简单投票 DApp

接下来我们要开始真正做一个 DApp，尽管它这是很简单的一个投票应用，但会包含完整的工作流程和交互页面。构建这个应用的主要步骤如下：

1. 我们首先安装一个叫做 `ganache` 的模拟区块链，能够让我们的程序在开发环境中运行。
2. 写一个合约并部署到 `ganache` 上。
3. 然后我们会通过命令行和网页与 `ganache` 进行交互。

我们与区块链进行通信的方式是通过 RPC（Remote Procedure Call）。web3js 是一个 JavaScript 库，它抽象出了所有的 RPC 调用，以便于你可以通过 JavaScript 与区块链进行交互。另一个好处是，web3js 能够让你使用你最喜欢的 JavaScript 框架构建非常棒的 web 应用。

开发准备-Linux

下面是基于 Linux 的安装指南。这要求我们预先安装 nodejs 和 npm，再用 npm 安装 ganache-cli、web3 和 solc，就可以继续项目的下一步了。

```
mkdir simple_voting_dapp
cd simple_voting_dapp
npm init
npm install ganache-cli web3@0.20.1 solc
node_modules/.bin/ganache-cli
```

如果安装成功，运行命令 node_modules/.bin/ganache-cli，应该能够看到下图所示的输出。

```
Ganache CLI v6.0.3 (ganache-core: 2.0.2)

Available Accounts
==================
(0) 0x5c252a0c0475f9711b56ab160a1999729eccce97
(1) 0x353d310bed379b2d1df3b727645e200997016ba3
(2) 0xa3ddc09b5e49d654a43e161cae3f865261cabd23
(3) 0xa8a188c6d97ec8cf905cc1dd1cd318e887249ec5
(4) 0xc0aa5f8b79db71335dacc7cd116f357d7ecd2798
(5) 0xda695959ff85f0581ca924e549567390a0034058
(6) 0xd4ee63452555a87048dcfe2a039208d113323790
(7) 0xc60c8a7b752d38e35e0359e25a2e0f6692b10d14
(8) 0xba7ec95286334e8634e89760fab8d2ec1226bf42
(9) 0x208e02303fe29be3698732e92ca32b88d80a2d36

Private Keys
==================
(0) a6de9563d3db157ed9926a993559dc177be74a23fd88ff5776ff0505d21fed2b
(1) 17f71d31360fbafbc90cad906723430e9694daed3c24e1e9e186b4e3ccf4d603
(2) ad2b90ce116945c11eaf081f60976d5d1d52f721e659887fcebce5c81ee6ce99
(3) 68e2288df55cbc3a13a2953508c8e0457e1e71cd8ae62f0c78c3a5c929f35430
(4) 9753b05bd606e2ffc65a190420524f2efc8b16edb8489e734a607f589f0b67a8
(5) 6e8e8c468cf75fd4de0406a1a32819036b9fa64163e8be5bb6f7914ac71251cc
(6) c287c82e2040d271b9a4e071190715d40c0b861eb248d5a671874f3ca6d978a9
(7) cec41ef9ccf6cb3007c759bf3fce8ca485239af1092065aa52b703fd04803c9d
(8) c890580206f0bbea67542246d09ab4bef7eeaa22c3448dcb7253ac2414a5362a
(9) eb8841a5ae34ff3f4248586e73fcb274a7f5dd2dc07b352d2c4b71132b3c73f0

HD Wallet
==================
Mnemonic:   cancel better shock lady capable main crunch alcohol derive alarm duck umb
Base HD Path: m/44'/60'/0'/0/{account_index}

Listening on localhost:8545
```

为了便于测试，ganache 默认会创建 10 个账户，每个账户有 100 个以太。。你需要用其中一个账户创建交易，发送、接收以太。

当然，你也可以安装 GUI 版本的 ganache 而不是命令行版本，在这里下载 GUI 版本：http://truffleframework.com/ganache/

Solidity 合约

我们会写一个叫做 Voting 的合约，这个合约有以下内容：

- 一个构造函数，用来初始化一些候选者。
- 一个用来投票的方法（对投票数加 1）

- 一个返回候选者所获得的总票数的方法

当你把合约部署到区块链的时候，就会调用构造函数，并只调用一次。与 web 世界里每次部署代码都会覆盖旧代码不同，在区块链上部署的合约是不可改变的，也就是说，如果你更新合约并再次部署，旧的合约仍然会在区块链上存在，并且数据仍在。新的部署将会创建合约的一个新的实例。

代码和解释

```
pragma solidity ^0.4.22;

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

Line 1. 我们必须指定代码将会哪个版本的编译器进行编译

Line 3. mapping 相当于一个关联数组或者是字典，是一个键值对。mapping votesReceived 的键是候选者的名字，类型为 bytes32。mapping 的值是一个未赋值的整型，存储的是投票数。

Line 4. 在很多编程语言中（例如 java、python 中的字典<HashTable 继承自字典>），仅仅通过 votesReceived.keys 就可以获取所有的候选者姓名。但是，但是在 solidity 中没有这样的方法，所以我们必须单独管理一个候选者数组 candidateList。

Line 14. 注意到 votesReceived[key] 有一个默认值 0，所以你不需要将其初始化为 0，直接加 1 即可。

你也会注意到每个函数有个可见性说明符（visibility specifier）（比如本例中的 public）。这意味着，函数可以从合约外调用。如果你不想要其他任何人调用这个函数，你可以把它设置为私有（private）函数。如果你不指定可见性，编译器会抛出一个警告。最近 solidity 编译器进行了一些改进，如果用户忘记了对私有函数进行标记导致了外部可以调用私有函数，编译器会捕获这个问题。

你也会在一些函数上看到一个修饰符 view。它通常用来告诉编译器函数是只读的（也就是说，调用该函数，区块链状态并不会更新）。

接下来，我们将会使用上一节安装的 solc 库来编译代码。如果你还记得的话，之前我们提到过 web3js 是一个库，它能够让你通过 RPC 与区块链进行交互。我们将会在 node 控制台里用这个库部署合约，并与区块链进行交互。

编译合约

```
In the node console> Web3 = require('web3')
> web3 = new Web3(new
Web3.providers.HttpProvider("http://localhost:8545"));
> web3.eth.accounts
['0x5c252a0c0475f9711b56ab160a1999729eccce97'
'0x353d310bed379b2d1df3b727645e200997016ba3']
> code = fs.readFileSync('Voting.sol').toString()
> solc = require('solc')
> compiledCode = solc.compile(code)
```

首先，在终端中运行 node 进入 node 控制台，初始化 web3 对象，并向区块链查询获取所有的账户。

确保与此同时 ganache 已经在另一个窗口中运行

为了编译合约，先从 Voting.sol 中加载代码并绑定到一个 string 类型的变量，然后像右边这样对合约进行编译。

当你成功地编译好合约，打印 compiledCode 对象（直接在 node 控制台输入 compiledCode 就可以看到内容），你会注意到有两个重要的字段，它们很重要，你必须要理解：

1. `compiledCode.contracts[':Voting'].bytecode`: 这就是 `Voting.sol` 编译好后的字节码。也是要部署到区块链上的代码。
2. `compiledCode.contracts[':Voting'].interface`: 这是一个合约的接口或者说模板（叫做 abi 定义），它告诉了用户在这个合约里有哪些方法。在未来无论何时你想要跟任意一个合约进行交互，你都会需要这个 abi 定义。你可以在这里 看到 ABI 的更多内容。

在以后的项目中，我们将会使用 truffle 框架来管理编译和与区块链的交互。但是，在使用任何框架之前，深入了解它的工作方式还是大有裨益的，因为框架会将这些内容抽象出去。

部署合约

让我们继续课程，现在将合约部署到区块链上。为此，你必须先通过传入 abi 定义来创建一个合约对象 VotingContract。然后用这个对象在链上部署并初始化合约。

```
Execute this in your node console:
> abiDefinition =
JSON.parse(compiledCode.contracts[':Voting'].interface)
> VotingContract = web3.eth.contract(abiDefinition)
> byteCode = compiledCode.contracts[':Voting'].bytecode
> deployedContract =
VotingContract.new(['Alice','Bob','Cary'],{data: byteCode, from:
web3.eth.accounts[0], gas: 4700000})
> deployedContract.address
'0x0396d2b97871144f75ba9a9c8ae12bf6c019f610'
// Your address will be different
> contractInstance = VotingContract.at(deployedContract.address)
```

VotingContract.new 将合约部署到区块链。

第一个参数是一个候选者数组，候选者们会竞争选举，这很容易理解。让我们来看一下第二个参数里面都是些什么：

1. **data**: 这是我们编译后部署到区块链上的字节码。

2. **from**: 区块链必须跟踪是谁部署了这个合约。在这种情况下，我们仅仅是从调用 web3.eth.accounts 返回的第一个账户，作为部署这个合约的账户。记住，web3.eth.accounts 返回一个 ganache 所创建 10 个测试账号的数组。在交易之前，你必须拥有这个账户，并对其解锁。创建一个账户时，你会被要求输入一个密码，这就是你用来证明你对账户所有权的东西。在下一节，我们将会进行详细介绍。为了方便起见，ganache 默认会解锁 10 个账户。
3. **gas**: 与区块链进行交互需要花费金钱。这笔钱用来付给矿工，因为他们帮你把代码包含了在区块链里面。你必须指定你愿意花费多少钱让你的代码包含在区块链中，也就是设定 “gas” 的值。你的 “from” 账户里面的 ETH 余额将会被用来购买 gas。gas 的价格由网络设定。

我们已经部署了合约，并有了一个合约实例（变量 contractInstance），我们可以用这个实例与合约进行交互。

在区块链上有上千个合约。那么，如何识别你的合约已经上链了呢？

答案是找到已部署合约的地址：deployedContract.address. 当你需要跟合约进行交互时，就需要这个部署地址和我们之前谈到的 abi 定义。

控制台交互

```
In your node console:
> contractInstance.totalVotesFor.call('Rama')
{ [String: '0'] s: 1, e: 0, c: [ 0 ] }
> contractInstance.voteForCandidate('Rama', {from:
web3.eth.accounts[0]})
```

```
'0xdedc7ae544c3dde74ab5a0b07422c5a51b5240603d31074f5b75c0ebc786bf53'
> contractInstance.voteForCandidate('Rama', {from: web3.eth.accounts[0]})
'0x02c054d238038d68b65d55770fabfca592a5cf6590229ab91bbe7cd72da46de9'
> contractInstance.voteForCandidate('Rama', {from: web3.eth.accounts[0]})
'0x3da069a09577514f2baaa11bc3015a16edf26aad28dffbcd126bde2e71f2b76f'
> contractInstance.totalVotesFor.call('Rama').toLocaleString()'3'
```

{ [String: '0'] s: 1, e: 0, c: [0] } 是数字 0 的科学计数法表示. 这里返回的值是一个 bigNumber 对象，可以用它的的.toNumber()方法来显示数字：

contractInstance.totalVotesFor.call('Alice').toNumber()

web3.fromWei(web3.eth.getBalance(web3.eth.accounts[1]).toNumber(),'ether')

BigNumber 的值以符号，指数和系数的形式,以十进制浮点格式进行存储。

s 是 sign 符号，也就是正负；

e 是 exponent 指数，表示最高位后有几个零；

c 是 coefficient 系数，也就是实际的有效数字；bignumber 构造函数的入参位数限制为14位,所以系数表示是从后向前截取的一个数组,14位截取一次。

为候选者投票并查看投票数

继续课程，在你的 node 控制台里调用 voteForCandidate 和 totalVotesFor 方法并查看结果。

每为一位候选者投一次票，你就会得到一个交易 id:

比如：

‘0xdedc7ae544c3dde74ab5a0b07422c5a51b5240603d31074f5b75c0ebc786bf53’。这个交易 id 就是交易发生的凭据，你可以在将来的任何时候引用这笔交易。这笔交易是不可改变的。

对于以太坊这样的区块链，不可改变是其主要特性之一。在接下来的章节，我们将会利用这一特性构建应用。

网页交互

至此，大部分的工作都已完成，我们还需要做的事情就是创建一个简单的 html，里面有候选者姓名并调用投票命令（我们已经在 nodejs 控制台里试过）。你可以在右侧找到 html 代码和 js 代码。将它们放到 chapter1 目录，并在浏览器中打开 index.html。

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Voting DApp</title>
  <link
href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/boot
strap.min.css' rel='stylesheet' type='text/css'>
</head>
<body class="container">
  <h1>A Simple Voting Application</h1>
  <div class="table-responsive">
```

```
    <table class="table table-bordered">
      <thead>
        <tr>
          <th>Candidate</th>
          <th>Votes</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Alice</td>
          <td id="candidate-1"></td>
        </tr>
        <tr>
          <td>Bob</td>
          <td id="candidate-2"></td>
        </tr>
        <tr>
          <td>Cary</td>
          <td id="candidate-3"></td>
        </tr>
      </tbody>
    </table>
  </div>
  <input type="text" id="candidate" />
  <a href="#" onclick="voteForCandidate()" class="btn
btn-primary">Vote</a>
</body>
<script
src="https://cdn.jsdelivr.net/gh/ethereum/web3.js/dist/web3.mi
n.js">
</script>
```

```
<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js">
</script>
<script src="./index.js"></script>
</html>
```

Tips：

1. <head>中用 link 形式引入 bootstrap 的 css 类型库，以下 container、table-responsive 等 class 均来自 bootstrap
2. <th>表头单元格，<td>表单元格，候选人名字后的单元格为得票数，用 id 区分以方便写入，之后 js 中写死了对应关系
3. <input>一个输入框，定义 id 方便在 js 中取值
4. <a>超链接形式的按键 btn，href="#"为跳转至本页，即不跳转；onclick 指向 js 中方法

为了简化项目，我们已经硬编码了候选者姓名。如果你喜欢的话，可以调整代码使其动态选择候选者。

index.js

```
web3 = new Web3(new
Web3.providers.HttpProvider("http://localhost:8545"));
abi = JSON.parse('[{"constant":false,…}]')
VotingContract = web3.eth.contract(abi);
contractInstance =
VotingContract.at('0x329f5c190380ebcf640a90d06eb1db2d68503a53'
);
candidates = {"Alice": "candidate-1", "Bob": "candidate-2",
"Cary": "candidate-3"};
```

```
function voteForCandidate(candidate) {
        candidateName = $("#candidate").val();
        try {
                contractInstance.voteForCandidate(candidateName,
                        {from: web3.eth.accounts[0]},
                        function() {
                          let div_id = candidates[candidateName];
                          $("#"+div_id).html(
                              contractInstance.totalVotesFor
                               .call(candidateName)
                               .toString());
                          }
                );
        } catch (err) {
        }
}
$(document).ready(function() {
        candidateNames = Object.keys(candidates);
        for (var i = 0; i < candidateNames.length; i++) {
                let name = candidateNames[i];
                let val = contractInstance.totalVotesFor
                             .call(name).toString()
                $("#"+candidates[name]).html(val);
        }
});
```

在第 4 行，用你自己的合约地址替换代码中的合约地址。合约地址是之前的 deployedContract.address

如果一切顺利的话，你应该能够在文本框中输入候选者姓名，然后投票数应该加 1 。

注意：由于网络原因，web3.js 可能无法获取，可自行下载到本地导入。

如果你可以看到页面，为候选者投票，然后看到投票数增加，那就已经成功创建了第一个合约，恭喜！所有投票都会保存到区块链上，并且是不可改变的。任何人都可以独立验证每个候选者获得了多少投票。当然，我们所有的事情都是在一个模拟的区块链上（ganache）完成，在接下来的课程中，我们会将这个合约部署到真正的公链上。在 Part 2，我们会把合约部署到叫做 Ropsten testnet 的公链，同时也会学习如何使用 truffle 框架构建合约，管理 dapp。

总结一下，下面是你到目前为止已经完成的事情：

1. 通过安装 node, npm 和 ganache，你已经配置好了开发环境。
2. 你编码了一个简单的投票合约，编译并部署到区块链上。
3. 你通过 nodejs 控制台与网页与合约进行了交互。