基于 token 的投票（二）

——基于 token 的投票 DApp

之前的课程中我们已经学习了用 truffle 来构建 DApp 并部署到 Rinkeby 测试网络，接下来我们就会在原先简单投票的基础上将合约进行扩展，实现一个基于 token 的投票 DApp。

代币和支付

在以太坊中，你会遇到的一个重要概念就是 token（代币）。token 就是在以太坊上构建的数字资产。token 可以代表物理世界里的一些东西，比如黄金，或者可以是自己的数字资产（就像货币一样）。token 实际上就是智能合约，并没有什么神奇之处。

1. Gold Token（黄金代币）：银行可以有 1 千克的黄金储备，然后发行 1 千的 token。买 100 个 token 就等于买 100 克的黄金。
2. Shares in a company（公司股票）：公司股票可以用以太坊上的 token 来表示。通过支付以太，人们可以购买公司 token（股票）。
3. Gaming currency（游戏货币）：你可以有一个多玩家的游戏，游戏者可以用以太购买 token，并在游戏购买中进行花费。
4. Golem Token：这是一个以太坊项目的真实 token，你可以通过租售空闲的 CPU 来赚取 token。
5. Loyalty Points（忠诚度）：当你在一个商店购物，商店可以发行 token 作为忠诚度点数，它可以在将来作为现金回收，或是在第三方市场售卖。

在合约中如何实现 token，实际上并没有限制。但是，有一个叫做 ERC20 的 token 标准，该标准也会不断进化。ERC20 token 的优点是很容易其他的 ERC20 token 互换。同时，也更容易将你的 token 集成到其他 dapp 中。

在接下来的课程中，我们向 Voting 项目中包含 token 和支付。总的来说，我们会覆盖以下内容：

1. 使用 struct 来定义更加复杂的数据类型，在区块链上组织和存储数据
2. 实现投票的 token 化表达
3. 连接 token、投票应用和以太坊上的支付，构建完整的 DApp。

项目描述

一提到投票，你通常会想起普通的选举，你会通过投票来选出国家的首相或总统。在这种情况下，每个公民都会有一票，可以投给他们看中的候选者。

还有另外一种叫做加权投票（weighted voting）的投票方式，它常常用于公开上市交易的公司。在这些公司，股东使用它们的股票进行投票。比如，如果你拥有 10,000 股公司股票，你就有 10,000 个投票权（而不是普通选举中的一票）。我们会实现加权投票。

项目细节

比如说，我们有一个叫做 Block 的上市公司。公司有 3 个职位空闲，分别是总裁，副总裁和部长。这几个职位有 3 个竞争人选。公司想要进行选举，股东决定哪个候选人得到哪个职位。拥有最高投票的候选人将会成为总裁，然后是副总裁，最后是部长。我们会构建一个项目，并发行公司股票，允许任何人购买股票。基于所拥有的股票数，他们可以为候选人投票。比如，如果你有 10,000 股，你可以一个候选人投 5,000 股，另一个候选人 3,000 股，第三个候选人 2,000 股。

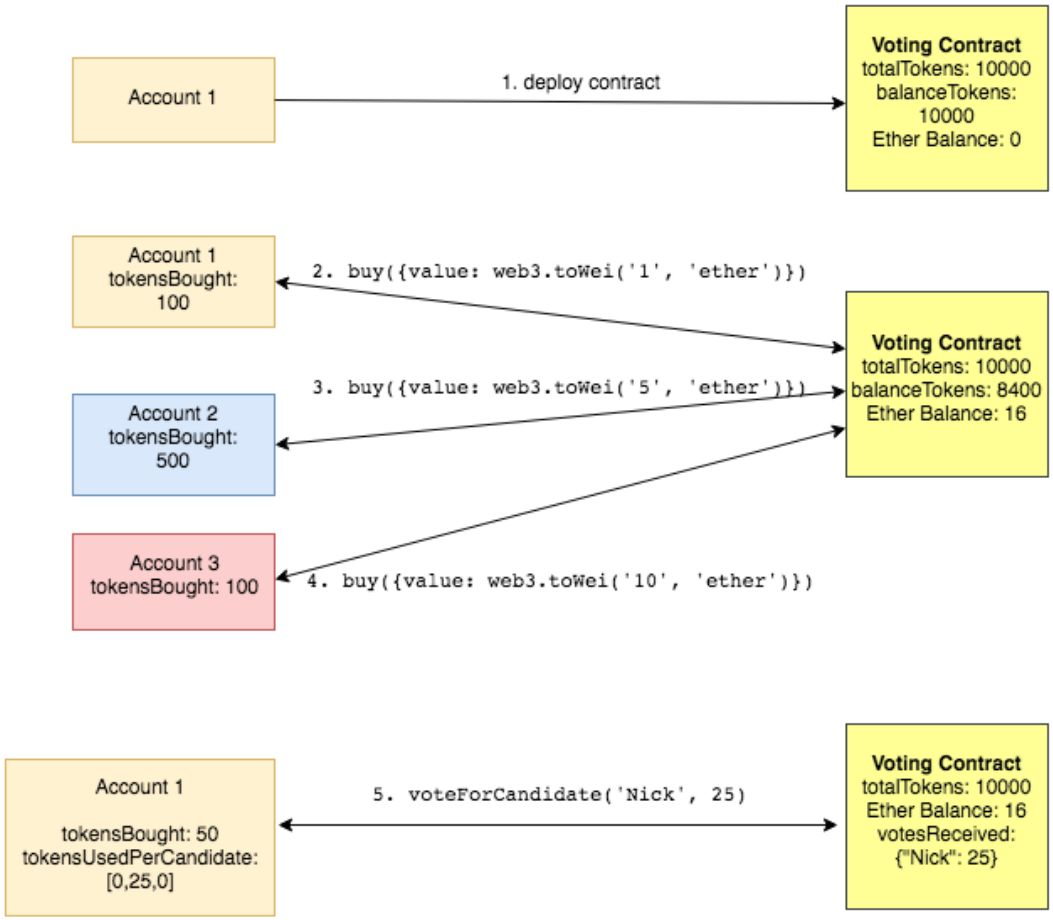
接下来，我们将会勾勒出实现框架，并随后实现构建完整应用的所有组件。

实现计划

1. 我们首先会创建一个与之前类似新的 truffle 项目。并且再次与 2_deploy_contracts.js，Voting.sol，index.html，app.js 和 app.css 打交道。

2. 我们会初始化在选举中竞争的候选者。从之前的课程中，我们已经知道了如何实现这一点。我们将会在 2_deploy_contracs.js 中完成这个任务。
3. 对于投票的股东，他们需要持有公司股票。所以，我们会先初始化公司股票。这些股票就是构成公司的数字资产。在以太坊的世界中，这些数字资产就叫做 token。从现在开始，我们将会把这些股票称为 token。除了候选者，我们还会 deployment 文件里的合约构造函数里初始化所有的 token。（提示，股票可以看做是 token，但是并非所有的以太坊 token 都是股票。股票仅仅是我们前一节中提到的 token 使用场景的一种）
4. 我们会向合约中引入一个新的方法，让任何人购买这些 token，他们会用这些 token 给候选人投票。
5. 我们也会加入一个函数来查询投票人信息，以及他们已经给谁投了票，有多少 token，他们的 token 余额。
6. 为了跟踪所有这些数据，我们会用到几个 mapping 字段，并会引入一个新的数据结构 struct 来组织投票信息。

下图是我们将要在本课程实现应用的图示。现在并不需要理解图示中的所有内容。在后面我们将会进一步阐释。



初始化 truffle 项目

在之前的学习中，你已经在系统里安装好了 webpack 和 truffle。如下所示，初始化 truffle 项目，并从 contracts 目录下移除 MetaCoin.sol。

```
>mkdir token_based_voting_dapp
>cd token_based_voting_dapp
>truffle unbox webpack
>ls
README.md       contracts       node_modules      test
webpack.config.js   truffle.jsapp          migrations
package.json
>ls app/
```

```
index.html javascripts stylesheets
>ls contracts/
ConvertLib.sol MetaCoin.sol Migrations.sol
>ls migrations/
1_initial_migration.js 2_deploy_contracts.js
>rm contracts/ConvertLib.sol contracts/MetaCoin.sol
```

投票合约

创建合约代码 Voting.sol。下面会给出详细的代码解释。

```
pragma solidity ^0.4.18;
contract Voting {
	struct voter {
		address voterAddress;
		uint tokensBought;
		uint[] tokensUsedPerCandidate;
	}
	mapping (address => voter) public voterInfo;
	mapping (bytes32 => uint) public votesReceived;
	bytes32[] public candidateList;
	uint public totalTokens;
	uint public balanceTokens;
	uint public tokenPrice;
	constructor(uint tokens, uint pricePerToken, bytes32[]
candidateNames) public {
		candidateList = candidateNames;
		totalTokens = tokens;
		balanceTokens = tokens;
		tokenPrice = pricePerToken;
	}
```

```
        function buy() payable public returns (uint) {
                uint tokensToBuy = msg.value / tokenPrice;
                require(tokensToBuy <= balanceTokens);
                voterInfo[msg.sender].voterAddress = msg.sender;
                voterInfo[msg.sender].tokensBought += tokensToBuy;
                balanceTokens -= tokensToBuy;
                return tokensToBuy;
        }
        function totalVotesFor(bytes32 candidate) view public
returns (uint) {
                return votesReceived[candidate];
        }
        function voteForCandidate(bytes32 candidate, uint
votesInTokens) public {
                uint index = indexOfCandidate(candidate);
                require(index != uint(-1));
                if ( voterInfo[msg.sender].
                        tokensUsedPerCandidate.length == 0) {
                        for(uint i = 0; i < candidateList.length
                            ;i++) {
                            voterInfo[msg.sender]
                            .tokensUsedPerCandidate
                            .push(0);
                        }
                }
                uint availableTokens =
                        voterInfo[msg.sender].tokensBought -
                            totalTokensUsed(voterInfo[msg.sender]
                            .tokensUsedPerCandidate);
                require (availableTokens >= votesInTokens);
                votesReceived[candidate] += votesInTokens;
```

```
                voterInfo[msg.sender]
                .tokensUsedPerCandidate[index] += votesInTokens;
        }
        function totalTokensUsed(uint[] _tokensUsedPerCandidate)
private pure returns (uint) {
                uint totalUsedTokens = 0;
                for(uint i = 0; i < _tokensUsedPerCandidate.length;
                        i++) {
                        totalUsedTokens +=
                            _tokensUsedPerCandidate[i];
                }
                return totalUsedTokens;
        }
        function indexOfCandidate(bytes32 candidate) view public
returns (uint) {
                for(uint i = 0; i < candidateList.length; i++) {
                        if (candidateList[i] == candidate) {
                            return i;
                        }
                }
                return uint(-1);
        }
        function tokensSold() view public returns (uint) {
                return totalTokens - balanceTokens;
        }
        function voterDetails(address user) view public returns
(uint, uint[]) {
                return (voterInfo[user].tokensBought
                        , voterInfo[user].tokensUsedPerCandidate);
        }
        function transferTo(address account) public {
                account.transfer(this.balance);
/
```

```
        }
        function allCandidates() view public returns (bytes32[]) {
                return candidateList;
        }
}
```

之前，我们仅仅有 2 个合约属性：一个数组 candidateList 存储所有的候选者，一个 mapping votesReceived 跟踪每个候选者获得的投票。

在这个合约中，我们必须再额外跟踪几个值：

- 每个投票人的信息: solidity 有个叫做 struct 的数据类型，它可以用来一组相关数据。用 struct 来存储投票人信息非常好（如果你之前没有听过 struct，把它想成一个面向对象的类即可，里面有 getter 和 setter 方法来获取这些属性）。我们会用 struct 存储投票人的地址，他们已经购买的所有 token 和给每个候选者投票所用的 token。（Line 5-9）
- 查询投票人信息的 mapping：给定一个投票人的账户地址，我们想要显示他的信息。我们会使用 voterInfo 字段来存储信息。（Line 11）
- Tokens: 我们需要有存储发行 token 总量的合约变量，还需要存储所有剩余的 token 和每个 token 的价格。（Line 17-19）

Line 21: 像上一节一样初始化构造函数。因为我们会发行任何人都可以购买的 token ，除了候选者，我们必须设置所有售卖的 token 和每个 token 的价格。

Line 28: buy 函数用于购买 token。注意关键字 “payable”。通过向一个函数添加一个关键字，任何人调用这个函数，你的合约就可以接受支付（通过以太）。

Line 28 - 35: 当你调用合约的 buy 方法时，在请求里设置你想要用于购买 token 的所有以太。以太的值通过 msg.value 。基于以太的值和 token 价格，你就可以计算出所有的 token，并将这些 token 赋予购买人。购买人的地址通过 msg.sender 可以获取。

下面是从 truffle 控制台调用 buy 的一个案例，参数传入一个 options 对象，这是 web3 v0.2x 的用法：

```
truffle(development)> Voting.deployed().then(function(contract)
{contract.buy({value: web3.toWei('1', 'ether'), from:
web3.eth.accounts[1]})})
```

它相当于 web3 v1.0 中的

```
contract.buy().send({options})
```

如果是消息调用的话就应该是

```
contract.method(parameters).call({options})
```

Line 41 - 56: voteForCandidate 方法现在有一点复杂，因为我们不仅要增加候选人的投票数，还是跟踪投票人的信息，比如投票人是谁（即他们的账户地址），给每个候选人投了多少票。

Line 83 - 85: 当一个用户调用 buy 方法发送以太来购买 token 时，所有的以太去了哪里？所有以太都在合约里。每个合约都有它自己的地址，这个地址里面存储了这些钱。可这些钱怎么拿出来呢？我们已经在这里定义了 transferTo 函数，它可以让你转移所有钱到指定的账户。该方法目前所定义的方式，任何人都可以调用，并向他们的账户转移以太，这并不是一个好的选择。你可以给谁能取钱上施加一些限制。虽然这已经超过了本课程的内容，但是我们推荐在未来实现这一点。

合约里面剩下的方法都是 getter 方法，仅仅返回合约变量的值。

注意方法上的 view 修改符，比如 tokensSold, voterDetails 等等。这些方法并不会改变区块链状态，也就是说这些是只读的方法。执行这些交易不会耗费任何 gas。

合约部署

与之前类似，更新 migrations/2_deploy_contracts.js，不过这次你需要传入两个额外的参数 “total tokens to issue”（ 示例给了 10000）和每个 token 的成本（0.01 以太）。所有的价格需要以 Wei 为单位计价，所以我们需要用 toWei 将 Ether 转换为 Wei。

```
var Voting = artifacts.require("./Voting.sol");
module.exports = function(deployer) {
       deployer.deploy(Voting, 10000,
              web3.toWei('0.01', 'ether'),
              ['Alice', 'Bob', 'Cary']);
};
```

让我们将合约部署到 ganache，测试与交互，确保代码如期工作。然后我们会把合约部署到公共的测试网。如果已经运行了 geth，停止 geth 然后启动 ganache。记得将 truffle.js 里的 ganache 改为 development，port 改为 8545；之后继续并将合约部署到网络上。

```
> truffle compile
Compiling Migrations.sol...
Compiling Voting.sol...
Writing artifacts to ./build/contracts
> truffle migrate
Running migration: 1_initial_migration.js
```

```
Deploying Migrations...
Migrations: 0x3cee101c94f8a06d549334372181bc5a7b3a8bee
Saving successful migration to network...
Saving artifacts...
Running migration: 2_deploy_contracts.js
Deploying Voting...
Voting: 0xd24a32f0ee12f5e9d233a2ebab5a53d4d4986203
Saving successful migration to network...
Saving artifacts...
```

控制台交互

```
> truffle console
```

接下来我们做一个控制台交互测试。如果成功地将合约部署到了 ganache，启动 truffle 控制台并执行以下操作，在 truffle 控制台打印（console.log）：

1. 一个候选人（比如 Alice）有多少投票？
2. 一共初始化了多少 token？
3. 已经售出了多少 token？
4. 购买 100 token
5. 购买以后账户余额是多少？
6. 已经售出了多少？
7. 给 Alice 投 25 个 token，给 Bob 和 Cary 各投 10 个 token。
8. 查询你所投账户的投票人信息（除非用了其他账户，否则你的账户默认是 web3.eth.accounts[0]）
9. 现在每个候选人有多少投票？
10. 合约里有多少 ETH？（当你通过 ETH 购买 token 时，合约接收到的 ETH）

```
truffle(development)> Voting.deployed().then(function(instance)
{instance.totalVotesFor.call('Alice').then(function(i)
{console.log(i)})})

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.totalTokens.call().then(function(v)
{console.log(v)}))})

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.tokensSold.call().then(function(v)
{console.log(v)}))})

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.buy({value: web3.toWei('1',
'ether')}).then(function(v) {console.log(v)}))})

truffle(development)> web3.eth.getBalance(web3.eth.accounts[0])

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.tokensSold.call().then(function(v)
{console.log(v)}))})

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.voteForCandidate('Alice',
25).then(function(v) {console.log(v)}))})

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.voteForCandidate('Bob',
10).then(function(v) {console.log(v)}))})

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.voteForCandidate('Cary',
10).then(function(v) {console.log(v)}))})

truffle(development)> Voting.deployed().then(function(instance)
{console.log(instance.voterDetails.call(web3.eth.accounts[0]).
then(function(v) {console.log(v)}))})
```

```
truffle(development)> Voting.deployed().then(function(instance)
{instance.totalVotesFor.call('Alice').then(function(i)
{console.log(i)})})


truffle(development)>
web3.eth.getBalance(Voting.address).toNumber()
```

Html 视图

现在，我们已经知道了合约如期工作。让我们来构建前端逻辑，以便于能够通过网页浏览器与合约交互。

将下面内容拷贝到 app/index.html。

```
<!DOCTYPE html>
<html>
  <head>
    <title>Decentralized Voting App</title>
    <link
          href='https://fonts.googleapis.com/css?family=Open+San
          s:400,700' rel='stylesheet' type='text/css'>
    <link
          href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/
          css/bootstrap.min.css' rel='stylesheet'
          type='text/css'>
     <style></style>
  </head>
  <body class="row">
    <h1 class="text-center banner">Decentralized Voting
          Application (Ropsten Testnet)</h1>
    <div class="container">
```

```
<div class="row margin-top-3">
  <div class="col-sm-12">
    <h3>How to use the app</h3>
    <strong>Step 1</strong>: Install the
     <a href="https://metamask.io/"
     target="_blank">metamask plugin</a>
     and create an account on Ropsten Test Network
     and load some Ether.
    <br>
    <strong>Step 2</strong>: Purchase tokens below by
     entering the total number of tokens you like
     to buy.
    <br>
    <strong>Step 3</strong>: Vote for candidates by entering
     their name and no. of tokens to vote with.
    <br>
    <strong>Step 4</strong>: Enter your account address to
     look up your voting activity.
  </div>
</div>
<div class="row margin-top-3">
  <div class="col-sm-7">
    <h2>Candidates</h2>
    <div class="table-responsive">
      <table class="table table-bordered">
        <thead>
          <tr>
            <th>Candidate</th>
            <th>Votes</th>
             </tr>
```

```
            </thead>
            <tbody id="candidate-rows">
            </tbody>
          </table>
        </div>
      </div>
    <div class="col-sm-offset-1 col-sm-4">
      <h2>Tokens</h2>
      <div class="table-responsive">
        <table class="table table-bordered">
          <tr>
            <th>Tokens Info</th>
            <th>Value</th>
          </tr>
          <tr>
            <td>Tokens For Sale</td>
            <td id="tokens-total"></td>
          </tr>
          <tr>
            <td>Tokens Sold</td>
            <td id="tokens-sold"></td>
          </tr>
          <tr>
            <td>Price Per Token</td>
            <td id="token-cost"></td>
          </tr>
          <tr>
            <td>Balance in the contract</td>
            <td id="contract-balance"></td>
          </tr>
```

```
            </table>
          </div>
        </div>
      </div>
      <hr>
      <div class="row margin-bottom-3">
        <div class="col-sm-7 form">
          <h2>Vote for Candidate</h2>
          <div id="msg"></div>
          <input type="text" id="candidate" class="form-control"
                 placeholder="Enter the candidate name"/>
          <br>
          <br>
          <input type="text" id="vote-tokens" class="form-control"
                 placeholder="Total no. of tokens to vote"/>
          <br>
          <br>
          <a href="#" onclick="voteForCandidate(); return false;"
                 class="btn btn-primary">Vote</a>
        </div>
        <div class="col-sm-offset-1 col-sm-4">
          <div class="col-sm-12 form">
            <h2>Purchase Tokens</h2>
            <div id="buy-msg"></div>
              <input type="text" id="buy" class="col-sm-8"
                     placeholder="Number of tokens to buy"/>
              <a href="#" onclick="buyTokens(); return false;"
                     class="btn btn-primary">Buy</a>
            </div>
            <div class="col-sm-12 margin-top-3 form">
```

```
            <h2 class="">Lookup Voter Info</h2>
            <input type="text" id="voter-info", class="col-sm-8"
                   placeholder="Enter the voter address" />
            <a href="#" onclick="lookupVoterInfo(); return
                   false;" class="btn btn-primary">Lookup</a>
            <div class="voter-details row text-left">
              <div id="tokens-bought" class="margin-top-3
                     col-md-12"></div>
              <div id="votes-cast" class="col-md-12"></div>
            </div>
          </div>
        </div>
      </div>
    </body>
    <script
        src="https://code.jquery.com/jquery-3.1.1.slim.min.js">
    </script>
    <script src="app.js"></script>
</html>
```

如果仔细看代码的话，你会发现已经没有硬编码的值了。候选者的名字会通过向部署好的合约查询进行填充。

它也会显示公司所发行的所有 token，已售出和剩余的 token。

有一节，你可以输入一个账户地址（投票人的地址），观察他们的投票行为和 token。

JavaScript

通过移除候选者姓名等等的硬编码，我们已经大幅改进了 HTML 文件。我们会使用 javascript/web3js 来填充 html 里面的所有值，并实现购买 token 的查询投票人信息的额外功能。

我们推荐用 JavaScript 自己实现，代码仅作参考之用。按照下述指引帮助实现：

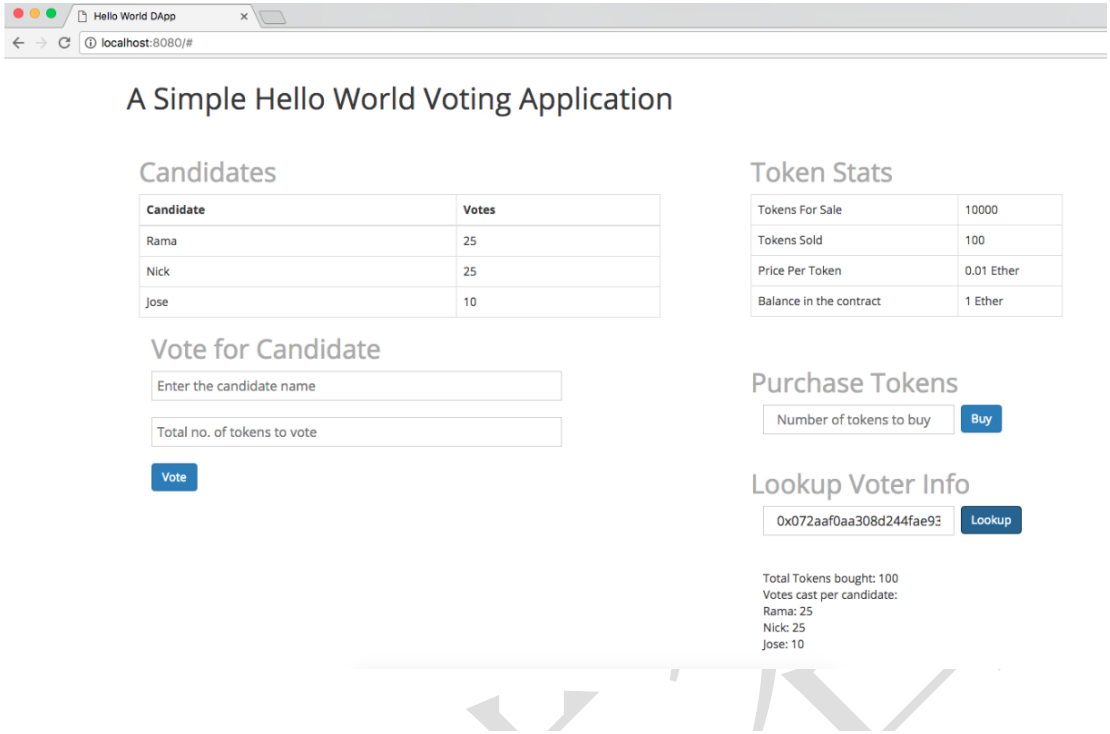
- 创建一个 Voting 合约的实例
- 在页面加载时，初始化并创建 web3 对象。（第一步和第二步与之前的课程一模一样）
- 创建一个在页面加载时调用的函数，它需要：
- 使用 Voting 合约对象，向区块链查询来获取所有的候选者姓名并填充表格。
- 再次查询区块链得到每个候选人所获得的所有投票并填充表格的列。
- 填充 token 信息，比如所有初始化的 token，剩余 token，已售出的 token 以及 token 成本。
- 实现 buyTokens 函数，它在上一节的 html 里面调用。你已经在控制台交互一节中购买了 token。buyTokens 代码与那一节一样不可或缺。
- 类似地，实现 lookupVoterInfo 函数来打印一个投票人的细节。

网页交互

CSS: app/styles/app.css.

在命令行中，使用 npm run dev 启动 web 服务器，完后你应该看到下面的内容。

如果一切顺利，你应该可以购买更多的 token，为任意候选者投票并查看投票人信息。



测试网络

现在，你可以关闭 ganache，再次启动 geth 并运行 truffle 部署到测试网。

鉴于这是一个部署在区块链上的去中心化应用，任何人都可以接入你的应用并与之交互。如果你还记得上一课，你需要将 ABI 和合约地址分享给那些想要接入你的应用的人。你可以在 truffle 的 build/contracts/Voting.json 找到 ABI 和合约地址。（这会让任何人通过命令行进行交互。如果喜欢其他人通过 GUI 使用你的应用，你仍然需要托管 web 前端。）

练习

现在合约的实现方式，用户购买 token 并用 token 投票。但是他们投票的方式是向合约发送 token。如果他们还需要在未来的选举中投票怎么办？每次投票都需要购买 token 显然是不合理的，而他们所有的 token 都会保留在合约中，并不在自己手上。进一步改善合约的方式是，加入一个方式以便于用户能够取回他们的 token。你必须实现这样一个方法，查询用户投票的所有 token，并将这些 token 返回给他们。

测试

Truffle 自带了一个自动化的测试框架，这使得测试合约非常容易。你可以通过两种方式用这个框架来写测试：

1. Solidity
2. Javascript

一般的经验是用 solidity 写单元测试，用 JavaScript 写功能测试。但是，从我们的经验来看，大部分开发者常常只会用 JavaScript 写测试。在这一章节中，你将会学习如何用这两种方式编写测试。

在这一小节中，我们会写一些 Solidity 的测试，并在下一个小节中涉及 JavaScript 测试。

当你创建好 truffle 项目后，truffle 就会在 test 目录下自动创建好 TestMetacoin.sol 和 metacoin.js。因为我们已经不再需要这些示例合约了，所以删除这些文件。

Solidity 测试

下面是 solidity 测试文件，File: TestVoting.sol

```
pragma solidity ^0.4.16;
import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/Voting.sol";
contract TestVoting {
       uint public initialBalance = 2 ether;
       function testInitialTokenBalanceUsingDeployedContract()
                 public {
          Voting voting = Voting(DeployedAddresses.Voting());
          uint expected = 10000;
          Assert.equal(voting.balanceTokens(), expected, "10000
                    Tokens not initialized for sale");
       }

       function testBuyTokens() public {
```

```
        Voting voting = Voting(DeployedAddresses.Voting());
        voting.buy.value(1 ether)();
        Assert.equal(voting.balanceTokens(), 9900, "9900
                tokens should have been available");
    }
}
```

解释如下：

1. 测试文件应该像这样命名 “Test.sol”. 这样，truffle 框架才能知道这是我们要测试合约对应的测试文件。

2. Line 2: Truffle 框架提供了一个断言的库 Assert.sol, 你可以用它来断言合约相关的任何值。它有一些函数用来断言 `equal`, `notEqual`, `isAbove`, `isBelow`, `isAtLeast`, `isAtMost`, `isZero` 和 `isNotZero`.

3. Line 3: 每当运行一个测试时，truffle 都会部署你的合约。`DeployedAddress` 是一个 truffle 框架的帮助库。通过调用 `DeployedAddress.()` 即可获取部署合约的地址。

4. Line 6: 在这个测试文件中，你会用 `TestVoting` 合约与实际的 `Voting` 合约进行交互。为了测试合约能够执行函数，它需要以太。声明一个 `initialBalance` 共有变量，并初始化一些以太。

5. Line 7 - 11: 在 `testInitialTokenBalanceUsingDeployedContract` 中，我们是测试当部署合约后，确保初始化了 10000 个代币。如果你还记得的话，代币的数量在 `migrations/2_deploy_contracts.js` 是在进行了指定。

6. Line 12 - 16: 在 `testBuyTokens` 中，智能合约购买代币，我们断言确保售出 100 个代币。记住，如果你不提供 `initialBalance`，测试合约就没有以太来购买代币，交易就会失败。

如下所示运行测试，如果你的合约代码没有任何 bug，那么测试应该会通过。我们鼓励大家多写几个测试来练习其他合约函数。

```
>truffle test test/TestVoting.sol
```

Javascript 测试

下面的 JavaScript 测试代码对你来说可能看着比较熟悉，因为我们这就是我们通过 truffle 控制台和 app.js 与合约交互的方式。Truffle 使用了 Mocha 测试框架和 Chai 用于断言。

File: voting.js

```
var Voting = artifacts.require("./Voting.sol");

contract('Voting', function(accounts) {
  it("should be able to buy tokens", function() {
        var instance;
        var tokensSold;
        var userTokens;
        return Voting.deployed().then(function(i) {
                instance = i;
                return i.buy({value: web3.toWei(1, 'ether')});
        }).then(function() {
                return instance.tokensSold.call();
        }).then(function(balance) {
                tokensSold = balance;
                return instance.voterDetails
                                .call(web3.eth.accounts[0]);
        }).then(function(tokenDetails) {
                userTokens = tokenDetails[0];
        });
  assert.equal(balance.valueOf(),100,"100 tokens were not sold");
  assert.equal(userTokens.valueOf(), 100, "100 tokens were not
        sold"); });

  it("should be able to vote for candidates", function() {
        var instance;
        return Voting.deployed().then(function(i) {
                instance = i;
                return i.buy({value: web3.toWei(1, 'ether')});
        }).then(function() {
                return instance.voteForCandidate('Alice', 3);
        }).then(function() {
                return instance.voterDetails
                                .call(web3.eth.accounts[0]);
        }).then(function(tokenDetails) {
  assert.equal(tokenDetails[1][0].valueOf(), 3, "3 tokens were
        not used for voting to Alice");
```

```
        });
  });
});
```

在 test 目录下创建一个叫做 voting.js 的文件，并将右侧代码拷贝进去。

我们有了两个测试，用于测试购买代币和为候选者投票的功能测试，并检测投票是否正确。

```
>truffle test test/voting.js
```

如果你对 JavaScript 和 promises 不太熟悉，你可能会觉得代码块中的 return 语句有点看不懂。实际上，当这些代码成功执行后，返回值会进入 then 代码块。

第 14 行的 balance 是 13 行代码的返回值。注意我们在第 12 行并没有保存任何值，因为 buy 函数没有返回任何值。20 -21 行断言确保了售出 100 个代币，并且用户拥有这 100 个代币。当出现错误时，测试就会失败，并输出一些信息（assert.equal 函数的第 3 个参数）。