如何编写智能合约（Smart Contract）？（II）建立加密代币

接着上一篇如何编写智能合约（Smart Contract）？，本篇文章，我们将写一个简单的加密代币的智能合约来给大家诠释加密代币的原理，当然这篇文章只是告诉你加密代币的原理，存在很多漏洞，不能直接使用。

启动testrpc

打开终端，启动 `testrpc`，相关环境在如何编写智能合约（Smart Contract）？这篇文章里面已经有具体说明。

```
liyuechun:~ yuechunli$ testrpc
EthereumJS TestRPC v4.1.3 (ganache-core: 1.1.3)
...
```

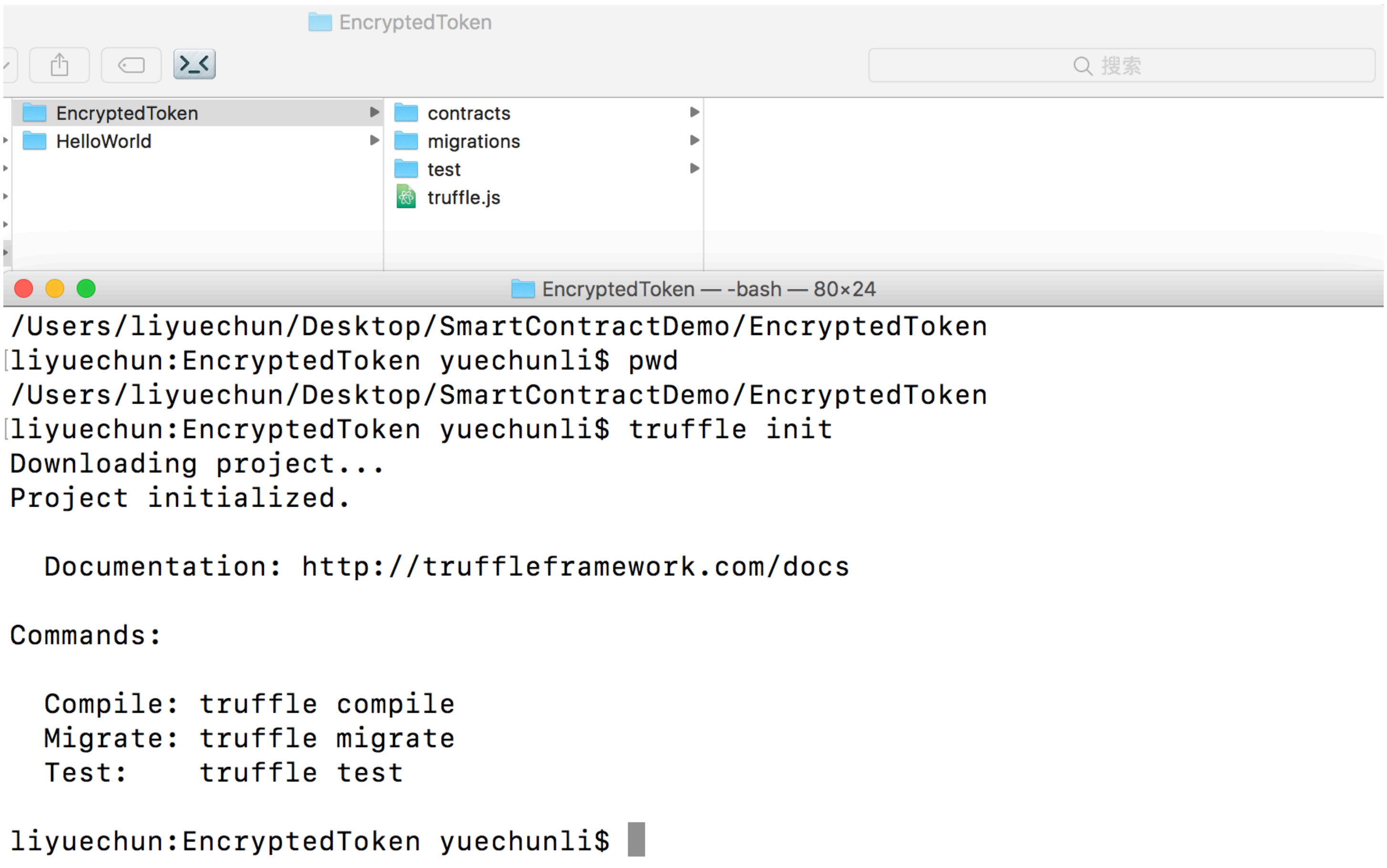
接下来我们就可以一步步的创建我们的加密代币项目了。

代币合约的基本概念

代币合约扮演的角色相当于银行的角色。使用者在代币合约中，用自己的 `以太币帐户地址` 当作银行帐户，可以透过代币合约执行转账（ `transfer`，将代币由一个帐户转到另一个帐户），查询余额（ `balanceOf`，查询指定帐户中拥有的代币）等原本由银行负责的工作。因为合约部署在公开区块链上，所有的交易都是公开透明，可供检验的。

创建代币合约项目

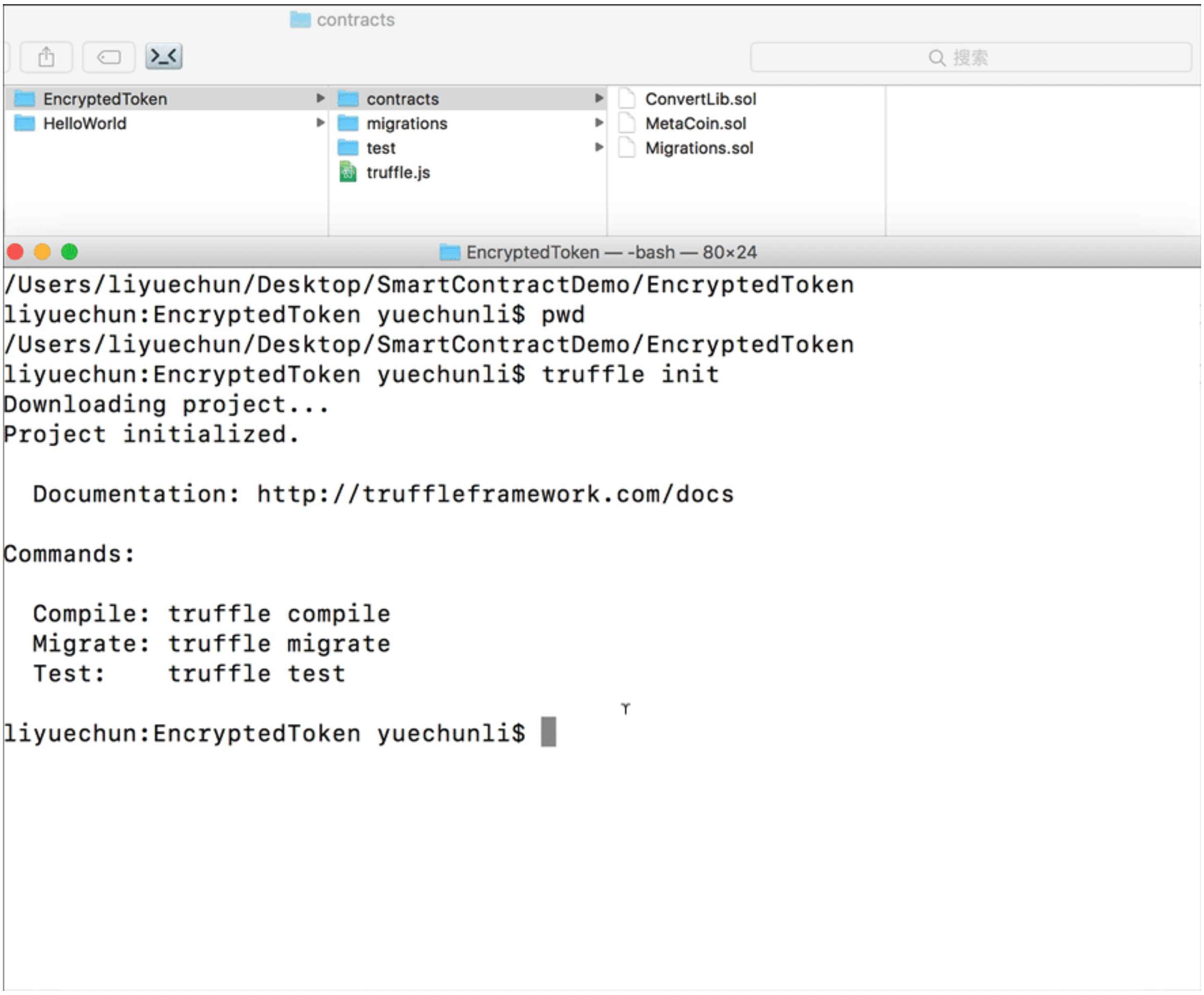
```
liyuechun:EncryptedToken yuechunli$ pwd
/Users/liyuechun/Desktop/SmartContractDemo/EncryptedToken
liyuechun:EncryptedToken yuechunli$ truffle init
Downloading project...
Project initialized.

  Documentation: http://truffleframework.com/docs

Commands:
  Compile: truffle compile
  Migrate: truffle migrate
  Test:    truffle test

liyuechun:EncryptedToken yuechunli$
```



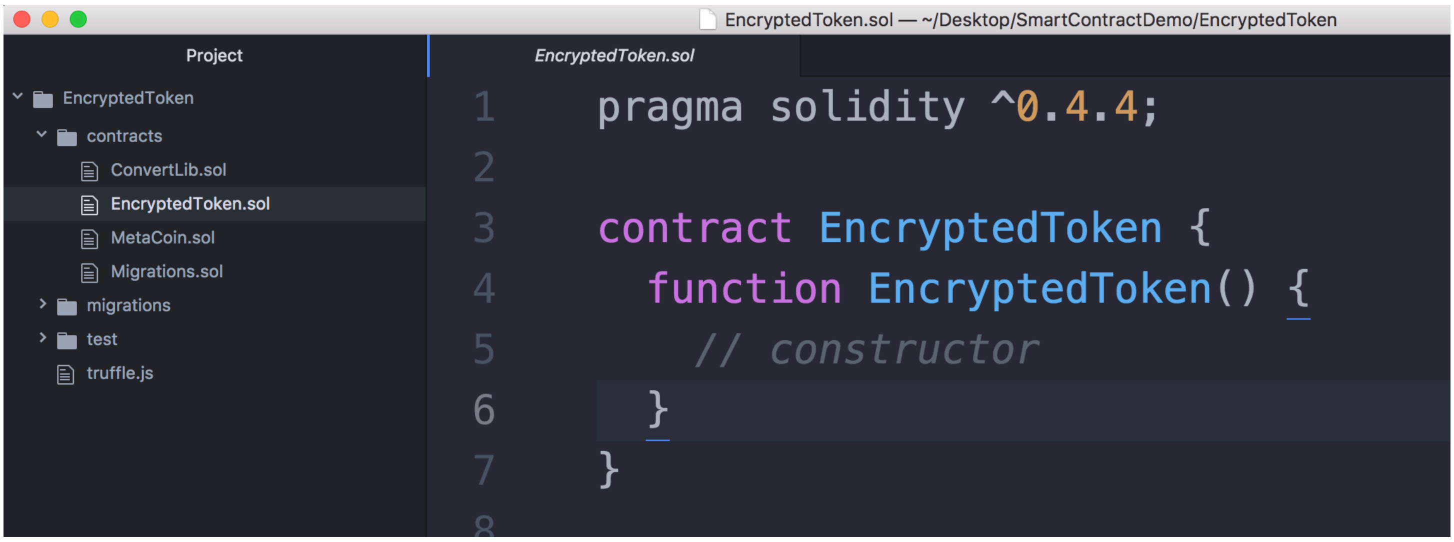
```
/Users/liyuechun/Desktop/SmartContractDemo/EncryptedToken
liyuechun:EncryptedToken yuechunli$ pwd
/Users/liyuechun/Desktop/SmartContractDemo/EncryptedToken
liyuechun:EncryptedToken yuechunli$ truffle init
Downloading project...
Project initialized.

  Documentation: http://truffleframework.com/docs

Commands:

  Compile: truffle compile
  Migrate: truffle migrate
  Test:    truffle test

liyuechun:EncryptedToken yuechunli$
```

新建代币合约

终端执行`truffle create contract EncryptedToken`命令创建`EncryptedToken.sol`合约。



```
/Users/liyuechun/Desktop/SmartContractDemo/EncryptedToken
liyuechun:EncryptedToken yuechunli$ pwd
/Users/liyuechun/Desktop/SmartContractDemo/EncryptedToken
liyuechun:EncryptedToken yuechunli$ truffle init
Downloading project...
Project initialized.

  Documentation: http://truffleframework.com/docs

Commands:

  Compile: truffle compile
  Migrate: truffle migrate
  Test:    truffle test

liyuechun:EncryptedToken yuechunli$ 
```



```
pragma solidity ^0.4.4;

contract EncryptedToken {
  function EncryptedToken() {
    // constructor
  }
}
```

编写合约代码

将下面的合约代码拷贝，替换`EncryptedToken.sol`文件的代码。

```
pragma solidity ^0.4.4;

contract EncryptedToken {
  uint256 INITIAL_SUPPLY = 666666;
  mapping(address => uint256) balances;
  function EncryptedToken() {
    balances[msg.sender] = INITIAL_SUPPLY;
  }
  // 转账到一个指定的地址
  function transfer(address _to, uint256 _amount) {
    assert(balances[msg.sender] < _amount);
    balances[msg.sender] -= _amount;
    balances[_to] += _amount;
  }
  // 查看指定地址的余额
  function balanceOf(address _owner) constant returns (uint256) {
    return balances[_owner];
  }
}
```

`pragma solidity ^0.4.4` 中的 `0.4.4` 代表 `solidity` 的版本， `^` 代表 `0.4.4 ~ 0.4.9` 之间的 `solidity` 都可以正常编译当前版本的合约。

`contract` 相当于其他语言中的 `class` ， `EncryptedToken` 相当于 `类` 的名字。 `contract EncryptedToken` 可以理解为 `class EncryptedToken extends Contract` 。

`uint256 INITIAL_SUPPLY = 666666` 声明了一个变量 `INITIAL_SUPPLY` ，初始化存储了一个 `666666` 的整数作为部署当前合约的钱包地址的代币数。

`mapping(address => uint256) balances;` ， `balances` 是一个 `key` 类型为 `address` ， `value` 类型为 `uint256` 的键值对(mapping)，相当于Java中的 `map` 、iOS中的 `NSDictionary` 。

`function EncryptedToken()` 函数是 `EncryptedToken` 合约的构造函数(contructor)，当 `EncryptedToken` 合约调用时，会先执行它的构造函数。

在构造函数中，会以当前部署合约的钱包地址为 `key` ，以 `INITIAL_SUPPLY` 为 `value` 初始化一个键值对。

```
function transfer(address _to, uint256 _amount) {
    assert(balances[msg.sender] < _amount);
    balances[msg.sender] -= _amount;
    balances[_to] += _amount;
}
```

`transfer` 函数是声明用来从转账到指定钱包地址的函数， `_to` 代表转账的目的地地址， `_amount` 代表转账金额。

`assert(balances[msg.sender] < _amount)` ，这句代码中，我声明了一个断言，当 `balances[msg.sender] < _amount` ，即当前钱包余额小于要转账的额度时，就会抛出异常。

`balances[msg.sender] -= _amount;` 从当前钱包额度中减去 `_amount` 。

`balances[_to] += _amount;` ，将目标地址的额度增加 `_amount` 。

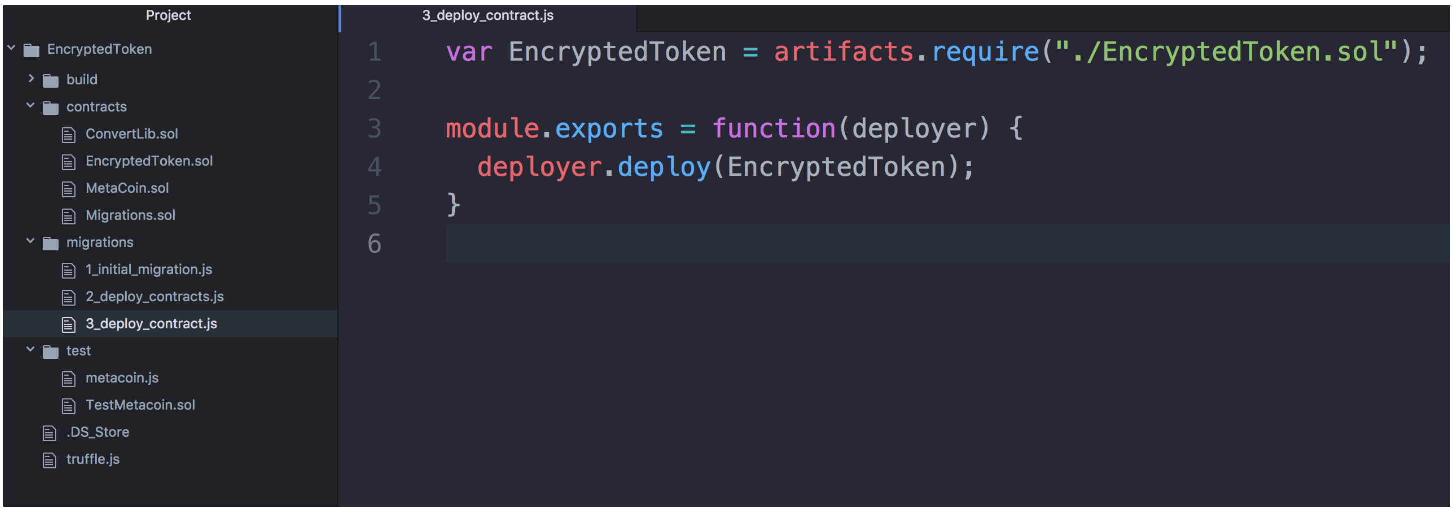
```
  function balanceOf(address _owner) constant returns (uint256) {

     return balances[_owner];
  }
```

`balanceOf(address _owner)` 函数是用来查询指定钱包地址的余额， `_owner` 即是指定的钱包地址， `returns (uint256)` 代表返回值的类型为 `uint256` ， `constant` 关键字的作用是，当我们调用 `balanceOf` 函数时，它会自动调用 `call()` 方法，表明只是读书数据，而不需要往区块链写入数据，调用这个方法，不需要花费手续费。

编译与部署

在 `migrations/` 目录下创建一个名字叫做 `3_deploy_contract.js` 的文件。文件中的内容为：

```
var EncryptedToken = artifacts.require('./EncryptedToken.sol');

module.exports = function(deployer) {
  deployer.deploy(EncryptedToken);
}
```



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

module.exports = function(deployer) {
  deployer.deploy(EncryptedToken);
}

```

接下来执行`compile`和`migrate`命令：

```
liyuechun:EncryptedToken yuechunli$ truffle compile
Compiling ./contracts/ConvertLib.sol...
Compiling ./contracts/EncryptedToken.sol...
Compiling ./contracts/MetaCoin.sol...
Compiling ./contracts/Migrations.sol...
Writing artifacts to ./build/contracts

liyuechun:EncryptedToken yuechunli$ truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x8b31575b8ac09efae7bdf933823fcf91885b4e1c93b4d9b37421d22ab73cef25
  Migrations: 0x9b34fffa8e0d2e6d09f59ee289db13e0662b68fe
Saving successful migration to network...
  ... 0x45ba02af03a7e94f66bebbb72eebf547ffe128a49f01437aca6b45c2f489b0a1
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying ConvertLib...
  ... 0xae9c5fd334d8925b027d8de3ca9aeb13e2bd1d2d3b95b49ef3c7b472446d73ae
  ConvertLib: 0x3f7fff3afe38a0a152806a0f4f416e52a338829f
  Linking ConvertLib to MetaCoin
  Deploying MetaCoin...
  ... 0x8da8a4c881eb9bb62a1477db13044ce84707aac0df54f7ab1baf8b7f0904d251
  MetaCoin: 0x7b9cc3ebba27a1ef8dda0b1825ae47369d647739
Saving successful migration to network...
  ... 0x319cebe2651f0dfbb4adb0a187745a6d47bde22243e4cb32346790ea9abbc5c8
Saving artifacts...
Running migration: 3_deploy_contract.js
  Deploying EncryptedToken...
  ... 0x5e6203aaf1280bac1f991cb3d2858342d9287c0970da8068eeae657b54022cab
  EncryptedToken: 0x16685fb5d7e88293fc1c79a0428f3ab9d2165384
Saving successful migration to network...
```

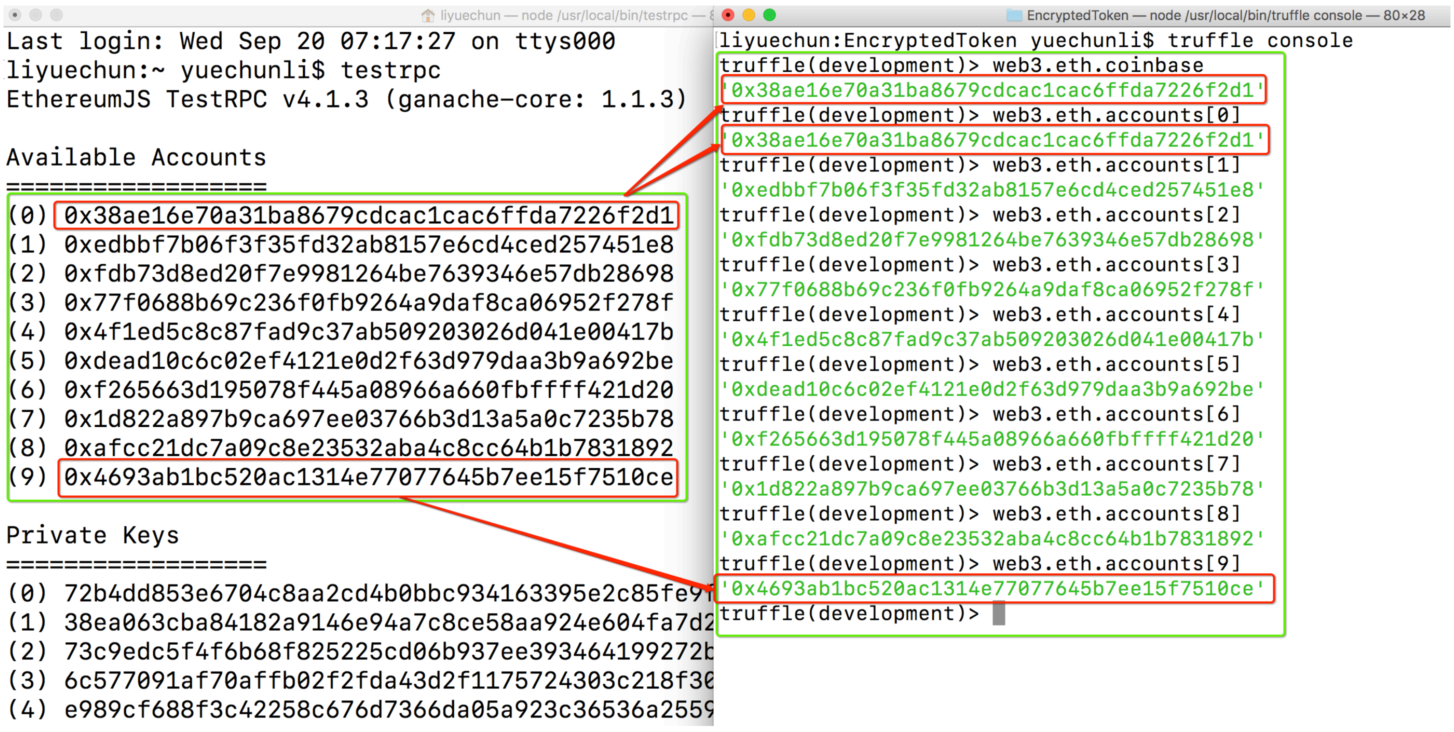
```
  ... 0x713227a22cafea3ff1fa3bd35ff38cb367d9c8dbdc698124053f72020d37bf28
Saving artifacts...
liyuechun:EncryptedToken yuechunli$
```

如上所示，我们已经将`EncryptedToken`代币合约部署到了`testrpc`上。

合约验证

合约部署完成后，我们通过`truffle console`开启`console`控制台，在这个控制台中对已经部署的合约进行验证。

```
liyuechun:EncryptedToken yuechunli$ truffle console
truffle(development)> web3.eth.coinbase
'0x38ae16e70a31ba8679cdcac1cac6ffda7226f2d1'
truffle(development)> web3.eth.accounts[0]
'0x38ae16e70a31ba8679cdcac1cac6ffda7226f2d1'
truffle(development)> web3.eth.accounts[1]
'0xedbbf7b06f3f35fd32ab8157e6cd4ced257451e8'
truffle(development)> web3.eth.accounts[2]
'0xfdb73d8ed20f7e9981264be7639346e57db28698'
truffle(development)> web3.eth.accounts[3]
'0x77f0688b69c236f0fb9264a9daf8ca06952f278f'
truffle(development)> web3.eth.accounts[4]
'0x4f1ed5c8c87fad9c37ab509203026d041e00417b'
truffle(development)> web3.eth.accounts[5]
'0xdead10c6c02ef4121e0d2f63d979daa3b9a692be'
truffle(development)> web3.eth.accounts[6]
'0xf265663d195078f445a08966a660fbffff421d20'
truffle(development)> web3.eth.accounts[7]
'0x1d822a897b9ca697ee03766b3d13a5a0c7235b78'
truffle(development)> web3.eth.accounts[8]
'0xafcc21dc7a09c8e23532aba4c8cc64b1b7831892'
truffle(development)> web3.eth.accounts[9]
'0x4693ab1bc520ac1314e77077645b7ee15f7510ce'
truffle(development)>
```


liyuechun — node /usr/local/bin/testrpc
EncryptedToken — node /usr/local/bin/truffle console — 80×28

```
Last login: Wed Sep 20 07:17:27 on ttys000
liyuechun:~ yuechunli$ testrpc
EthereumJS TestRPC v4.1.3 (ganache-core: 1.1.3)

Available Accounts
==================
(0) 0x38ae16e70a31ba8679cdcac1cac6ffda7226f2d1
(1) 0xedbbf7b06f3f35fd32ab8157e6cd4ced257451e8
(2) 0xfdb73d8ed20f7e9981264be7639346e57db28698
(3) 0x77f0688b69c236f0fb9264a9daf8ca06952f278f
(4) 0x4f1ed5c8c87fad9c37ab509203026d041e00417b
(5) 0xdead10c6c02ef4121e0d2f63d979daa3b9a692be
(6) 0xf265663d195078f445a08966a660fbffff421d20
(7) 0x1d822a897b9ca697ee03766b3d13a5a0c7235b78
(8) 0xafcc21dc7a09c8e23532aba4c8cc64b1b7831892
(9) 0x4693ab1bc520ac1314e77077645b7ee15f7510ce

Private Keys
==================
(0) 72b4dd853e6704c8aa2cd4b0bbc934163395e2c85fe9
(1) 38ea063cba84182a9146e94a7c8ce58aa924e604fa7d2
(2) 73c9edc5f4f6b68f825225cd06b937ee393464199272b
(3) 6c577091af70affb02f2fda43d2f1175724303c218f30
(4) e989cf688f3c42258c676d7366da05a923c36536a2559
```

```
liyuechun:EncryptedToken yuechunli$ truffle console
truffle(development)> web3.eth.coinbase
'0x38ae16e70a31ba8679cdcac1cac6ffda7226f2d1'
truffle(development)> web3.eth.accounts[0]
'0x38ae16e70a31ba8679cdcac1cac6ffda7226f2d1'
truffle(development)> web3.eth.accounts[1]
'0xedbbf7b06f3f35fd32ab8157e6cd4ced257451e8'
truffle(development)> web3.eth.accounts[2]
'0xfdb73d8ed20f7e9981264be7639346e57db28698'
truffle(development)> web3.eth.accounts[3]
'0x77f0688b69c236f0fb9264a9daf8ca06952f278f'
truffle(development)> web3.eth.accounts[4]
'0x4f1ed5c8c87fad9c37ab509203026d041e00417b'
truffle(development)> web3.eth.accounts[5]
'0xdead10c6c02ef4121e0d2f63d979daa3b9a692be'
truffle(development)> web3.eth.accounts[6]
'0xf265663d195078f445a08966a660fbffff421d20'
truffle(development)> web3.eth.accounts[7]
'0x1d822a897b9ca697ee03766b3d13a5a0c7235b78'
truffle(development)> web3.eth.accounts[8]
'0xafcc21dc7a09c8e23532aba4c8cc64b1b7831892'
truffle(development)> web3.eth.accounts[9]
'0x4693ab1bc520ac1314e77077645b7ee15f7510ce'
truffle(development)>
```

在`testrpc`启动时，系统会给我们分配10个钱包地址，如上图所示我们可以通过`web3.eth.coinbase`或者`web3.eth.accounts[0]`获取首个钱包地址，当然也可以根据索引获取其他的钱包地址。

接下来声明一个合约变量存储`EncryptedToken`合约实例。

```
truffle(development)> let contract;
undefined
truffle(development)> EncryptedToken.deployed().then(instance => contract =
instance)
.....
truffle(development)>
```

验证`web3.eth.coinbase`和`web3.eth.accounts[1]`中的余额。

```
truffle(development)> contract.balanceOf(web3.eth.coinbase)
{ [String: '666666'] s: 1, e: 5, c: [ 666666 ] }
truffle(development)> contract.balanceOf(web3.eth.accounts[1])
{ [String: '0'] s: 1, e: 0, c: [ 0 ] }
truffle(development)>
```

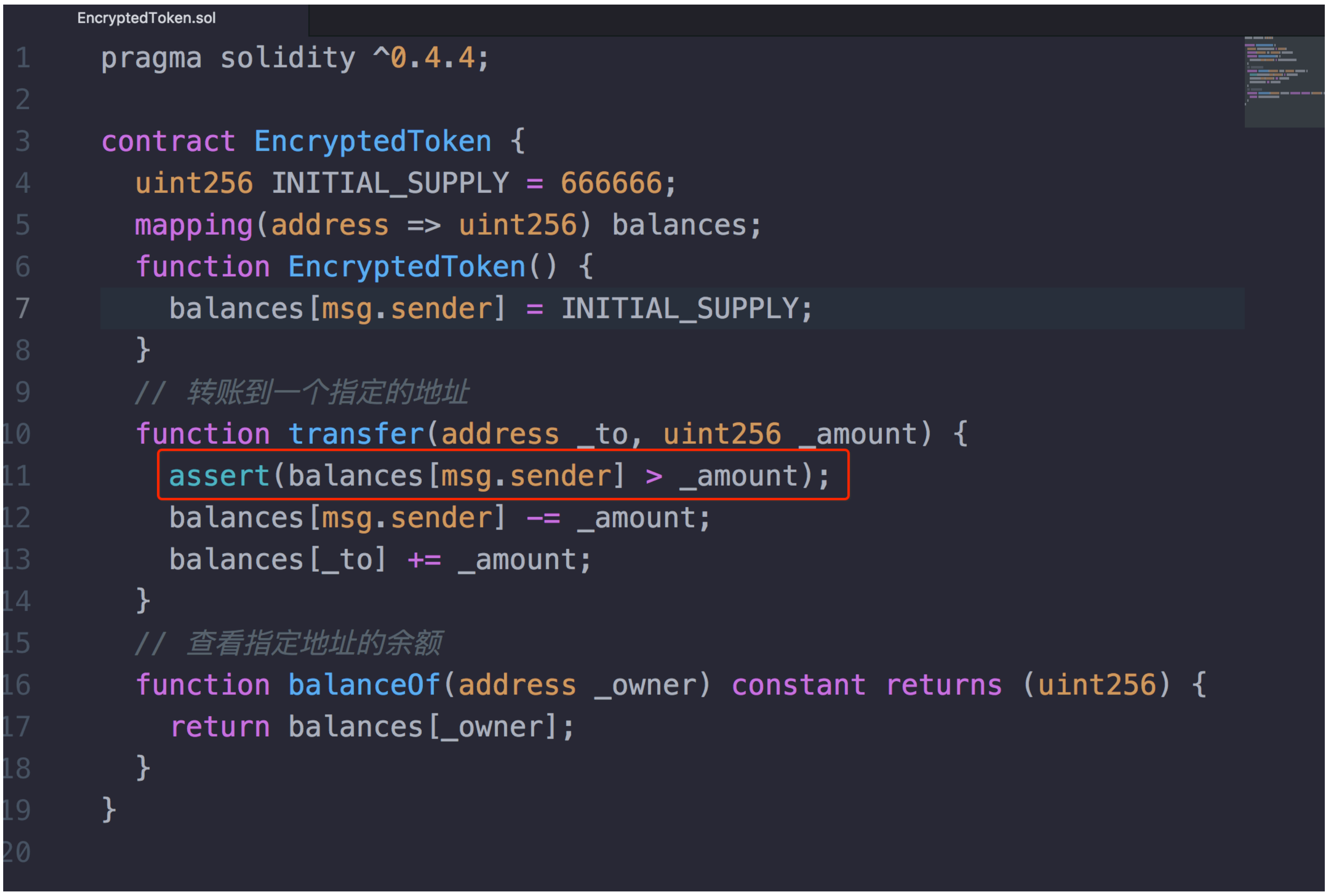
经验证，第0个钱包地址中的代币余额为`666666`，第1个钱包地址中的代币余额为`0`。

下一步，我们将从第0个账号中向第1个账号转账`666`个代币。

```
truffle(development)> contract.transfer(web3.eth.accounts[1], 666)
```

```
Error: VM Exception while processing transaction: invalid opcode
    at Object.InvalidResponse (/usr/local/lib/node_modules/truffle/build/cli
.bundled.js:37295:16)
    at /usr/local/lib/node_modules/truffle/build/cli.bundled.js:224765:36
    at /usr/local/lib/node_modules/truffle/build/cli.bundled.js:208348:9
    at XMLHttpRequest.request.onreadystatechange (/usr/local/lib/node_module
s/truffle/build/cli.bundled.js:209773:13)
    at XMLHttpRequestEventTarget.dispatchEvent (/usr/local/lib/node_modules/
truffle/build/cli.bundled.js:67130:18)
    at XMLHttpRequest._setReadyState (/usr/local/lib/node_modules/truffle/bu
ild/cli.bundled.js:67420:12)
    at XMLHttpRequest._onHttpResponseEnd (/usr/local/lib/node_modules/truffl
e/build/cli.bundled.js:67575:12)
    at IncomingMessage.<anonymous> (/usr/local/lib/node_modules/truffle/buil
d/cli.bundled.js:67535:24)
    at emitNone (events.js:110:20)
    at IncomingMessage.emit (events.js:207:7)
truffle(development)>
```

如上所示，转账过程中出现了异常，转账失败，仔细检查一下，不难发现是因为在我们合约代码 `EncryptedToken.sol` 中有这么一句代码 `assert(balances[msg.sender] < _amount);`，也就是说只有当 `balances[msg.sender]` 小于 `_amount` 时，才不会出现异常，所以我们应该将 `<` 符号换成 `>` 符号，即当 `balances[msg.sender]` 余额不足时抛出异常。



```
pragma solidity ^0.4.4;

contract EncryptedToken {
  uint256 INITIAL_SUPPLY = 666666;
  mapping(address => uint256) balances;
  function EncryptedToken() {
    balances[msg.sender] = INITIAL_SUPPLY;
  }
  // 转账到一个指定的地址
  function transfer(address _to, uint256 _amount) {
    assert(balances[msg.sender] > _amount);
    balances[msg.sender] -= _amount;
    balances[_to] += _amount;
  }
  // 查看指定地址的余额
  function balanceOf(address _owner) constant returns (uint256) {
    return balances[_owner];
  }
}
```

代码修改后，需要重新编译，部署。

```
liyuechun:EncryptedToken yuechunli$ pwd
/Users/liyuechun/Desktop/SmartContractDemo/EncryptedToken
liyuechun:EncryptedToken yuechunli$ ls
build       contracts   migrations  test        truffle.js
liyuechun:EncryptedToken yuechunli$ rm -rf build/
liyuechun:EncryptedToken yuechunli$ ls
contracts   migrations  test        truffle.js
liyuechun:EncryptedToken yuechunli$ truffle compile
Compiling ./contracts/ConvertLib.sol...
Compiling ./contracts/EncryptedToken.sol...
Compiling ./contracts/MetaCoin.sol...
Compiling ./contracts/Migrations.sol...
Writing artifacts to ./build/contracts

liyuechun:EncryptedToken yuechunli$ truffle migrate --reset
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0x04c008d8e7008afa1ace7c0dc294eb2e5b02a94960e559811d12187c53b07e56
  Migrations: 0x192929621fd7cf4dd2ac72b42c5f52b40ccfa999
Saving successful migration to network...
  ... 0xd4c65a206e68caab63957343e59282f7a34d92b0f1e1ac99cf77c127d014f958
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying ConvertLib...
  ... 0x4d9e476d05e1b7aea3764c0fd44799ce984d04006435d2959a87b97f62ee2c5a
  ConvertLib: 0xdf7dc01c43772881f21f857f2d689ea591a66a20
  Linking ConvertLib to MetaCoin
  Deploying MetaCoin...
  ... 0xe4ef8fc746d9772a964d8137af4c14eab4c03a7071e1b1b1f27ec963a28558b0
  MetaCoin: 0x685a10d1a8d5196e20a8ae31e0fe80f122e4dd47
Saving successful migration to network...
  ... 0xdd3a300b3f22997540ae019f31f0de4b6b8117cf1dcbc61ed5c2b25bb8011e7c
Saving artifacts...
Running migration: 3_deploy_contract.js
  Deploying EncryptedToken...
  ... 0x08ff437889f6bca7650550491937ed078d09bbbec5a2986146824ba7b7bd0e27
  EncryptedToken: 0xe4de630b2c2818dc18a1ac16935a945634c33a03
Saving successful migration to network...
  ... 0x6eadbc688aca46b34d11696ceca3564ded694dec772412ecf3da4cb6310f7d23
Saving artifacts...
liyuechun:EncryptedToken yuechunli$
```

备注：编译时，一定要先将`build`文件夹删除，其次在部署合约时，一定要添加`--reset`，否

则修改后的合约没法部署成功。

打开控制台，按照如下操作进行验证。

```
liyuechun:EncryptedToken yuechunli$ truffle console
truffle(development)> let contract
undefined
truffle(development)> EncryptedToken.deployed().then(instance => contract = instance)
truffle(development)> contract.balanceOf(web3.eth.coinbase)
{ [String: '666666'] s: 1, e: 5, c: [ 666666 ] }
truffle(development)> contract.balanceOf(web3.eth.accounts[1])
{ [String: '0'] s: 1, e: 0, c: [ 0 ] }
truffle(development)> contract.transfer(web3.eth.accounts[1], 666)
{ tx: '0x93a38d9c86520cdd5c033511bb67f4aa5230811acf8eb6a703d907d31a4d044d',
  receipt:
   { transactionHash: '0x93a38d9c86520cdd5c033511bb67f4aa5230811acf8eb6a703d907d31a4d044d',
     transactionIndex: 0,
     blockHash: '0xe02e7c9859d43e398bf0ece48e3a15510bb76beefa9bd48ce46a39022ccf5a43',
     blockNumber: 31,
     gasUsed: 48952,
     cumulativeGasUsed: 48952,
     contractAddress: null,
     logs: [] },
  logs: [] }
truffle(development)> contract.balanceOf(web3.eth.coinbase)
{ [String: '666000'] s: 1, e: 5, c: [ 666000 ] }
truffle(development)> contract.balanceOf(web3.eth.accounts[1])
{ [String: '666'] s: 1, e: 2, c: [ 666 ] }
truffle(development)>
```

如上所示，代币转账成功。

总结

在这篇文章中，只是简单介绍了代币系统的逻辑，并没有对安全进行相关操作，比如：余额不够的处理、地址合不合法的处理等等。当然，通过这篇文章的学习，你再回头看我们初始项目中的 `MetaCoin` 代码时，你就应该能看懂了。

打赏地址

比特币：1FcbBw62FHBJKTiLGNoguSwkBdVnJQ9NUn
以太坊：0xF055775eBD516e7419ae486C1d50C682d4170645

技术交流

- 区块链技术交流QQ群：348924182
- 「区块链部落」官方公众号

长按，识别二维码，加关注

参考资料

- [1] http://solidity.readthedocs.io/en/latest/index.html
- [2] https://ethereum.github.io/browser-solidity/
- [3] http://truffleframework.com/
- [4] https://github.com/iurimatias/embark-framework
- [5] https://github.com/ethereum/ens
- [6] https://github.com/ethereumjs/testrpc
- [7] https://github.com/ethereumjs/ethereumjs-vm
- [8] http://web3js.readthedocs.io/en/1.0/index.html