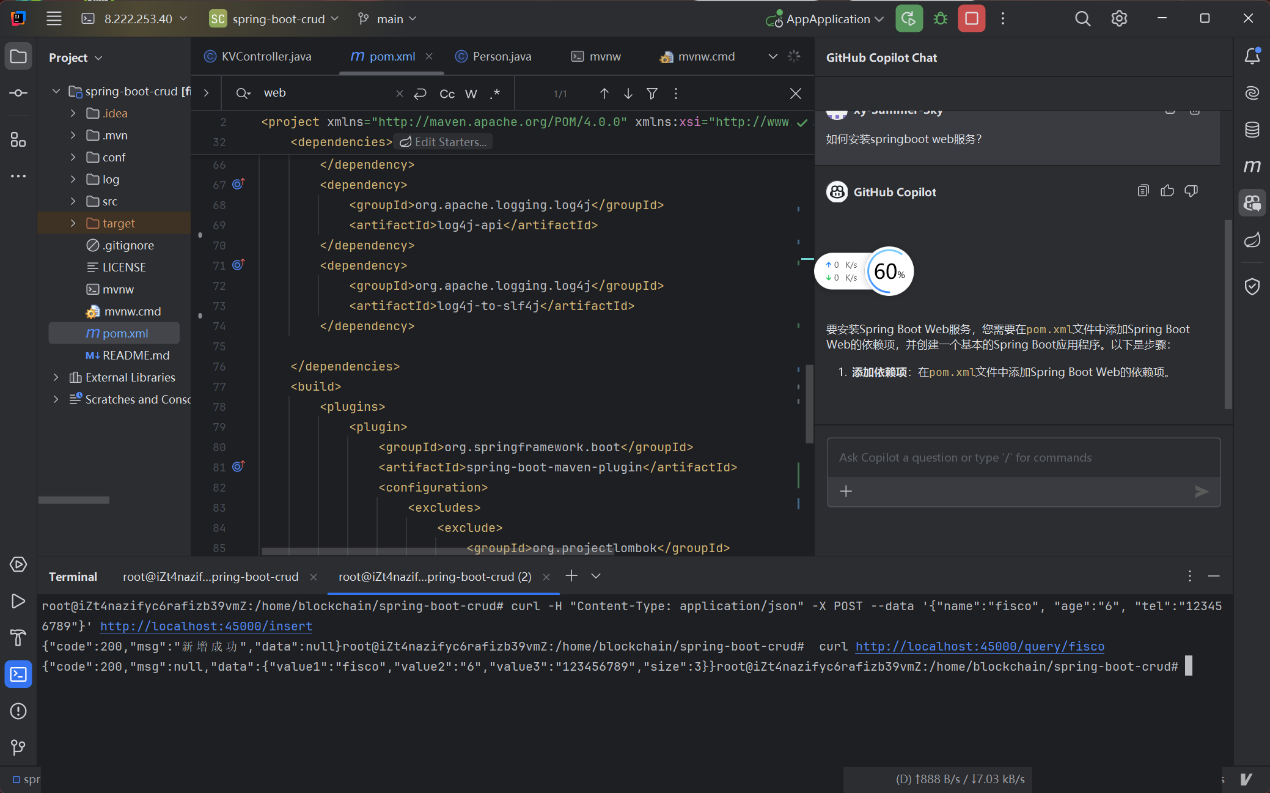
1. **com.fisco.app.contract**：存放智能合约相关的代码和类。智能合约是区块链应用程序的核心部分，负责调用部署合约并定义和执行区块链上的业务逻辑。
2. **com.fisco.app.controller**：提供对外的api端点，负责处理 HTTP 请求，调用服务层，并返回响应给客户端。
3. **com.fisco.app.entity：**将数据实体组织为类

**在命令行发起HTTP请求调用服务接口（2分）**

根据demo教程指导，编译项目，并启动服务。在终端中或者其他工具发起HTTP请求到服务中，截图展示结果。



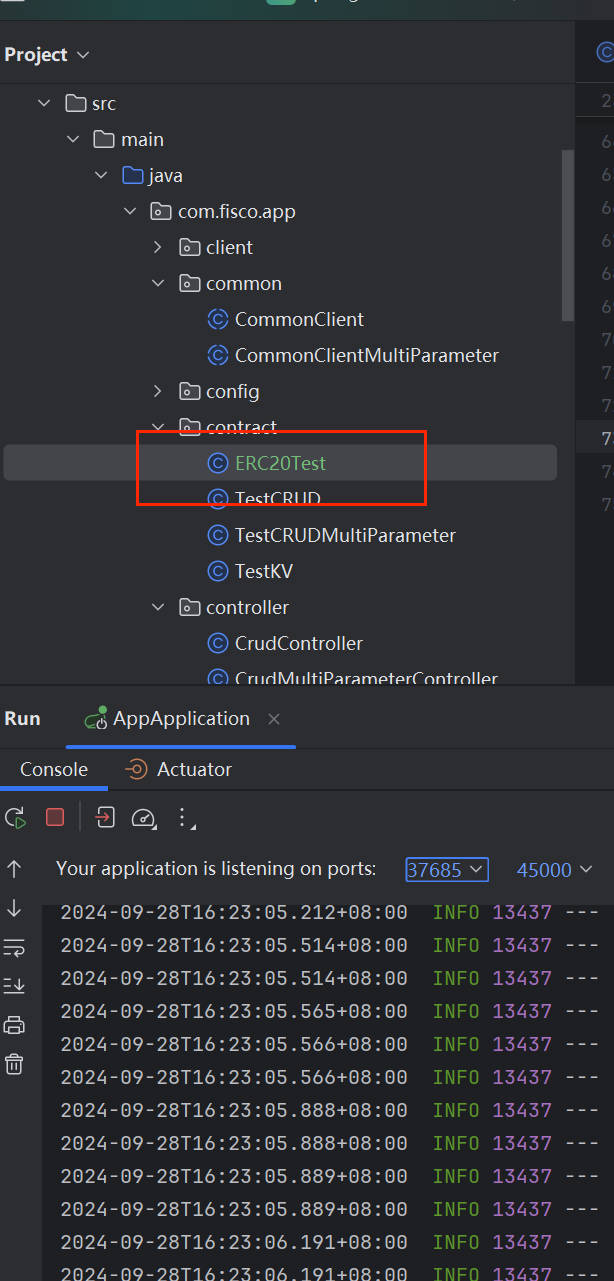


****

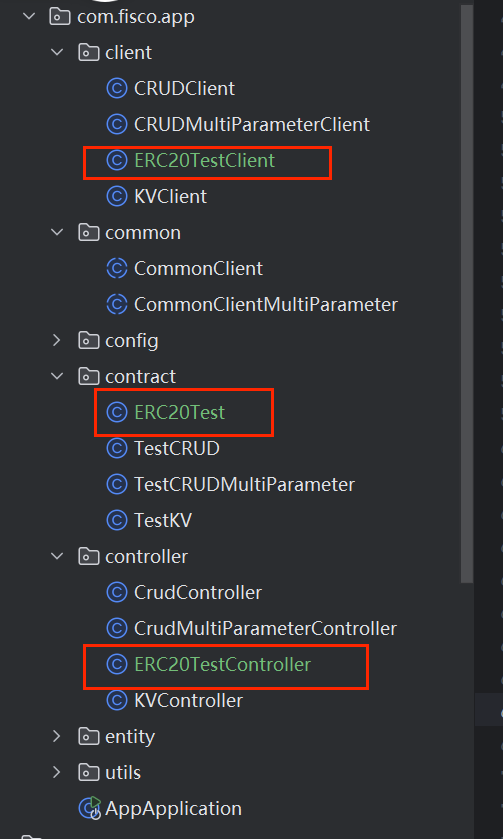
**Extra Bonus（非必做题）：仿造Spring-boot-crud的写法构建ERC20Test合约的Java服务（10分）**

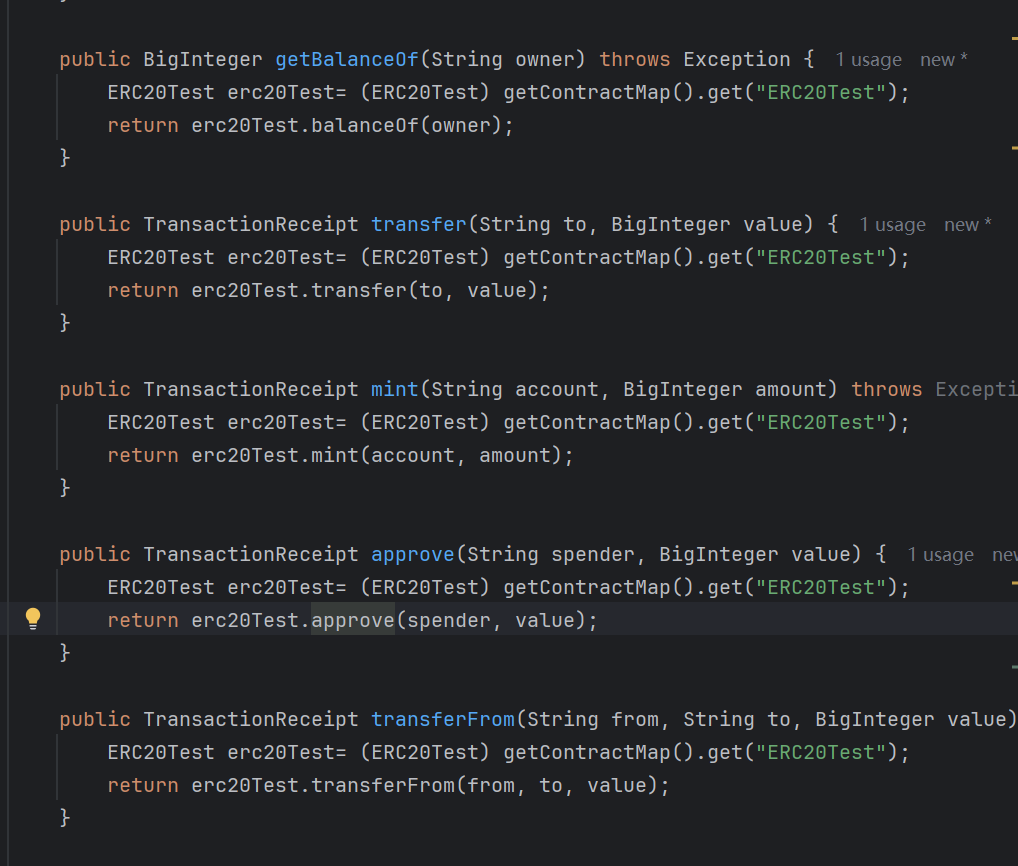
1. 使用本实验第一步骤生成的Java 接口文件ERC20Test.java，新增到spring-boot-crud项目中contract文件夹。（1分）

放置在相应文件夹下



1. 实现ERC20TestClient类，实现5个调用合约的基本接口：mint、transfer、approve、transferFrom、balanceOf。将这个类新增到项目client文件夹。（2分）

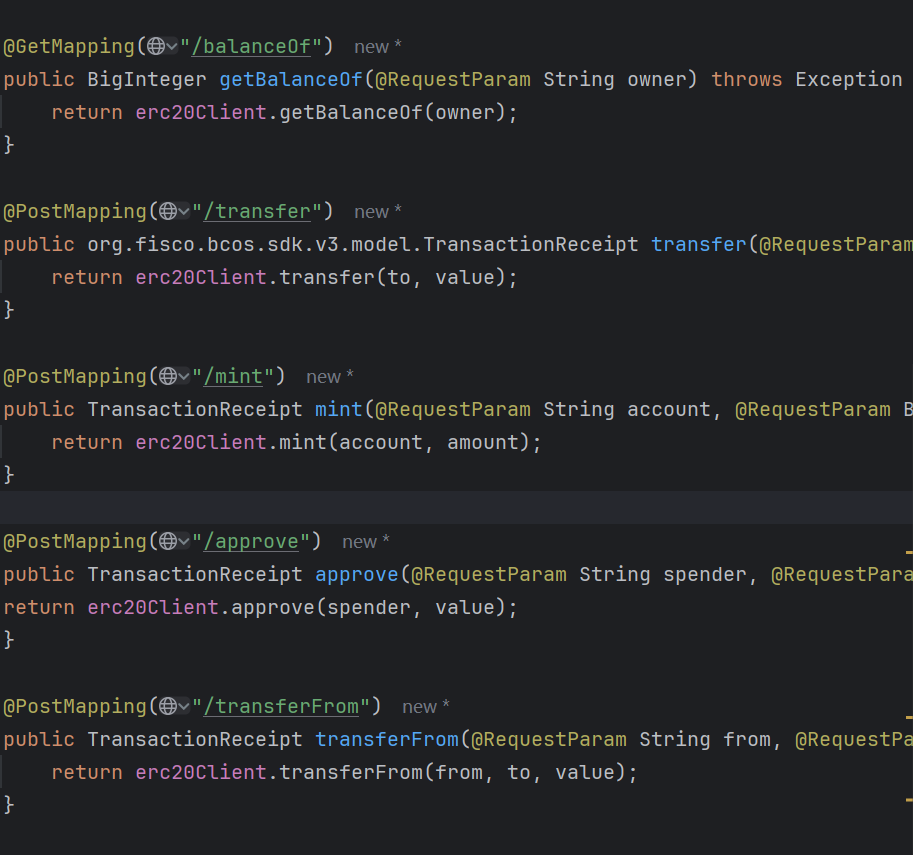




1. 实现ERC20TestController类，用于控制HTTP请求接口调用到ERC20TestClient的方法。将这个类新增到项目controller文件夹。（5分）

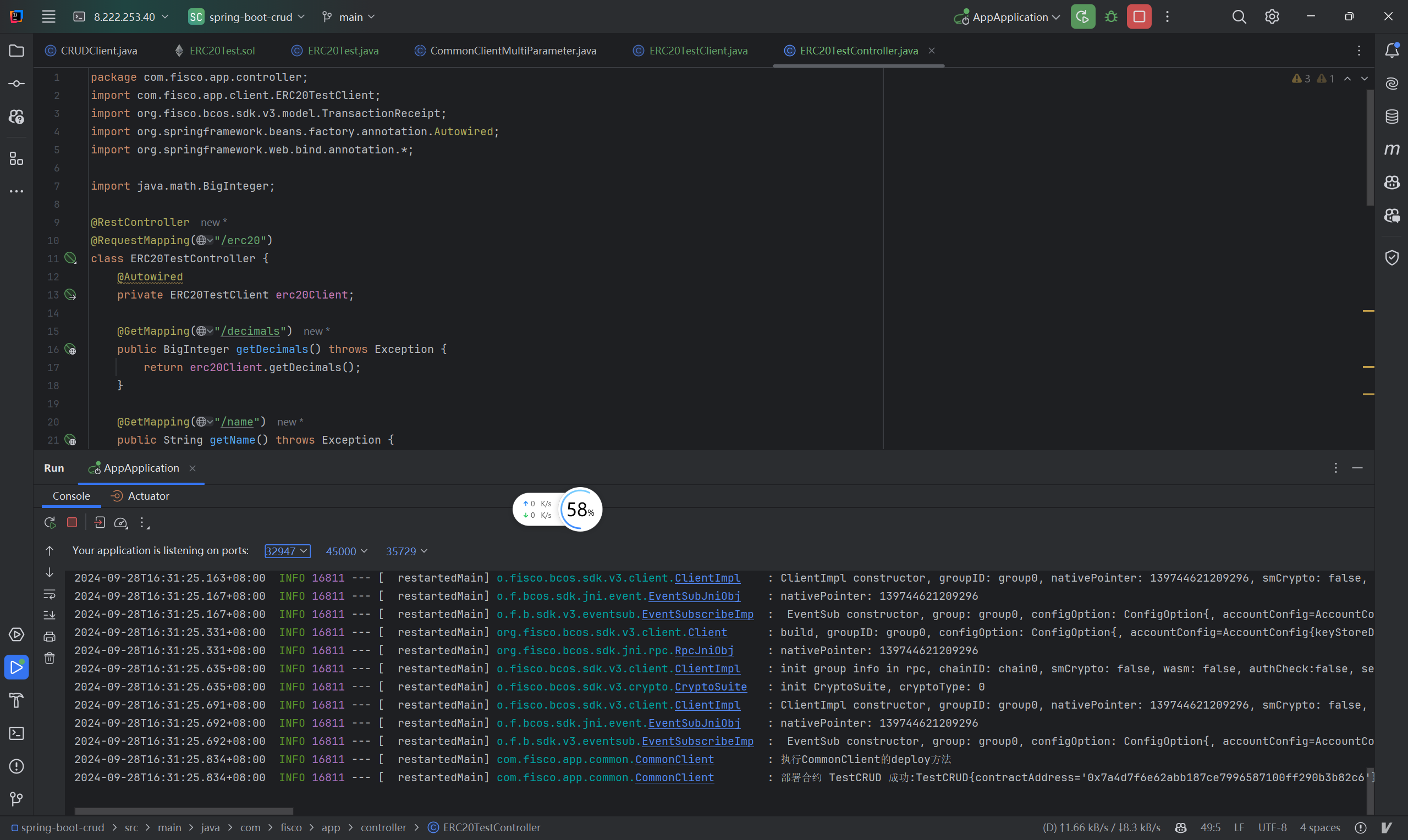


通过调用Client的service实现restapi如下

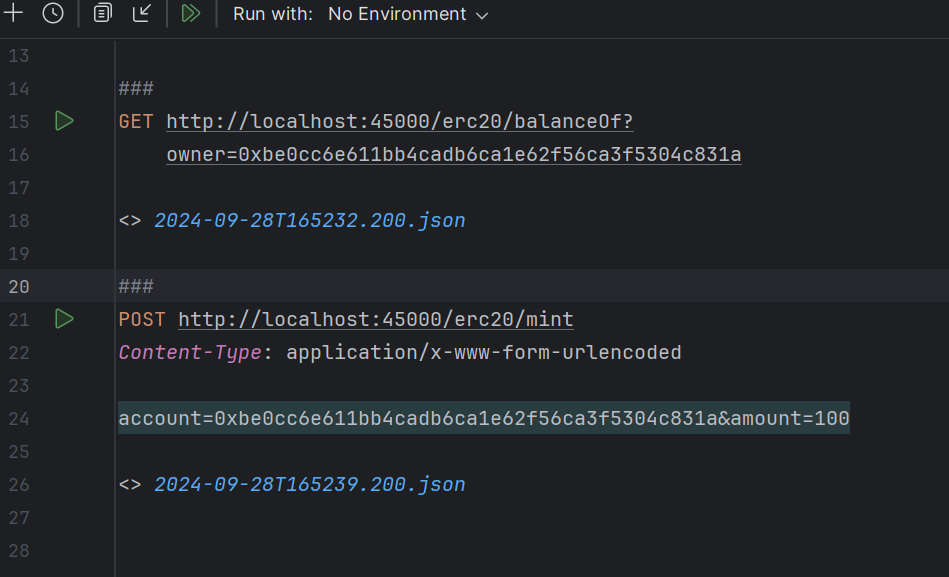


1. 尝试编译启动服务，并发起HTTP请求到新增接口中。（2分）

成功启动：



发起mint和获取余额：



Mint响应如下：



获取余额：

