说明文档

一、代码说明

1. dataType.sol:

在写合约的时候我主要创建了三个数据类型 —— project（项目）、contributor（贡献者）、fileContent（项目文件）结构体。

其中，project 结构体主要包含 id、name、creator（项目创建者的地址）、allContributors（项目所有贡献者的地址组成的数组）、weiPerContribution（购买每个贡献度所需的 wei）、minJoinContribution（想要加入项目所需购买的最少贡献度）、contributionPerLine（交换每行代码访问权限所需的贡献度）、totalContribution（项目的总贡献度）、profit（项目收益）、contributors（贡献度地址到贡献度结构体的映射）、codeFiles（文件名到项目文件结构体的映射）属性。

contributor 结构体包含 adr、personalContribution（贡献者的总贡献度）、unusedContribution（未执行贡献度）、lastBonusTime（上次分红时间）、obtainedBonus（已分配得到的收益）、isIn（贡献者是否在项目中）、isOwner（是否是项目创建者）、joinTime 属性。

fileContent 结构体包含 totalLength、content（文件内容，类型为 string 数组）、isExist（标识某项目下某文件名对应的文件是否存在）属性。

2. ContributionUsage.sol:

该部分实现了交换代码访问权限合约和利润分配合约这两个合约。

交换代码访问权限合约的主要实现代码在 ExchangeCodeAccess 方法中，如下:

```
/**
 * 交换代码访问权限
 */
function ExchangeCodeAccess(uint256 _projectId,string memory _fileName,uint256 startLine,uint256 lines)
public isExist(_projectId) returns(string[] memory _obtainedContent){
    project storage pro=projects[_projectId];
    require(pro.contributors[msg.sender].isIn,"Please join the project first");
    require(pro.codeFiles[_fileName].isExist,"The file does not exist");
    uint256 unusedContribution = pro.contributors[msg.sender].unusedContribution;
    uint256 neededContribution = lines * pro.contributionPerLine;
    require(unusedContribution >= neededContribution,"Insufficient unexecuted contributions");
    require(startLine+lines-1 <= pro.codeFiles[_fileName].totalLength,
    "The number of currently selected lines exceeds the file length limit");
    string[] memory obtainedContent = new string[](lines);
    for(uint256 fileContentIdx = 0; fileContentIdx < lines; fileContentIdx++){
        obtainedContent[fileContentIdx] = pro.codeFiles[_fileName].content[startLine+fileContentIdx-1];
    }
    pro.contributors[msg.sender].unusedContribution -= neededContribution;
    return obtainedContent;
}
```

参数依次为项目 id、文件名、起始行、总兑换行数。

首先通过 isExist 修改器判断该项目 id 对应的项目是否存在，再通过 require 判断当前调用该方法的用户是否已经加入项目以及文件名对应的项目是否存在。计算得要兑换所输入行数代码所需的贡献度，判断其是否小于当前贡献者拥有的未执行贡献度。如果贡献度足够，且最大访问行没有超过文件总行数，则从起始行对文件进行遍历，读取文件对应行代码存入变量 obtainedContent 中，读取完总兑换行数的代码后相应减少当前贡献者未执行贡献度，并且返回变量 obtainedContent。

利润分配合约主要实现代码在 profitDistribution 方法中，如下：

```
/**
 * 每次用户购买软件时，调用此方法分配利润
 * @param _projectID 项目id
 */
function profitDistribution(uint256 _projectID) public payable isExist(_projectID){
    project storage pro=projects[_projectID];
    address[] memory contributionAddr = pro.allContributors;
    uint256 proContri = pro.totalContribution;
    if(proContri>0){
    //给当前项目下的每一个贡献者分配利润
    for(uint i=0;i<contributionAddr.length;i++){
        //当前贡献者地址
        address curContriAddr = contributionAddr[i];
        //if(curContriAddr == pro.creator){
        //    continue;
        //}
        //当前贡献者总共贡献度
        uint256 personalContri = pro.contributors[curContriAddr].personalContribution;
        address payable _payableAddr = payable(curContriAddr);
        uint256 bounsAmount = msg.value * 8 / 10 * personalContri / proContri;
        (_payableAddr).transfer(bounsAmount);
        pro.contributors[curContriAddr].lastBounsTime = block.timestamp;
        pro.contributors[curContriAddr].obtainedBonus += bounsAmount;
    }
    //从项目的收益余额扣除已分配的80%的收入（假设所有收入在调用该方法前已存入项目收益余额）
    pro.profit -= msg.value * 8 / 10;
    }
}
```

参数只有项目 id。

首先也是通过 isExist 修改器判断该项目 id 对应的项目是否存在。循环遍历项目下的每一个贡献者，通过公式 msg.value * 8/10 * personalContri / proContri 计算每个贡献者可以分得的利润，其中 msg.value 即为用户购买项目软件所花以太币，personalContri / proContri 则为每个贡献者贡献度占项目总贡献度的比率。更新贡献者最后分红时间和已分配所得利润这两个变量。通过 transfer 方法给相应贡献者地址发送对应数目的以太币。最后从项目收益中减去分配给各贡献者的 80%的该次软件购买所得的收入（调用该利润分配方法前，该次软件软件的收入已存入项目收入）。

ContributionUsage.sol 中其他方法：

（1）createProject 方法：用于创建项目。通过合约中维护的全局变量 projectId 实现每创建一个项目 id 增加。给项目的 name 等属性赋值。将创建者地址加入项目贡献者中，并将 isOwner 变量赋值为 true。

```
//创建项目
function createProject(string memory _name, uint256 _weiPerContribution, uint256 _minJoinContribution,
 uint256 _contributionPerLine) public{
    uint256 id = projectId;
    projects[id].id = id;
    projects[id].name = _name;
    projects[id].creator = msg.sender;
    projects[id].weiPerContribution = _weiPerContribution;
    projects[id].minJoinContribution = _minJoinContribution;
    projects[id].contributionPerLine = _contributionPerLine;
    //把创建者加入项目
    projects[id].allContributors.push(msg.sender);
    projects[id].contributors[msg.sender].adr = msg.sender;
    projects[id].contributors[msg.sender].isIn = true;
    projects[id].contributors[msg.sender].isOwner = true;
    projects[id].contributors[msg.sender].joinTime = block.timestamp;
    projectId++;
}
```

（2）joinProject 方法：实现了通过购买软件的形式加入项目组。首先验证该用户是否已经加入项目，已加入则出错。其次再验证所购买贡献度是否不小于 minJoinContribution。在该方法中调用了利润分配方法。将成功购买软件的用户加入项目贡献者中，更新相应的属性值（isOwner 变量赋值为 false）。

```
//加入项目
function joinProject(uint256 _projectId) public payable isExist(_projectId) returns(address[] memory){
    project storage pro=projects[_projectId];
    require(pro.contributors[msg.sender].isIn==false,"You have joined the project");
    //第一种方式，通过购买软件加入项目
    uint256 contriAmount = msg.value/pro.weiPerContribution;
    require(contriAmount >= pro.minJoinContribution,"The purchase contribution is too less");
    payable(address(this)).transfer(msg.value);
    pro.profit += msg.value;
    //调用分配利润方法
    profitDistribution(_projectId);
    //更新projecet信息
    pro.allContributors.push(msg.sender);
    pro.totalContribution += contriAmount;
    pro.contributors[msg.sender].adr = msg.sender;
    pro.contributors[msg.sender].isIn = true;
    pro.contributors[msg.sender].isOwner = false;
    pro.contributors[msg.sender].joinTime = block.timestamp;
    pro.contributors[msg.sender].personalContribution += contriAmount;
    pro.contributors[msg.sender].unusedContribution += contriAmount;
    return pro.allContributors;
}
```

（3）addFile 方法：用于在项目中增加文件，只有项目创建者可以增加文件。

```
function addFile(uint256 _projectId,string memory _fileName,string[] memory _fileContent) public isExist(_projectId){
    project storage pro = projects[_projectId];
    require(pro.contributors[msg.sender].isOwner,"Only owner can execute this method");
    require(pro.codeFiles[_fileName].isExist==false,"The file name already exists");
    pro.codeFiles[_fileName].totalLength = _fileContent.length;
    pro.codeFiles[_fileName].content = _fileContent;
    pro.codeFiles[_fileName].isExist = true;
}
```

（4）purchaseContribution 方法：用于项目贡献者购买贡献度。未加入项目的用户调用此方法会报错。

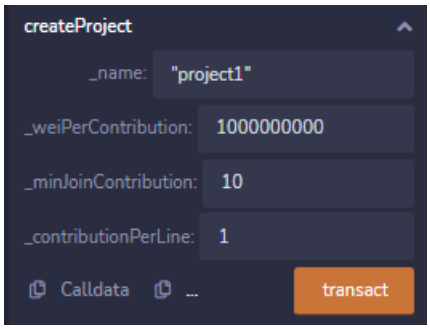
```
//购买贡献度
function purchaseContribution(uint256 _projectId) public payable isExist(_projectId){
    project storage pro=projects[_projectId];
    require(pro.contributors[msg.sender].isIn,"Please join the project first");
    uint256 contriAmount = msg.value/pro.weiPerContribution;
    payable(address(this)).transfer(msg.value);
    pro.profit += msg.value;
    pro.totalContribution += contriAmount;
    pro.contributors[msg.sender].personalContribution += contriAmount;
    pro.contributors[msg.sender].unusedContribution += contriAmount;
}
```

（5）其他 get 方法：用于在调用时展示信息。

二、部署及调用说明

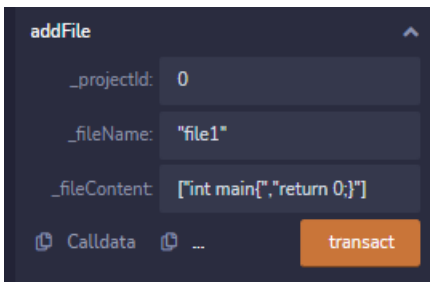
部署调用步骤及截图：

部署后，首先调用 create 方法创建项目，如图：

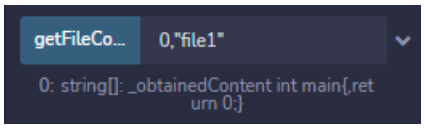


调用 get 方法获取项目信息进行验证：

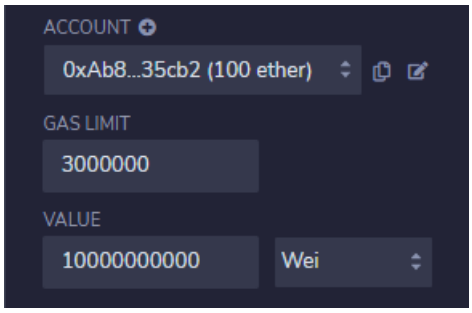
调用 addFile 方法增加文件：



调用相应 get 方法验证：

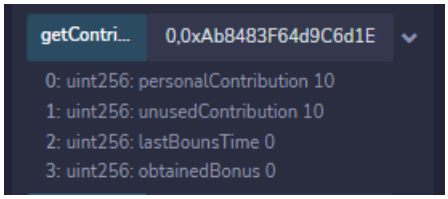


切换新账户 1，设置 value，调用 joinProject 方法：



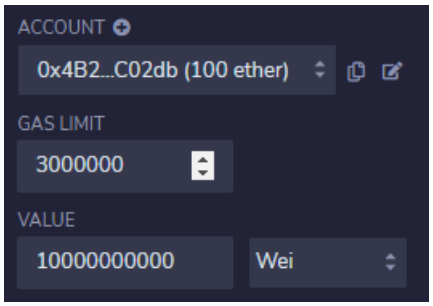
合约余额变化：

调用相应 get 方法验证贡献者信息：



未执行贡献度和总贡献度都为 10

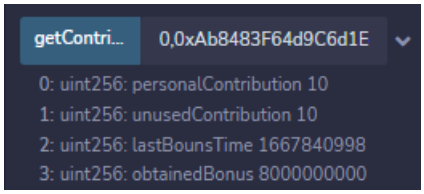
再切换新账户 2，调用 joinProject 方法加入项目：



因为在 joinProject 方法中调用了利润分配方法，所以合约余额也发生变化：

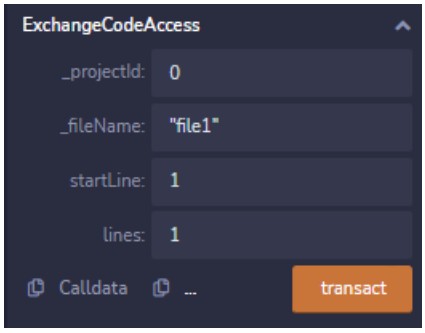
80%的收入被分配，所以合约余额只增加 0.000000002ETH

新账户 1 的已分得利润和最后分红时间也发生变化：

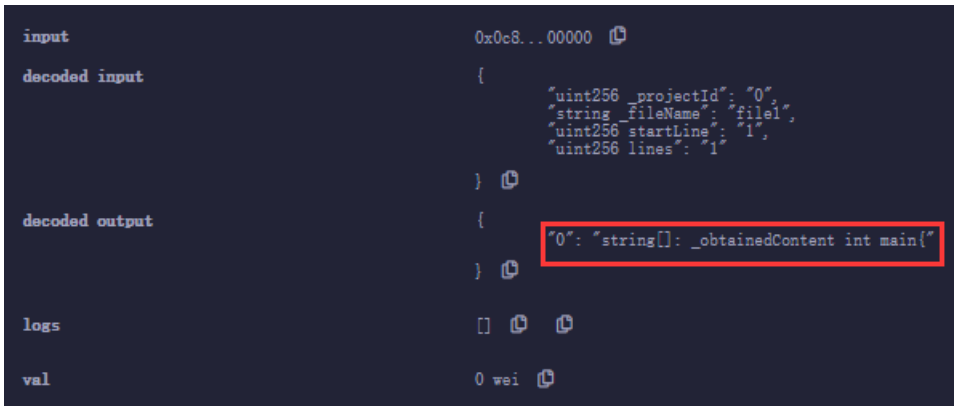


因为新账户 2 加入时，当前合约只有新账户 1 和合约创建者两个贡献者，而合约创建者没有贡献度，新账户 1 贡献度占项目总贡献度的比率为 100%，所以，所有 80%的收入被分配给新账户 1，新账户 1 已分得利润增加 8000000000，即 0.000000008ETH。

新账户 1 调用 ExchangeCodeAccess 方法：



返回如下，成功获取文件 file1 第一行代码



调用 get 方法验证，新账户 1 的未执行贡献度也相应减少：