# 区块链投票系统

## 目的及意义

1. 了解区块链、以太坊、比特币等相关概念；
2. 学习使用了solidity合约编写、remix开发工具开发合约；
3. 完成对matrix、aiman.js中SDK/API功能的测试；
4. 了解DAPP开发基本流程。

## DAPP功能实现

基于区块链技术的投票系统，达到了投票的不可篡改、安全透明和匿名性。

### DAPP功能逻辑：



1. 创建投票（对投票系统候选人进行初始化）；

在区块链上创建候选人

1. 投票查询

对链上所记录的数据做相应的查询

1. 投票

在链上对相应数据的修改

### 功能合约编写

1. 创建投票数据集合存储数据
2. *// 创建名为candidates的bytes32数组*
3. bytes32 [] candidates;
4. *// 创建名为candidatesVotingCount的映射，键为bytes32值为uint*
5. mapping(bytes32 => uint) candidatesVotingCount;
6. *// 初始化合约，创建候选人数组*
7. function createVoting(bytes32[] memory \_candidates) public {
8. candidates = \_candidates;
9. }
10. 增加候选人票数
11. function votingToPerson(bytes32 \_person) public {
12. *// 映射中候选人票数自加1*
13. candidatesVotingCount[\_person] += 1;
14. }
15. 通过映射集合查询候选人总票数
16. function votingTotalPerson(bytes32 \_person) public view returns(uint) {
17. *// 返回映射中候选人票数*
18. return candidatesVotingCount[\_person];
19. }
20. 考虑投票时已创建映射中是否存在候选人，若不存在，则中断投票
21. function isValidToPerson(bytes32 \_person) private view returns(bool) {
22. for(uint i = 0; i < candidates.length; i++) {
23. if(candidates[i]==\_person) {
24. return true;
25. }
26. }
27. return false;
28. }
29. *// 投票时或查询票数时应进行断言操作；断言是否为真，否则中断执行*
30. require(isValidToPerson(\_person));

### 项目搭建、运行开发

1. 本地合约编译

安装truffle

npm install -g truffle

安装集成项目

truffle unbox react或vue等

合约开发

truffle develop

合约编译

compile

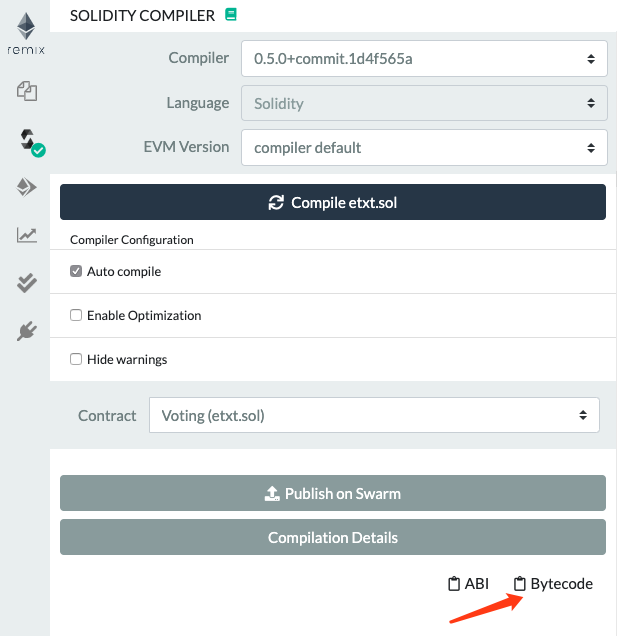
1. 线上编译工具编译

remix [http://remix.ethereum.org](http://remix.ethereum.org/)

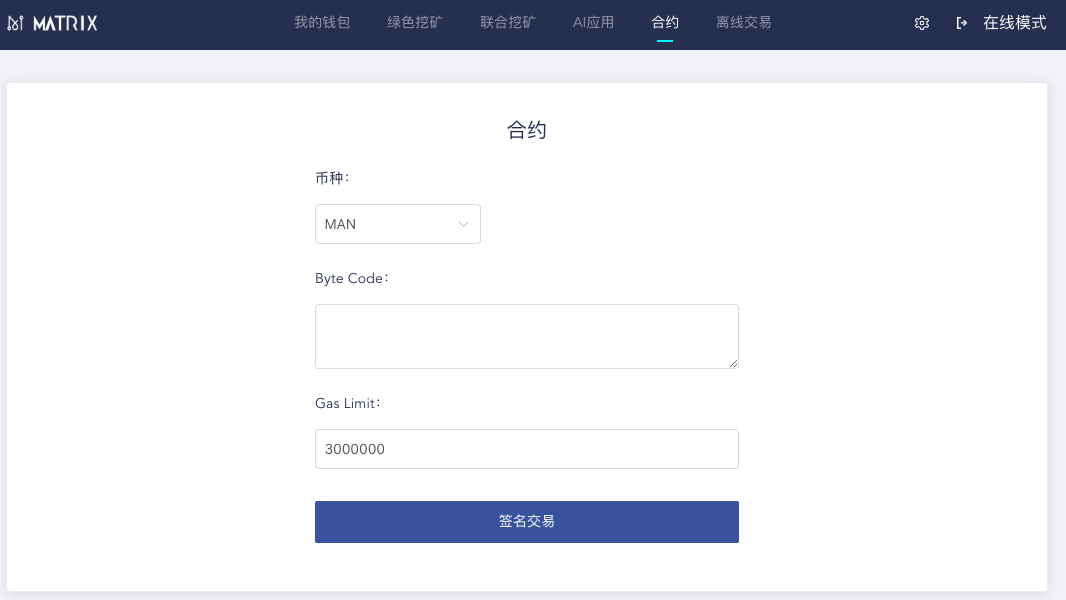
通过remix编译即可完成对ABI和Bytecode的获取

### 部署合约,获取合约地址

1. 通过remix编译获取Bytecode



1. 将Bytecode中键为object的值（可执行代码）放入钱包签署合约（Byte Code）当中



1. 签名交易获取MAN地址跟hash地址；



### 合约功能及API调用

1. 将获取到的合约地址跟hash地址应用于代码中

*// 合约MAN地址*

const contractManAdd = "MAN.3f1qSSMib1UVHTaBUSWoGgZVpwa3r";

*// 合约hash地址*

const contractAddress =

"0x3050cca9dbe0f0d4f1d0288ad47f5ce71adc9f3fbecfa9347baf60a45d410636";

1. 代码实现
2. 初始化一个aiman.js的实例，环境地址为<https://testnet.matrix.io>

let httpProvider = new HttpProvider("https://testnet.matrix.io");

1. 初始化一个可调用的合约

let contract = httpProvider.man

.contract(ABI)

.at(contractAddress);

1. 初始化合约中候选人数组，调用contract.方法.getData()后获取到本

次交易的可执行代码；

let runningCode = contract.createVoting.getData(names);

1. 上链产生交易时先调用man.getTransactionCount获取nonce值

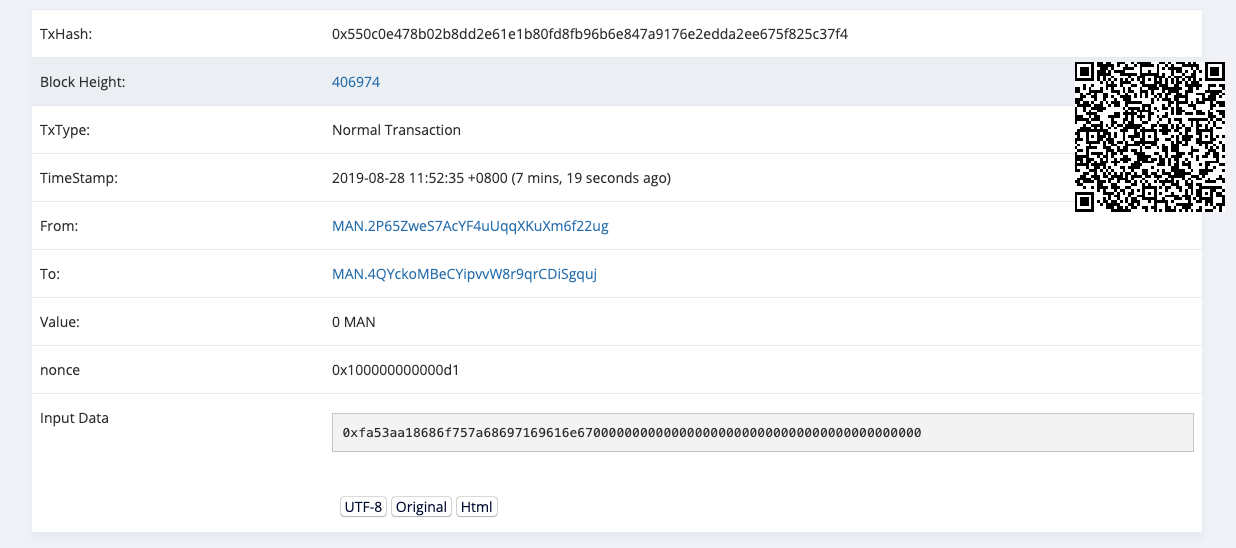
httpProvider.man.getTransactionCount(address);

1. 调用man.sendRawTransaction, 会产生当前操作需要上链的hash，将此

笔交易发送到节点上，准备上链。对链上数据修改、写入时调用该方法。

httpProvider.man.sendRawTransaction(rawTx);

1. 将hash复制到<https://jerry.matrix.io/home>中搜索查询此次交易是否成功上链，如果成功上链，则查询结果如下



1. 点击投票，调用投票方法对某个候选人进行投票

let votingCode = contract.votingToPerson.getData(name);

除执行代码不同，流程与初始化投票一致；

1. 投票成功后，查询投票。调用man.call查询数据。对链上数据进行查询

（无法修改、写入）时调用该方法。

httpProvider.man.call({

to: contractManAdd,

data: contract.votingTotalPerson.getData(name),

currency: "MAN"

};

);

1. 返回值为一个hex的十六进制值

例0x0000000000000000000000000000000000000000000000000000000000000003

调用aiman.toDecima将十六进制转为一个十进制的数字

httpProvider.toDecimal(resultData)

最终将获取到的投票数展示到列表中，查询所有人的投票数是，遍历查询每个人即可展示。

## 交易上链过程（matrix）

1. 合约/执行代码作为交易发送至某个节点；
2. 节点通过洪泛（向其他节点广播此交易）直至广播到矿工节点。
3. 矿工节点通过随机选举+轮流出块部分矿工获得出块权，将此笔交易打包上

链。

注：此处矿工节点获得出块权与以太坊不同，以太坊则为所有矿工均有出块权，一段时间内只有一人可以获得出块权（通过pow获得出块权）

## 交易费用

1. 在公有链上, 任何人都可以读写数据。读取数据是免费的, 但是向公有链中写数据时需要花费一定费用的, 这种开销有助于阻止垃圾内容, 并通过支付保护其安全性。
2. 网络上的任何节点(每个包含账本拷贝的连接设备被称作节点) 都可以参与称作挖矿的方式来保护网络。由于挖矿需要计算能力和电费, 所以矿工们的服务需要得到一定的报酬, 这也是矿工费的由来。
3. 以太坊和比特币的不同之处，以太坊引入了 gas 的概念，gas的目的是限制执行交易所需的工作量，同时为执行支付费用。gas 用来衡量你的这笔交易（或者合约代码调用）所消耗的资源（包括计算量，存储，带宽等）。
4. Gas Fee = Gas Limit x Gas Price

Gas Fee：矿工费总金额

Gas Limit：单个区块允许的最多gas总量

Gas Price：每个单位Gas 的价格

1. 矿工所得：发币机构奖励+矿工费