角色分配

管理者

- 通常为众包系统发起者
- 发布合约、更新合约

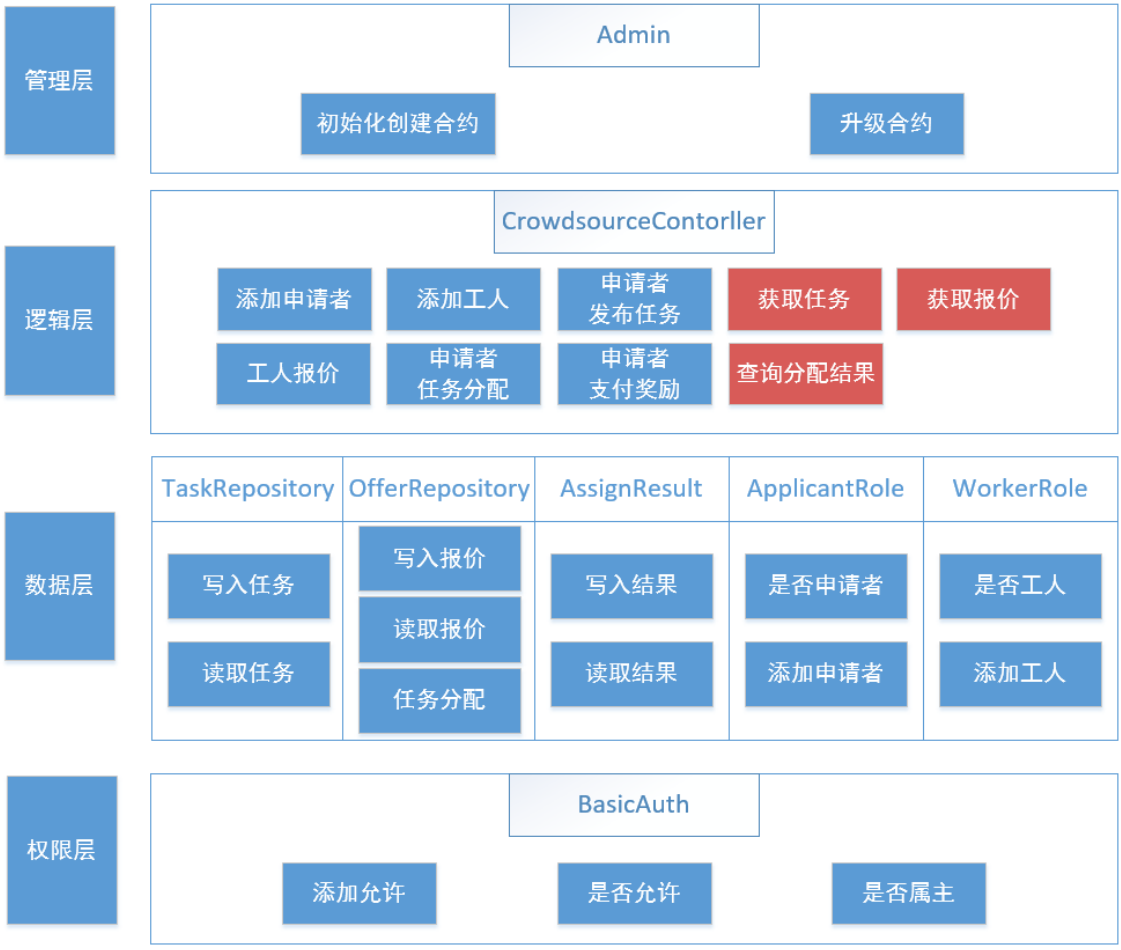
申请者

- 发布任务、分配任务、奖励支付

工人

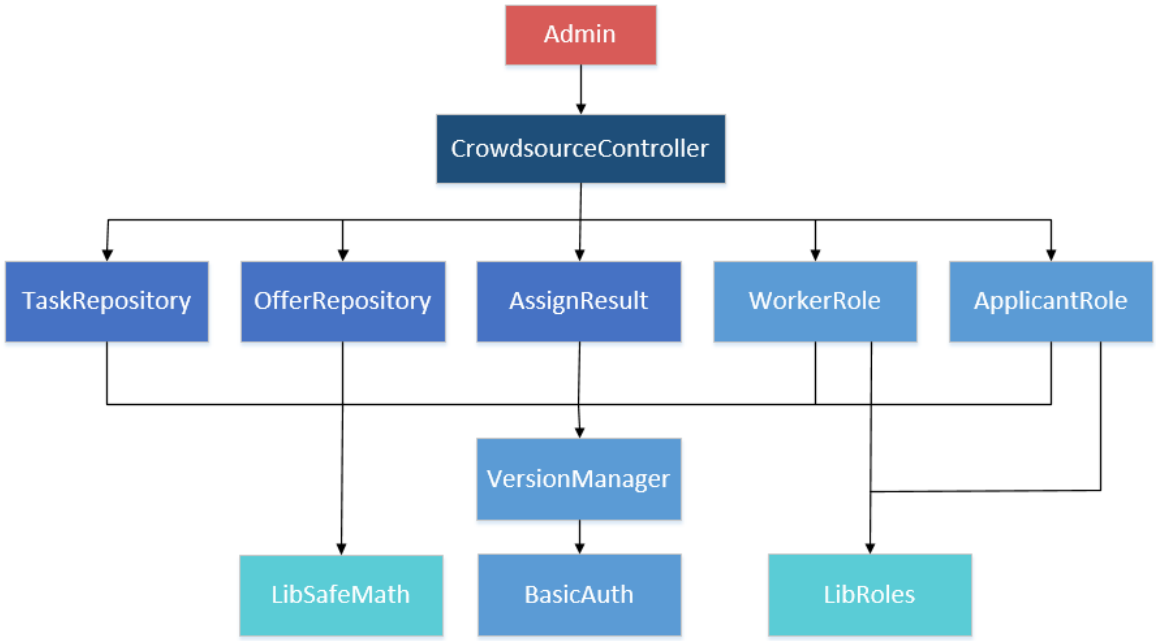
- 参与报价

合约层次结构



- **权限层:** 控制合约给指定授权用户访问。
- **数据层:** 存储合约相关数据，包括任务信息、工人报价、分配结果、申请者信息和工人信息。
- **逻辑层:** 提供对区块链中数据的查询和写入，蓝色代表写交易，红色代表读交易。
- **管理层:** 创建控制合约和数据合约，通过设置数据合约中控制合约的地址来实现合约升级。

合约关系图



- **Admin:** 初始化生成合约和升级合约，控制访问数据层合约的地址。
- **CrowdsourceController:** 为上层提供接口的控制器。
- **TaskRepository OfferRepository AssignResult:** 存储业务合约相关的数据。
- **WorkerRole ApplicantRole BasicAuth VersionManager:** 权限、角色、版本管理的工具合约。
- **LibSafeMath LibRoles:** 安全计算库和角色管理库。

逻辑层接口

逻辑层合约CrowdsourceController.sol包含所有提供给上层的接口。

合约调用（数据写入相关）

- function addApplicant() public

  addApplicant用于添加申请者，将调用该合约的用户设置为申请者。当用户信息不存在时，将用户信息存储到区块链中，否则提示报错。

- function addWorker() public

  addApplicant用于添加工人，将调用该合约的用户设置为工人。当用户信息不存在时，将用户信息存储到区块链中，否则提示报错。

- function releaseTask(string taskname, uint duration, string taskBrief, uint expectedPrice) onlyApplicant public

  releaseTask用于发布任务，合约调用者必须为申请者，输入参数为任务名，持续时间，任务概要，预期价格。当任务未被发布时，将任务信息存储到区块链中，否则提示报错。

- function workerOffer(string taskname, uint duration, uint cost) onlyWorker public

  workerOffer用于工人发送报价请求，合约调用者必须为工人，输入参数为任务名，持续时间，报价。当任务未被分配且工人未请求该任务时，将请求信息存储到区块链中，否则提示报错。

- function taskAssignment(string taskname) onlyApplicant public returns(string Taskname, address assignor, uint cost)

  taskAssignment用于任务分配，合约调用者必须为该任务的发布者，输入参数为任务名。当任务未被分配且存在工人请求时，系统会将任务分配给报价最低的工人，计算临界价值作为工人的奖励，并把该分配结果存储区块链中，否则提示错误。

- function BonusPayment(string taskname, bool IsQualified) onlyApplicant public

  BonusPayment用于奖励支付，合约调用者必须为该任务的发布者，输入参数为任务名和是否合格。若任务不合格，系统会扣除工人的信用积分，且工人获取的奖励与信用积分成正比。当任务对应的分配结果存在时，系统会进行账户余额的转移，否则提示错误。

接口	参数	返回值	描述
addApplicant	无	无	添加本用户为申请者
addWorker	无	无	添加本用户为工人
releaseTask	taskname(string), duration(uint), taskBrief（string）, expectedPrice（uint）	无	发布任务
workerOffer	taskname(string), duration(uint), cost(uint)	无	工人报价
taskAssignment	taskname(string)	string, address, uint	任务分配
BonusPayment	taskname(string), IsQualified（bool）	无	奖励支付

合约调用（数据查询相关）

- function QueryApplicant() onlyApplicant public view returns(address, uint balance)

  QueryApplicant用于查询调用该合约的申请者的信息，不需要参数。

- function QueryWorker() onlyWorker public view returns(address, uint balance, uint creditScore)

  QueryWorker用于查询调用该合约的工人的信息，不需要参数。

- function QueryTask(string taskname) public view returns(string, address, uint, uint, string, uint)

  QueryTask通过任务名查询系统中的任务信息，输入参数为任务名。

- function QueryAllMyTask() public returns(string[])

  QueryAllMyTask用于查询合约调用者发布的所有任务，不需要参数。

- function QueryAllTask() public view returns(string[])

  QueryAllTask用于查询系统中发布的所有任务信息，不需要参数。

- function QueryMyOffer(string taskname) onlyWorker public returns(uint, uint, uint, address, string)

  QueryMyOffer用于查询合约调用者对某个任务的请求信息，输入参数为任务名。

- function QueryAllMyOffer() onlyWorker public returns(string[])

  QueryAllMyOffer用于查询合约调用者的所有任务请求信息，不需要参数。

- function QueryResult(string taskname) public returns(string, address, uint)

  QueryResult用于查询任务分配结果信息，输入参数为任务名。

接口	参数	返回值	描述
QueryApplicant	无	address, uint	查询本申请者信息
QueryWorker	无	address, uint, uint	查询本工人信息
QueryTask	taskname(string)	string, address, uint, uint, string, uint	查询某个任务
QueryAllMyTask	无	string[]	查询本申请者发布的所有任务
QueryAllTask	无	string[]	查询系统中发布的所有任务
QueryMyOffer	taskname(string)	uint, uint, uint, address, string	查询本工人对某个任务的请求
QueryAllMyOffer	无	string[]	查询本工人的所有请求
QueryResult	taskname(string)	string, address, uint	查询某个任务的分配结果

部分合约说明

申请者合约

```
contract ApplicantRole is VersionManager{
    using LibSafeMath for uint256;
    using LibRoles for LibRoles.Role;
    LibRoles.Role private _applicants;
    mapping(address => uint) private _balances;

    event addApplicantEvent(address indexed account);

    function isApplicant(address account) public view returns(bool){
        return _applicants.has(account);
    }

    function addApplicant(address account) onlyLatestVersion public {
        _applicants.add(account);
        _balances[account] = 1000;
        emit addApplicantEvent(account);
    }
```

添加申请者

工人合约

```
contract WorkerRole is VersionManager{
    using LibSafeMath for uint256;
    using LibRoles for LibRoles.Role;
    LibRoles.Role private _workers;
    mapping(address => uint) private _balances;      //账号余额
    mapping(address => uint) private _creditScore;   //用户信用积分

    event AddworkerEvent(address indexed account);

    function isWorker(address account) public view returns(bool){
        return _workers.has(account);
    }

    function addWorker(address account) onlyLatestVersion public {
        _workers.add(account);
        _balances[account] = 1000;
        _creditScore[account] = 100;
        emit AddworkerEvent(account);
    }
```

添加工人

任务数据合约

```
contract TaskRepository is VersionManager{
    struct Task{
        string taskname;        //任务名
        address owner;          //任务发布者
        uint duration;          //任务持续时间
        uint startTime;         //任务发布时间
        string taskBrief;       //任务简介
        uint expectedPrice;     //预期价格
        bool IsEnd;             //任务是否结束
        bool IsPaid;            //是否给予奖励
    }
    mapping(string=>Task) private _Tasks;    //任务集
    string[] private _Tasknames ;            //任务名
    string[] private mytasknames;            //某申请者的任务名

    function setData(string taskname, uint duration, string taskBrief, uint expectedPrice) onlyLatestVersion public{
```

存储数据结构

限定版本

报价数据合约

```
contract OfferRepository is VersionManager
{
    using LibSafeMath for uint256;

    struct Offer{
        uint arrivalTime;     //到达时间
        uint departureTime;   //离开时间
        uint cost;            //报价
        address owner;        //工人名
        string taskname;      //任务名
    }
    mapping(string=>Offer[]) private _TaskOffers;
    mapping(address=>string[]) private _workerTasknames;

    function setData(string taskname, uint duration, uint cost) onlyLatestVersion public{
```

存储数据结构

分配结果合约

```
contract AssignResult is VersionManager
{
    struct Result{
        string taskname;      //任务名
        address assgignor;    //被分配者
        uint cost;            //奖励支付
    }
    Result[] private _Results;

    function setData(string taskname, address assgignor, uint cost) onlyLatestVersion public{
        Result memory resulttemp = Result(taskname, assgignor, cost);
        _Results.push(resulttemp);
    }
```

存储数据结构

版本管理合约

```
contract VersionManager is BasicAuth
{
    address _latestVersion;          //最新版本(controller)
    modifier onlyLatestVersion(){
        require(msg.sender == _latestVersion, "Not latestVersion");
        _;
    }
    function upgradeVersion(address newVersion) public{
        require(msg.sender == _owner, "Not owner");
        _latestVersion = newVersion;
    }
}
```

更新版本

控制合约

```
//--------3.申请者发布任务------------
//input: 任务名，任务持续时间
function releaseTask(string taskname, uint duration, string taskBrief, uint expectedPrice) onlyApplicant public{
    _task.setData(taskname, duration, taskBrief, expectedPrice);
    emit ReleaseTaskEvent(taskname, msg.sender, duration, now, taskBrief, expectedPrice);
}

//------- 4.工人报价----------------
//input: 任务名，报价持续时间，报价
function workerOffer(string taskname, uint duration, uint cost) onlyWorker public{
    //判断任务是否结束
    bool IsEnd;
    (,,,,,,IsEnd,) = _task.getData(taskname);
    require(!IsEnd,"The task has already ended!");

    _offer.setData(taskname, duration, cost);
    emit WorkerOfferEvent(now, now.add(duration*3600), cost, msg.sender, taskname);
}
```

限定申请者

限定工人

管理合约

```
constructor() public{
    TaskRepository task = new TaskRepository();
    OfferRepository offer = new OfferRepository();
    AssignResult result = new AssignResult();
    ApplicantRole applicant = new ApplicantRole();
    WorkerRole worker = new WorkerRole();
    _taskAddress = address(task);
    _offerAddress = address(offer);
    _resultAddress = address(result);
    _applicantAddress = address(applicant);
    _workerAddress = address(worker);

    CrowdsourceController controller = new CrowdsourceController(_taskAddress, _offerAddress,
     _resultAddress, _applicantAddress, _workerAddress);
    _controllerAddress = address(controller);

    task.upgradeVersion(_controllerAddress);
    offer.upgradeVersion(_controllerAddress);
    result.upgradeVersion(_controllerAddress);
    applicant.upgradeVersion(_controllerAddress);
    worker.upgradeVersion(_controllerAddress);
}
```

创建数据层合约

创建控制合约

```
function upgradeVersion(address newVersion) public onlyOwner{
    TaskRepository task = TaskRepository(_taskAddress);
    OfferRepository offer = OfferRepository(_offerAddress);
    AssignResult result = AssignResult(_resultAddress);
    ApplicantRole applicant = ApplicantRole(_applicantAddress);
    WorkerRole worker = WorkerRole(_workerAddress);

    task.upgradeVersion(newVersion);
    offer.upgradeVersion(newVersion);
    result.upgradeVersion(newVersion);
    applicant.upgradeVersion(newVersion);
    worker.upgradeVersion(newVersion);
}
```

升级合约

合约使用指南

前提条件

环境	版本
FISCO BCOS	2.3.0
WeBASE-Front	v1.3.0
Solidity	0.4.4

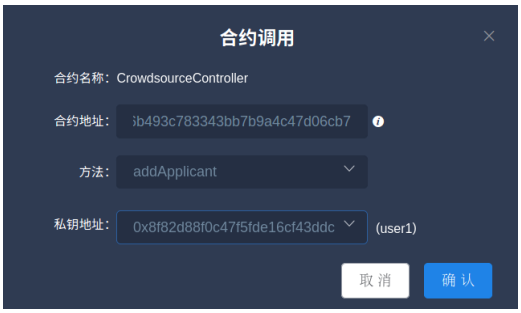
合约部署

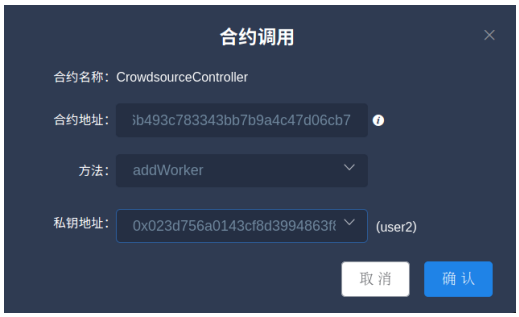
在WeBASE-Front上部署时所有合约放在同一目录下，管理者只需部署Admin.sol，然后调用Admin合约中_controllerAddress，可得到CrowdsourceContorller合约的地址，即可调用相关方法。

合约测试

在测试样例中，注册了三个用户，user1，user2，user3 ，user1为任务发布者，user2，user3为工人。user1发布了三个任务分别是task1，task2，task3，user2和user3分别对这三个任务发起了请求，user1在接受到请求后对任务进行了分配和奖励支付。测试流程如下：

- 用户注册





- user1发布任务

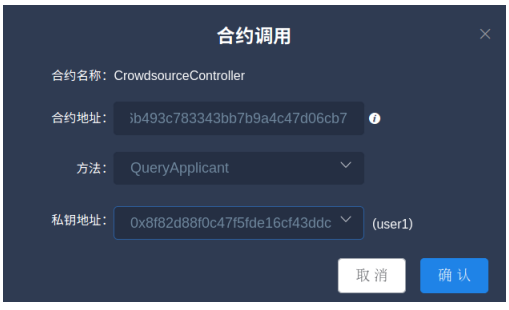
- user2和user3发送报价请求

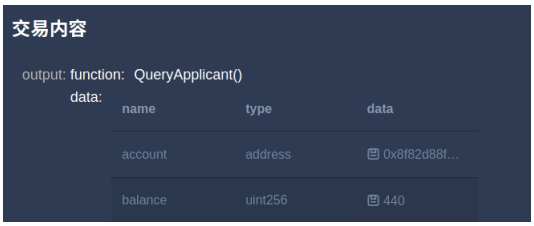
- user1分配任务

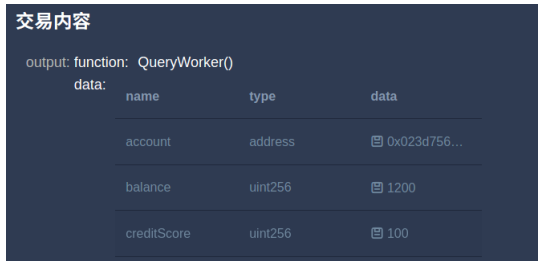
- user1验收任务并支付奖励

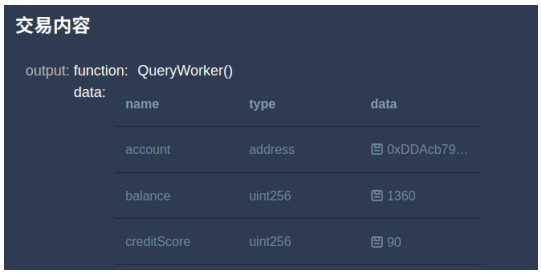
对区块链中的数据进行查询，结果如下：

- 查询申请者和工人信息

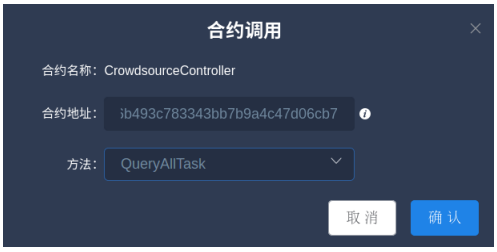


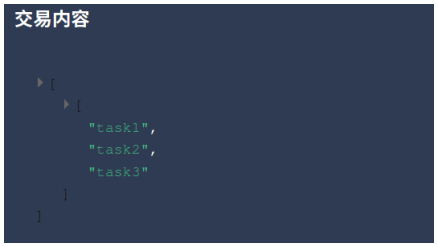






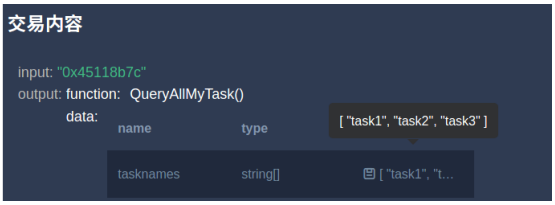
- 查询系统中所有任务



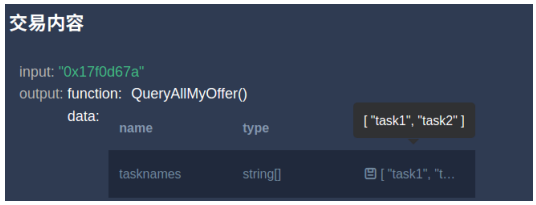


- 查询任务信息（以task1为例）

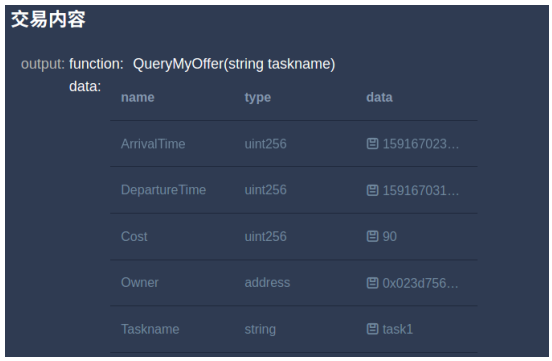
- 查询个人发布的所有任务（以user1为例）



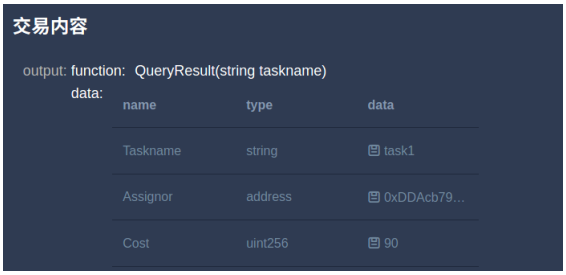
- 查询个人发布的请求（以user2为例）



- 查询个人对某个任务的请求（以user2为例）



- 查询某个任务的分配结果（以task1为例）



合约版本更新

管理员调用Admin合约的upgradeVersion方法来更新控制合约，输入参数为新的控制合约的地址。