用正确的姿势开发以太坊系列

【一个基于以太坊的投票小应用】

功夫小猫

tanzhiguo@cn.ibm.com

计划要写四篇关于以太坊开发的文章，这是第二篇。

```
//code from : https://gist.githubusercontent.com/maheshmurthy/3da385a42678c3e36a8328cbe47cae5b/raw/82f9d3b884841ab381b3ea86c143281c65dab89b/Voting.sol
pragma solidity ^0.4.18;

contract Voting {
  mapping (bytes32 => uint8) public votesReceived;
  bytes32[] public candidateList;

  function Voting(bytes32[] candidateNames) public {
    candidateList = candidateNames;
  }
  function totalVotesFor(bytes32 candidate) view public returns (uint8) {
    require(validCandidate(candidate));
    return votesReceived[candidate];
  }
  function voteForCandidate(bytes32 candidate) public {
    require(validCandidate(candidate));
    votesReceived[candidate]  += 1;
  }
  function validCandidate(bytes32 candidate) view public returns (bool) {
    for(uint i = 0; i < candidateList.length; i++) {
      if (candidateList[i] == candidate) {
        return true;
      }
    }
    return false;
   }
}
```

关于投票的这个智能合约的代码，来源于网络，代码仅仅 25 行，也很容易理解，candidateList 的定义主要是为了校验候选人是否存在而设计的数据结构，对于投票并没有特别实际的作用，合约并不是这篇文章要讲的重点，重点我们来看看怎么通过两种方式来执行这个合约。

参照上一篇文章的图表，我们这次采用 Ganache 模拟以太坊节点，启动 Gnanache 之后，默认开启 8545 这个端口，默认会生成 10 个 Account，而且每一个都默认带有 ether。

合约的编译，这次我们采用 solc，一个 npm 的实现，使用 web3 部署合约，ejs 主要是为了测试如何通过网页上的操作来执行智能合约，

app.js

```
const fs = require('fs');
const Web3 = require('web3');
const solc = require('solc');

const httpProvider = "http://localhost:8545";
const mmartContract = "Voting.sol";
const defaultCandidates = ['Rama', 'Nick', 'Jose'];

web3 = new Web3(new Web3.providers.HttpProvider(httpProvider));

console.log('Accounts list:\n');
console.log(web3.eth.accounts);
console.log('\n');
```

solc 编译后，就可以通过 web3 进行 deploy 了，我们可以看到 deploy 的形式和我们通过 Remix 在线执行的方式差不多，其中 gas 是可以通过 web3 进行计算的，

```
let source = fs.readFileSync(mmartContract, 'utf8').toString();
const defautAccount = web3.eth.accounts[0];

solc.loadRemoteVersion('latest', function(err, solcSnapshot) {
    if (err) {
        console.log('error:', err);
    }
    let compiledContract = solcSnapshot.compile(source, 1);
    let abi = compiledContract.contracts[':Voting'].interface;
    let contractCode = compiledContract.contracts[':Voting'].bytecode;
    let votingContract = web3.eth.contract(JSON.parse(abi));
    let contractData = votingContract.new.getData(defaultCandidates, { data: contractCode });
    let gasEstimate = web3.eth.estimateGas({ data: contractData });

    console.log('Us gas:', gasEstimate);
    console.log('\n');

    var deployedContract = votingContract.new(defaultCandidates, {
        data: contractCode,
        from: defautAccount,
        gas: gasEstimate
    }, function(err, voteContract) {
        //NOTE: The callback will fire twice
        if (!err) {
            if (!voteContract.address) {
                //console.log('error: ', voteContract.transactionHash)
            } else {
```

针对 gas，多说一句，以太坊设计了很好的刺激机制，应用中，你在进行部署、或者执行合约接口的时候，都需要花费 gas，互联网都是免费的，以太坊却为什么要设计成这种方式？这可能互联网未来的发展趋势，后面我会写一篇文章，分析一下为什么互联网不应该是「免费」的。

接下来的代码，用两种方式测试，第一种方式通过 node 用 cli 的方式进行交互，我们初始化了一组候选人，后面就可以根据提示完成投票，也就是调用合约中的两个接口，代码略，输出的结果如下，

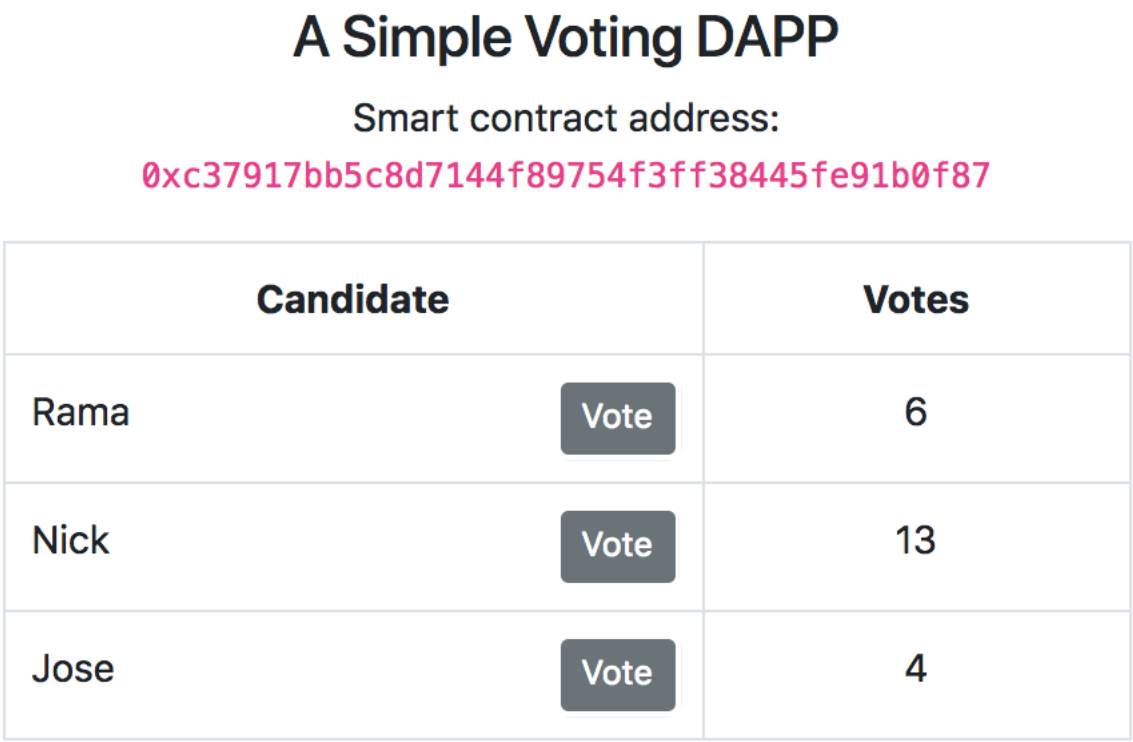
```
tams-MacBook-Pro:voting tam$ node app.js
Accounts list:

[ '0x673490daa7435a8ef9870c520b976893696fd8ee',
  '0x268b2279e2b04b3d6ce55dab1c126e67d01f1ac8',
  '0x8feb01e0fb551fabe845685cb533d40f5690fdca',
  '0x3ab7376cfc9f389efc4ee91d122415f00e3f8c13',
  '0x165c76f7e4071bff74b1212273ae47818ad934f5',
  '0x468361ee7143d0636ecf3d05e964d452b15de4a1',
  '0xf5cc7cc42f426d0d8b5ba4fa6720dc22ecd75cc2',
  '0x054e92e161867d5d4e2600ddbbac951e2b908487',
  '0xa7f08ff76e45ddee3af512439166044aab7dd07a',
  '0x59685ab7377e9e0b8ffc5321eec6a26e6edbc09a' ]


Us gas: 292703


Contract address:
0x1229aececfe592e101acc842341790fd3d22999b


Voting start("x" for quit,"o" for show vote result,otherwise input the candidate name):

Jose
Nick
Rama

Whom do you want to vote?
```

```
[Whom do you want to vote? Tom
 No this candidate.

[Whom do you want to vote? o
 Jose total vote now:  0
 Nick total vote now:  0
 Rama total vote now:  0

[Whom do you want to vote? Nick

 You voted  Nick
 Nick total vote : 1

[Whom do you want to vote? Rama

 You voted  Rama
 Rama total vote : 1

[Whom do you want to vote? o
 Jose total vote now:  0
 Nick total vote now:  1
 Rama total vote now:  1
```

第二种方式，我们把与合约的交互交给网页来操作，代码如下，

```
/* way 3 : use tempate & web3js with browser test to vote */
const http = require('http');
const ejs = require('ejs');
const tpl = fs.readFileSync('template/index.html', 'utf-8');
const options = {
    address: voteContract.address,
    candidates: defaultCandidates,
    candidatesList: JSON.stringify(defaultCandidates),
    httpProvider: httpProvider,
    abi: abi
};
var server = http.createServer(function(req, res) {
    res.end(ejs.render(tpl, options));
}).listen(3000);
```

将一些变量渲染到前端页面，然后启动 http server，把上面在 cli 中与合约交互的类似代码在前端通过 web3 来执行，结合 jQuery 和 bootstrap 完成页面工作，



我们可以看到，Dapp 的开发，前端部分，和传统的 App 开发方式没有任何差别，只是多了 web3 的 rpc 通讯机制。

如果您区块链技术感兴趣，请在公众号下回复「blockchain」，我们创建来代码仓库「区块链圣经」，对区块链技术进行知识梳理，也欢迎提交 PR 给我们！