**构建标准加密货币贝壳币（BKC）**

创建项目

有别于之前使用truffle init指令来初始化项目，在Truffle推出Boxes功能之后，我们可以直接套用称作react-box的样板，此样板已经整合create-react-app，可以直接用它来开发react web，省下项目设置的时间。

liyuechun:BloggerCoin yuechunli$ pwd

/Users/liyuechun/Desktop/SmartContractDemo/BloggerCoin

liyuechun:BloggerCoin yuechunli$ truffle unbox react-box

Downloading...

Unpacking...

Setting up...

Unbox successful. Sweet!

Commands:

Compile: truffle compile

Migrate: truffle migrate

Test contracts: truffle test

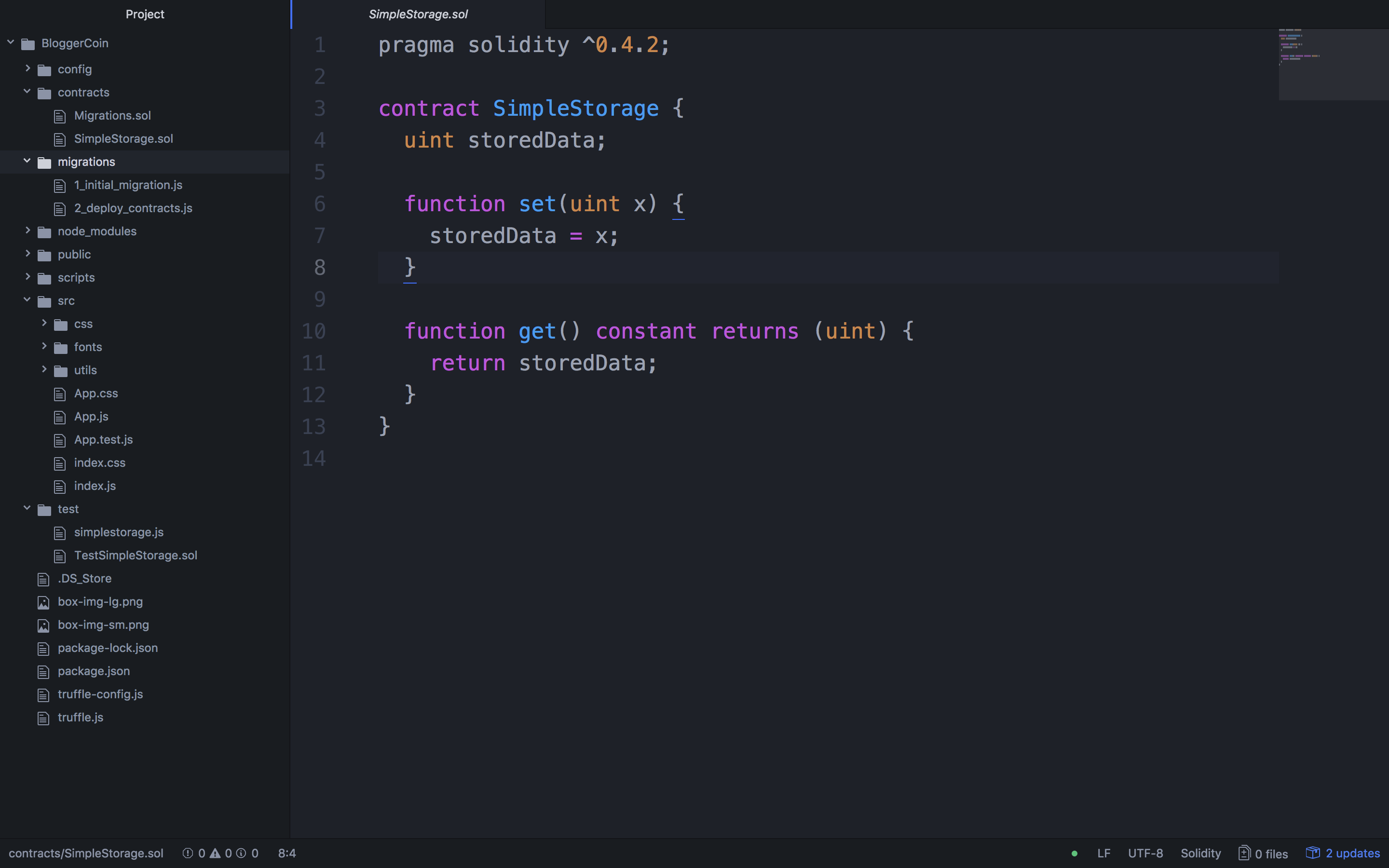
Test dapp: npm test

Run dev server: npm run start

Build for production: npm run build

liyuechun:BloggerCoin yuechunli$

目录结构：



* /contracts:存放智能合约原始码的地方，可以看到里面已经有放两个sol文件。我们开发的BloggerCoin.sol也会放在这里。
* /migrations:这是Truffle用来部署智能合约的功能，待会我们会修改2\_deploy\_contracts.js来部署BloggerCoin.sol。
* /test:测试智能合约的代码放这目录，支持js与sol测试。
* /public、/src:存放react web的地方，后面用到会再说明。

truffle.js: Truffle的设置文件。

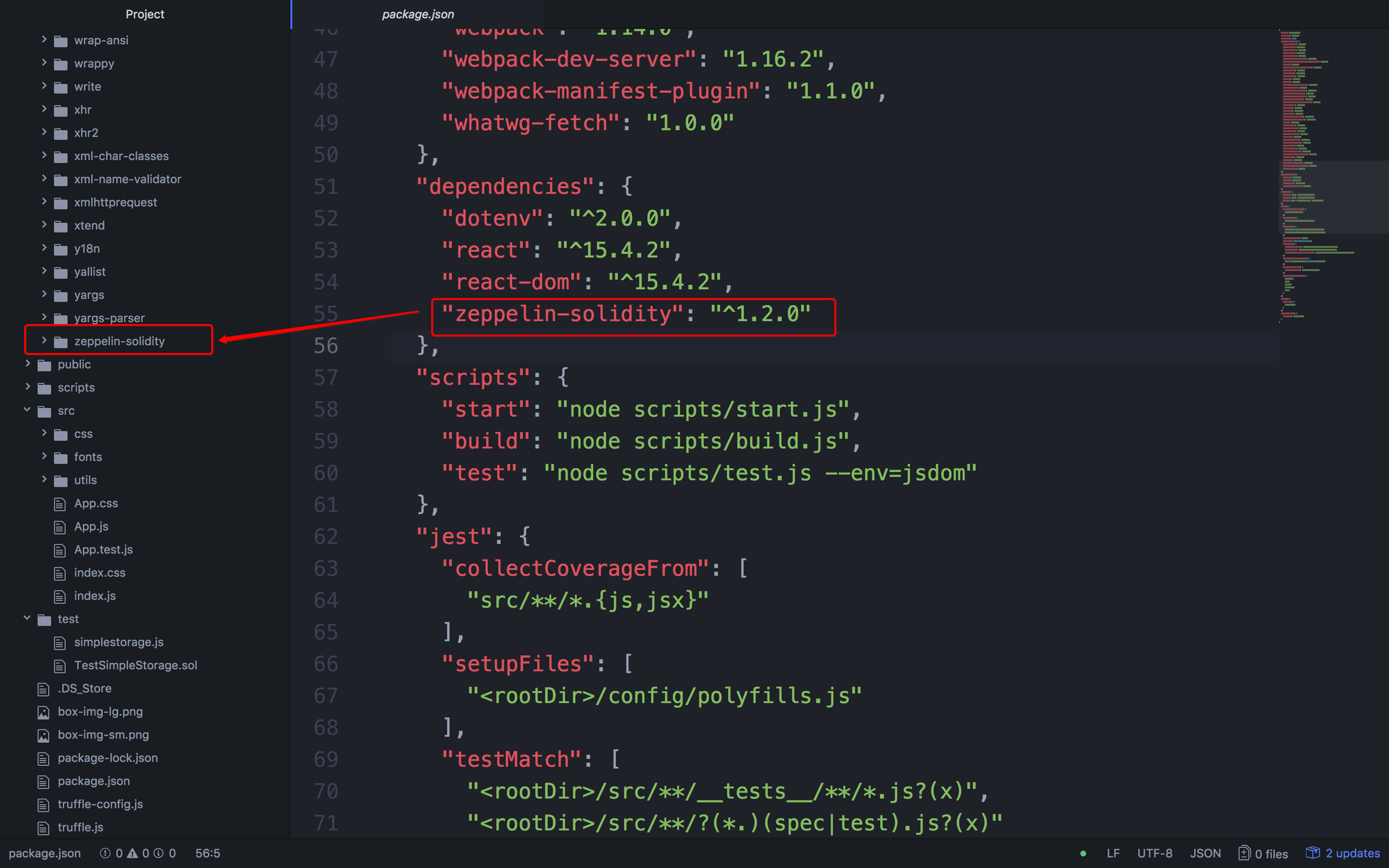
开发前的准备

* 打开终端，启动testrpc，继续通过testrpc模拟以太坊区块链测试环境。
* 创建的代币如果想要能够通过以太币钱包来进行转账和收帐，必须兼容于以太坊的ERC20标准，ERC20定义了支持钱包所必需的合约界面。
* 在本篇文章中，我们将安装OpenZeppelin来简化加密钱包开发的过程。OpenZeppelin是一套能够给我们方便提供编写加密合约的函数库，同时里面也提供了兼容ERC20的智能合约。

liyuechun:BloggerCoin yuechunli$ npm install zeppelin-solidity

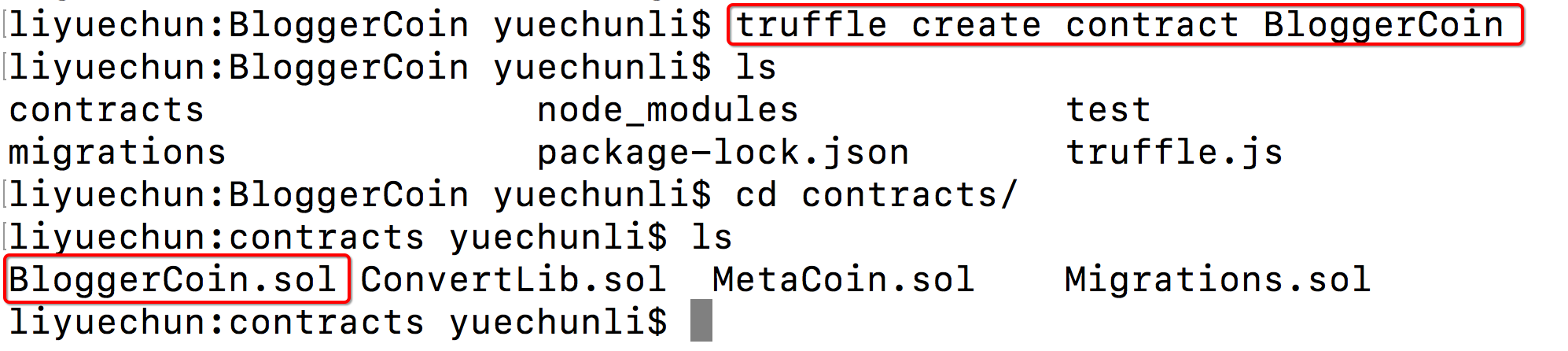
Atom打开项目查看zeppelin-solidity安装结果

通过Atom打开项目，在node\_modules中的最后一个文件夹就是zeppelin-solidity的内容。

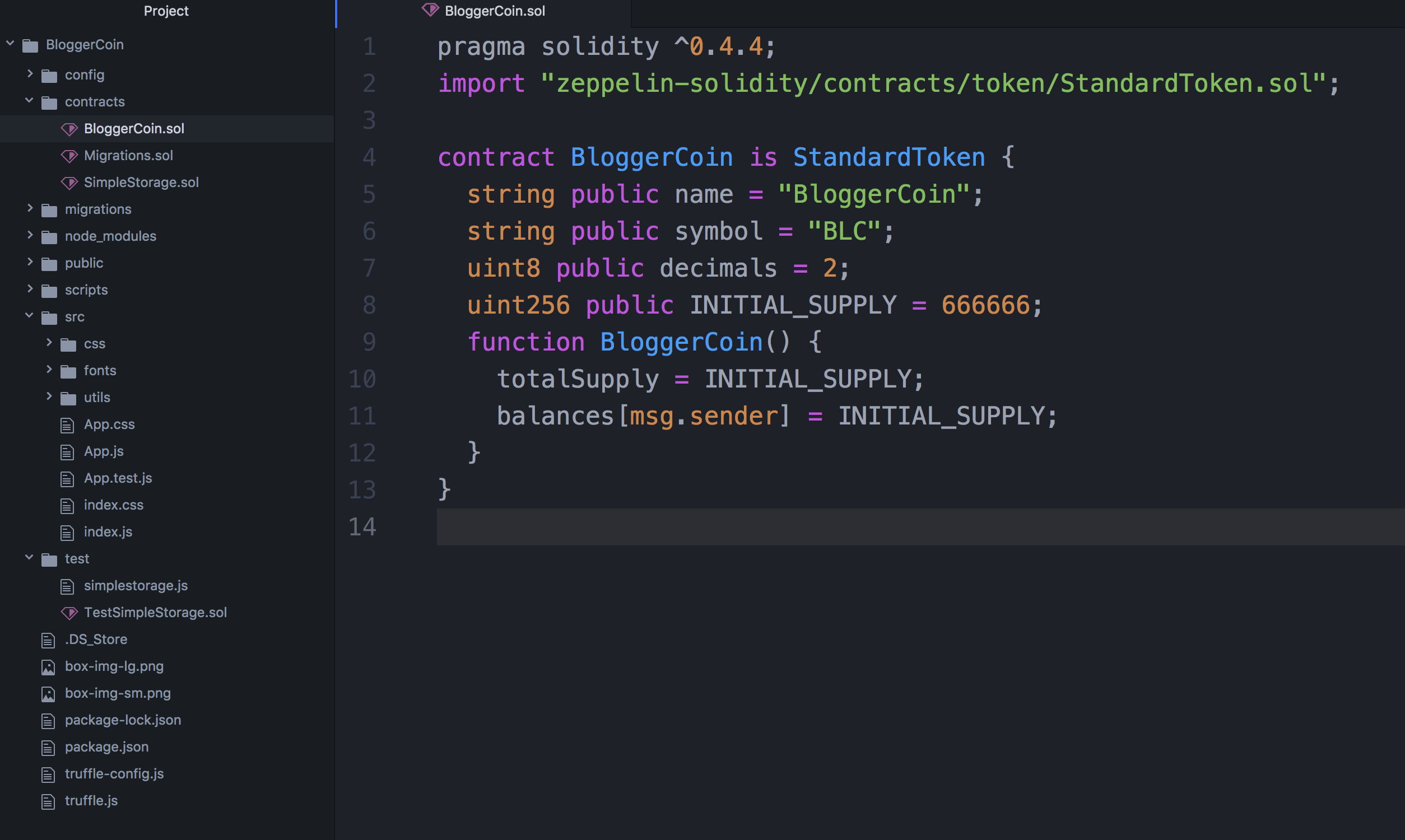


创建标准的「BLC」代币合约

在contracts/目录下建立一个BloggerCoin.sol文件。也可以使用以下命令来创建文件：



BloggerCoin.sol代码如下：



pragma solidity ^0.4.4;

import "zeppelin-solidity/contracts/token/StandardToken.sol";

contract BloggerCoin is StandardToken {

string public name = "BloggerCoin";

string public symbol = "BLC";

uint8 public decimals = 4;

uint256 public INITIAL\_SUPPLY = 666666;

function BloggerCoin() {

totalSupply = INITIAL\_SUPPLY;

balances[msg.sender] = INITIAL\_SUPPLY;

}

}

代码解释

pragma solidity ^0.4.4;

第一行代表solidity的版本，不同的版本编译的字节码不一样，^代表向上兼容，不过版本不能超过0.5.0。

import "zeppelin-solidity/contracts/token/StandardToken.sol";

这句代码是通过import来导入我们需要使用到的StandardToken合约。

contract BloggerCoin is StandardToken {

...

}

建立BloggerCoin合约时，让BloggerCoin合约直接继承自StandardToken。is既是继承。因此BloggerCoin继承了StandardToken所有的状态数据和方法。

当我们继承了StandardToken合约，也就支持了以下ERC20标准中规定的函数。

|  |  |
| --- | --- |
| **函数** | **方法** |
| totalSupply() | 代币发行的总量 |
| balanceOf(A) | 查询A帐户下的代币数目 |
| transfer(A,x) | 发送x个代币到A帐户 |
| transferFrom(A,x) | 从A帐户提取x个代币 |
| approve(A,x) | 同意A帐户从我的帐户中提取代币 |
| allowance(A,B) | 查询B帐户可以从A帐户提取多少代币 |

和之前一样，后面验证时会用到balanceOf和transfer两个函数。因为StandardToken合约中已经帮我们实现了这些函数，因此我们不需要自己从头再写一次。

string public name = "BloggerCoin";

string public symbol = "BLC";

uint8 public decimals = 4;

uint256 public INITIAL\_SUPPLY = 666666;

这边设定参数的目的是指定这个代币的一些特性。以人民币为例，人民币的名称（name）是RMB，美元的代号为¥，拿100元去找零最小可以拿到零钱是一分，也就是0.0001元。因为1元最小可分割到小数点后4位（0.0001），因此最小交易单位（decimals）为4。

这里将这个加密代币取名（name）为BloggerCoin（部落币），代币的代号（symbol）为BLC，最小分割单位是4（最小可以找0.0001个部落币）。

以下为人民币，比特币，以太币，部落币的对照表供参考：

|  |  |  |
| --- | --- | --- |
| **name** | **symbol** | **decimals** |
| RMB | ¥ | 4 |
| Bitcoin | BTC | 8 |
| Ethereum | ETH | 18 |
| BloggerCoin | BLC | 4 |

最后也定义了初始代币数目INITIAL\_SUPPLY。这里选择了一个吉祥数字666666。另外，当我们把全局变量设为public（公开），编译时就会自动新增一个读取公开变量的ABI接口，我们在truffle console中也可以读取这些变量。

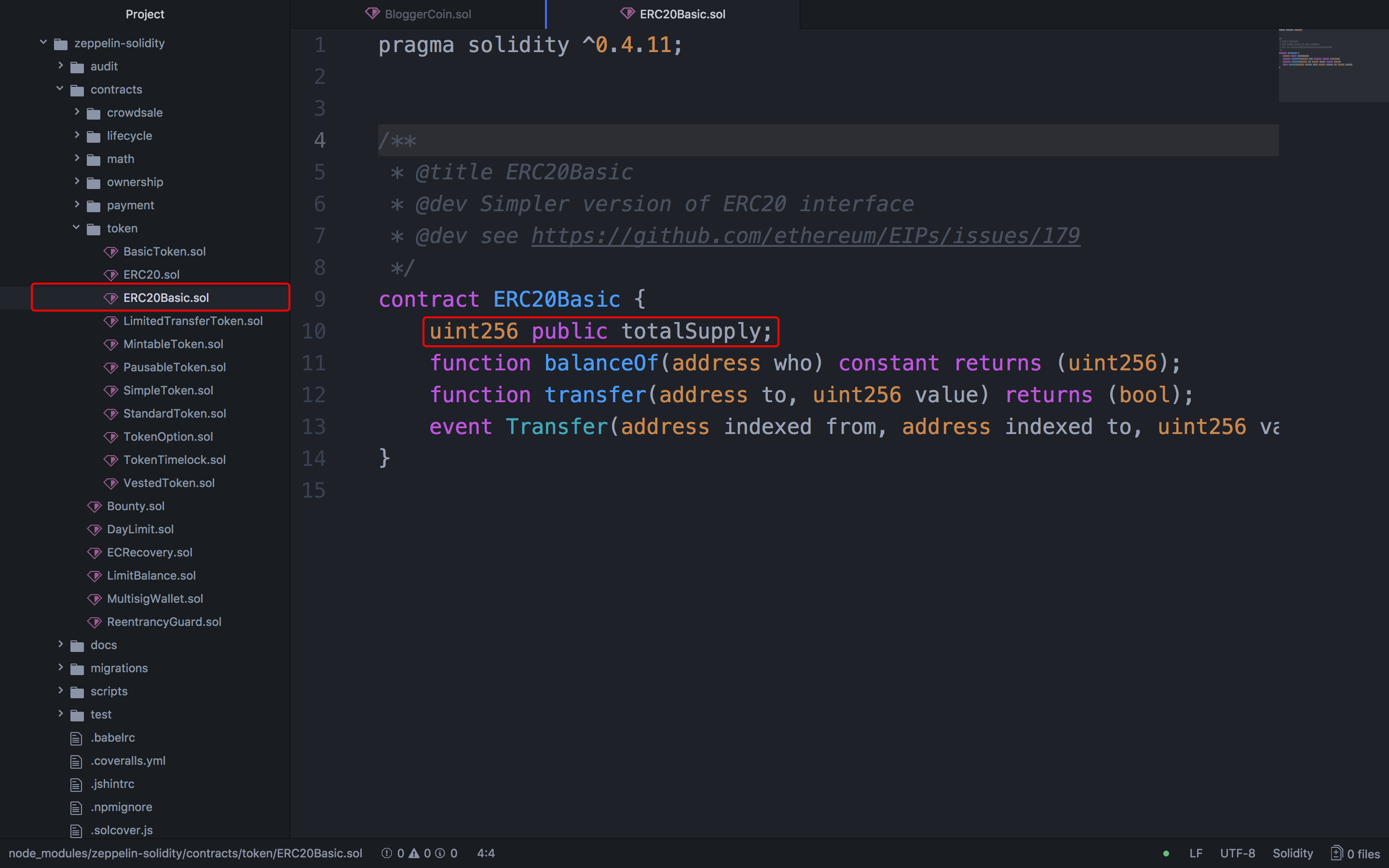
function BloggerCoin() {

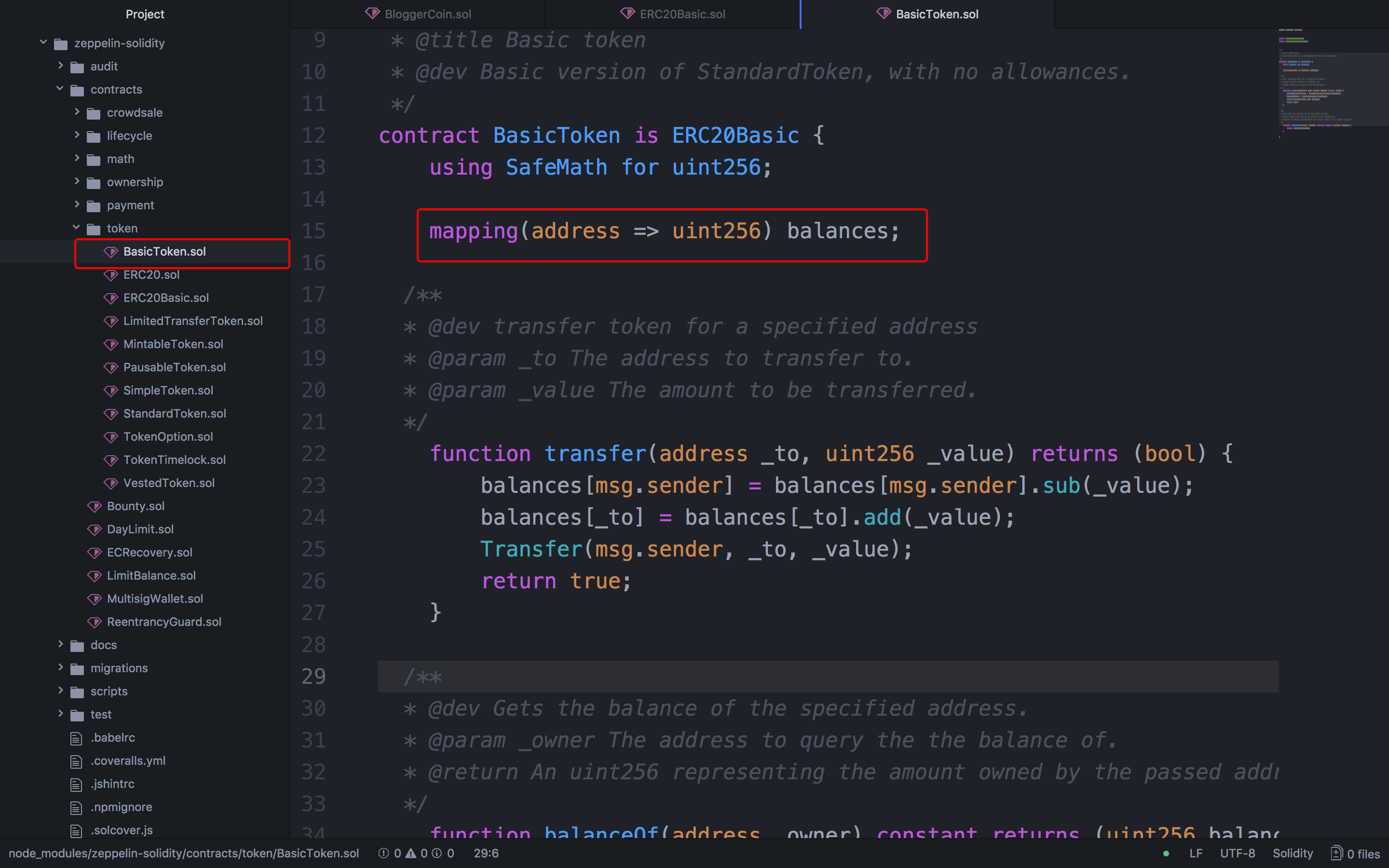
totalSupply = INITIAL\_SUPPLY;

balances[msg.sender] = INITIAL\_SUPPLY;

}

和合约同名的BloggerCoin方法，就是BloggerCoin合约的构造函函数（constructor）。在构造函数里指定了totalSupply数目，并将所有的初始代币INITIAL\_SUPPLY都指定给msg.sender帐号，也就是用来部署这个合约的帐号。totalSupply定义于ERC20Basic.sol中，balances定义于BasicToken.sol中。





pragma solidity ^0.4.11;

import './ERC20Basic.sol';

import '../math/SafeMath.sol';

/\*\*

\* @title Basic token

\* @dev Basic version of StandardToken, with no allowances.

\*/

contract BasicToken is ERC20Basic {

using SafeMath for uint256;

mapping(address => uint256) balances;

/\*\*

\* @dev transfer token for a specified address

\* @param \_to The address to transfer to.

\* @param \_value The amount to be transferred.

\*/

function transfer(address \_to, uint256 \_value) returns (bool) {

balances[msg.sender] = balances[msg.sender].sub(\_value);

balances[\_to] = balances[\_to].add(\_value);

Transfer(msg.sender, \_to, \_value);

return true;

}

/\*\*

\* @dev Gets the balance of the specified address.

\* @param \_owner The address to query the the balance of.

\* @return An uint256 representing the amount owned by the passed address.

\*/

function balanceOf(address \_owner) constant returns (uint256 balance) {

return balances[\_owner];

}

}

进一步追去看·BasicToken.sol合约的内容，可以发现BasicToken.sol合约中导入了SafeMath.sol合约。SafeMath`对各种数值运算加入了必要的验证，让合约中的数字计算更安全。

**如此一来，我们已写好一个可通过以太币钱包交易的新加密代币合约。这个合约一经部署，就可以一直存在于以太坊区块链上，世界上从此也就多了一种新的加密代币。只要你能找到人想拥有这种代币，这种代币就有交易的价值。**

编译、部署、验证

在migrations/目录下建立一个3\_deploy\_bloggerchain.js文件，内容如下：

现在执行compile与migrate命令

**备注：**确保testrpc处于运行状态。

truffle compile

/Users/liyuechun/Desktop/SmartContractDemo/BloggerCoin

liyuechun:BloggerCoin yuechunli$ truffle compile

Compiling ./contracts/BloggerCoin.sol...

Compiling ./contracts/Migrations.sol...

Compiling ./contracts/SimpleStorage.sol...

Compiling zeppelin-solidity/contracts/math/SafeMath.sol...

Compiling zeppelin-solidity/contracts/token/BasicToken.sol...

Compiling zeppelin-solidity/contracts/token/ERC20.sol...

Compiling zeppelin-solidity/contracts/token/ERC20Basic.sol...

Compiling zeppelin-solidity/contracts/token/StandardToken.sol...

Writing artifacts to ./build/contracts

liyuechun:BloggerCoin yuechunli$

* truffle migrate
* liyuechun:BloggerCoin yuechunli$ truffle migrate
* Using network 'development'.
* Running migration: 1\_initial\_migration.js
* Deploying Migrations...
* ... 0xac35fdd655a7b8916d5a43fb608227f1827aa666e4d4aa7b4d50347f8883de8a
* Migrations: 0x5c7102091425e16998b8bed1cd6634f499ab3684
* Saving successful migration to network...
* ... 0x1131a209a1ca27cadbec4ef8f84cecbe322e59d01b2b584f3e0ddada5a7a53d8
* Saving artifacts...
* Running migration: 2\_deploy\_contracts.js
* Deploying BloggerCoin...
* ... 0xc23199c5fe72206a5d74ad09797c9df17deb361c56ee1cb14b816ee0d874d5e2
* BloggerCoin: 0xbacb9b3da2e3140df11516be2244c4ea230d6d39
* Saving successful migration to network...
* ... 0x32bf4f5299bb4d260cc86da76591d9564376a82c4b8122261043d74a70c57b9e
* Saving artifacts...
* Running migration: 3\_deploy\_bloggerchain.js
* Replacing BloggerCoin...
* ... 0x87e8c7a24727a06da750a2c9f3b4ea1bc4b87c8c3e9c8a9219c3dada911e0991
* BloggerCoin: 0x5262d2b6de1a1187abdd203cb726b387bcd6140f
* Saving successful migration to network...
* ... 0x75166d7f6ee595437718df960d9a3bc76466bd890988a92b1aac1a396dc7f018
* Saving artifacts...
* liyuechun:BloggerCoin yuechunli$
* 验证
* liyuechun:BloggerCoin yuechunli$ truffle console
* truffle(development)> let contract
* undefined
* truffle(development)> BloggerCoin.deployed().then(instance => contract = instance)
* ......
* truffle(development)> contract.balanceOf(web3.eth.coinbase)
* { [String: '66666'] s: 1, e: 5, c: [ 666666 ] }
* truffle(development)> contract.balanceOf(web3.eth.accounts[1])
* { [String: '600000'] s: 1, e: 0, c: [ 0 ] }
* truffle(development)> contract.transfer(web3.eth.accounts[1], 600000)
* truffle(development)> contract.balanceOf(web3.eth.coinbase)
* { [String: '66666'] s: 1, e: 4, c: [ 66666 ] }
* truffle(development)> contract.balanceOf(web3.eth.accounts[1])
* { [String: '600000'] s: 1, e: 5, c: [ 600000 ] }
* truffle(development)>
* 验证过程中具体方法的讲解，请看这篇文章：[如何编写智能合约？（II） 建立简易的加密代币](http://liyuechun.org/2017/09/20/how-to-code-smart-contract-EncryptedToken/)

结语

我们用到OpenZeppelin来简化我们加密代币的开发，当然在正式的系统中，建议大家看看OpenZeppelin源码，检查一下是否还有缺陷，同时也可以从这个开源库中学到不少东西。