2019-03-21工作记录-安装nodejs

本机记录了在win10系统上安装nodejs的过程，并指导完成本机测试网络上部署合约过程。

请注意安装一些工具库时使用指定的版本，因为我第一次写本篇文章安装工具时使用了最新的版本，最后的结果是合约能部署但是无法send成功。后来重新安装指定版本运行正常了。

1 ATOM编辑器的安装

从网络搜索下载ATOM安装包，解压后是AtomSetup-x64.exe，直接运行安装即可。

2 下载安装nodejs

下载地址：<https://nodejs.org/zh-cn/download/>

选择长期支持版本，下载后文件是node-v10.15.3-x64.msi，直接安装。



安装完成后可以在命令行终端中检查nodejs的版本。

D:\code-nodejs\myproject>node -v

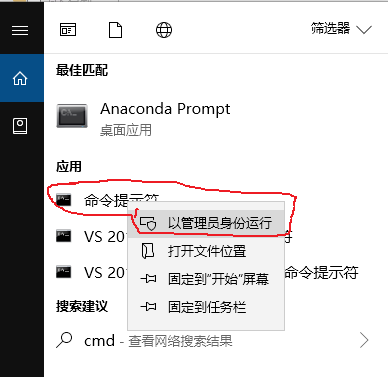
v10.15.3

D:\code-nodejs\myproject>npm -v

6.4.1

3 创建nodejs工程

1. 先在硬盘上建立工程目录：D:\code-nodejs\myproject， 在Atom软件中打开工程目录，此时显示空文件夹。
2. 重新以管理员身份启动cmd终端，这一步非常重要，后续的命令行安装命令非得要管理员身份才能成功。



1. 在终端下进入myproject目录，执行下列命令创建工程配置文件package.json。中间一路按回车就行了。

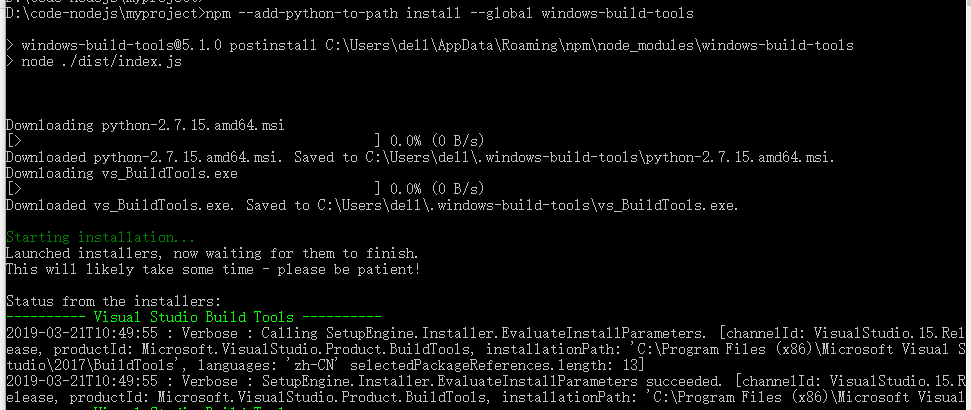
D:\code-nodejs\myproject>npm init

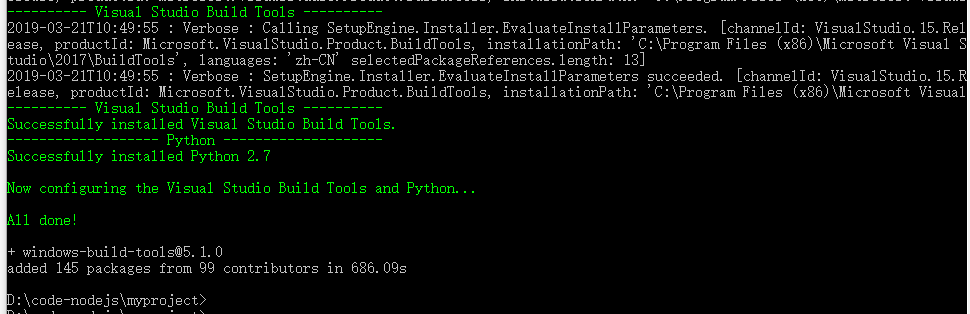
完成后就可以在Atom窗口中看到这个文件，并显示内容。

4 安装其他必须的依赖包（windows系统工具包、python包）

D:\code-nodejs\myproject>npm --add-python-to-path install --global windows-build-tools

参考链接<https://github.com/ethereum/web3.js/issues/1064>





5 安装web3包

windows平台下需要先安装git， 从<https://gitforwindows.org/>下载git-2.21.exe，安装教程：

<https://jingyan.baidu.com/article/8cdccae90beafa315413cd13.html>

安装完成后重新用管理员身份打开cmd，查看 git 是否安装成功,

C:\WINDOWS\system32>where git 查看 git 的安装路径

C:\Program Files\Git\cmd\git.exe

C:\WINDOWS\system32>git version 查看 git 版本

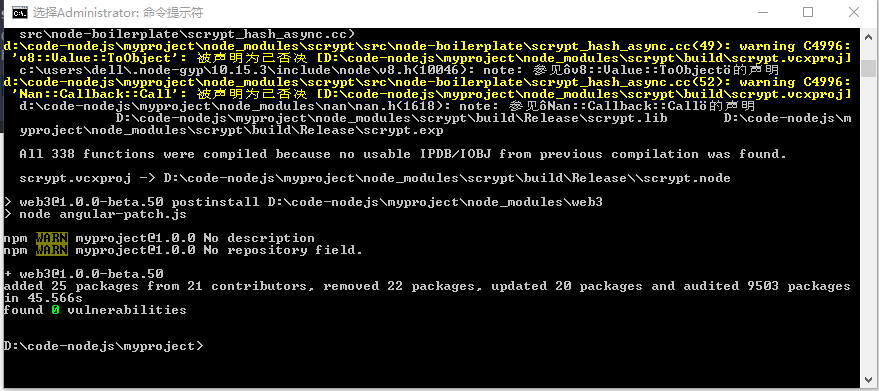
git version 2.21.0.windows.1

C:\WINDOWS\system32>

Web3这个包安装过程比较慢，需要耐心等待。Web3的作用是让我们的程序能与以太坊网络交互。

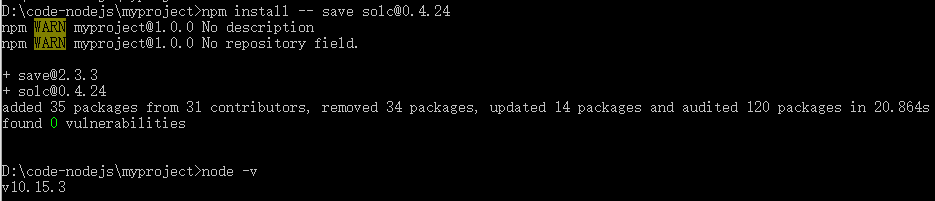
D:\code-nodejs\myproject>npm install --save web3@1.0.0-beta.36

目前安装后的版本是1.0.0-beta.36



6 安装最新的solidity编译器到本地项目中

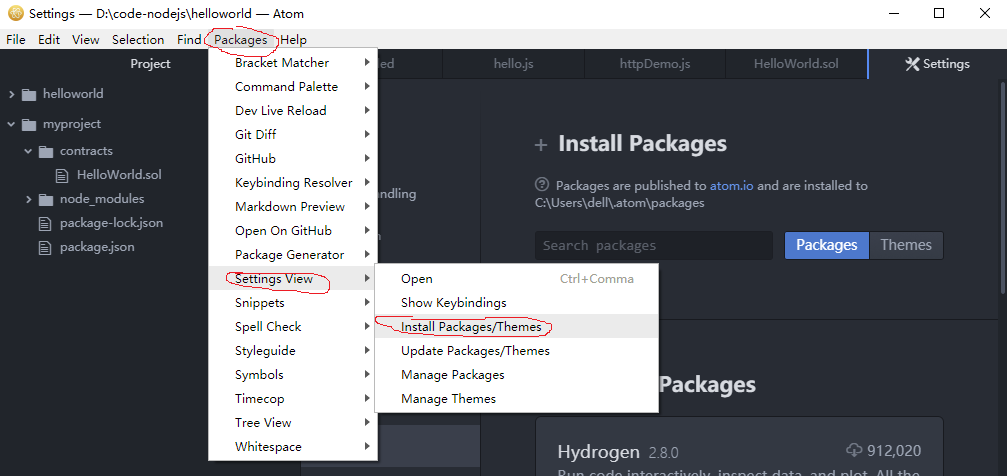
D:\code-nodejs\myproject>npm install -- save solc@0.4.25

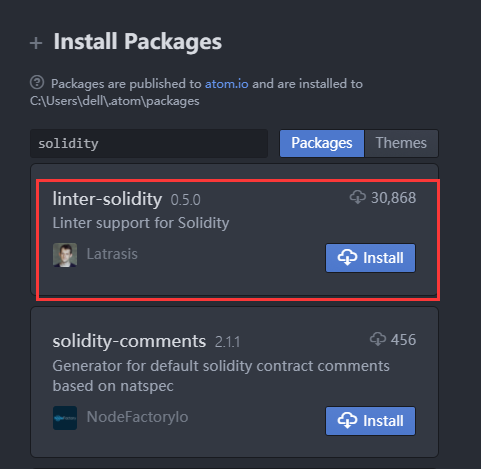


7 Atom中安装solidity插件，以后编写sol文件时就能识别关键字提示错误了。

菜单-Packages—Settings View – install Packages/Themes

搜索solidity，点击安装linter-solidity 0.5.0





8 安装测试用的mocha框架

D:\code-nodejs\myproject>npm install --save mocha@5.2.0

npm WARN myproject@1.0.0 No description

npm WARN myproject@1.0.0 No repository field.

+ mocha@6.0.2

added 195 packages from 577 contributors and audited 10642 packages in 60.384s

found 0 vulnerabilities

D:\code-nodejs\myproject>

9 安装测试网络框架ganache， 可以在本地生成一个测试网络，合约就能部署在这个测试网络上运行了。

D:\code-nodejs\myproject>npm install --save ganache-cli@6.1.8

npm WARN myproject@1.0.0 No description

npm WARN myproject@1.0.0 No repository field.

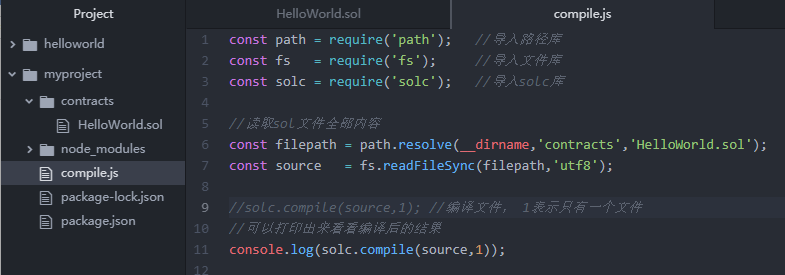
+ ganache-cli@6.4.1

added 54 packages from 46 contributors in 21.163s

D:\code-nodejs\myproject>

第二部分 在Atom中编写一个合约，在命令行下运行。

1 在工程目录下编写一个合约文件compile.js, 内容如下：



const path = require('path'); //导入路径库

const fs = require('fs'); //导入文件库

const solc = require('solc'); //导入solc库

//读取sol文件全部内容

const filepath = path.resolve(\_\_dirname,'contracts','HelloWorld.sol');

const source = fs.readFileSync(filepath,'utf8');

//solc.compile(source,1); //编译文件， 1表示只有一个文件

//可以打印出来看看编译后的结果

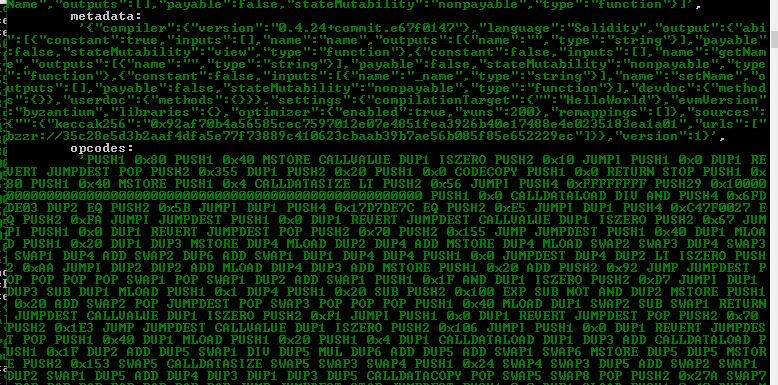
console.log(solc.compile(source,1));

2 在CMD中运行：

D:\code-nodejs\myproject>node compile.js

显示部分结果如下：





注意看这个结果是json格式，第一项contracts表示合约条目，在我们的HelloWorld.sol文件中只有一个合约，因此这里只有一项。这个合约的名字比较特殊，是':HelloWorld'，bytecode就是编译后的字节码，opcodes是汇编指令，用于EVM运行使用。

要想只显示这唯一的HelloWorld合约，可以这样写代码：

console.log(solc.compile(source,1).contracts[':HelloWorld']);

3 使用mocha框架测试：

新建test/mocha.test.js, 内容如下：

const assert = require('assert');

class Person{

eat(){

return "eat";

}

walk(){

return "walk";

}

}

//改进：把me提出来放到beforeEach中

var me;

beforeEach(()=>{

me = new Person();

});

describe('person',()=>{

it('can eat',()=>{

// var me = new Person();

assert.equal('eat',me.eat());

});

it('can walk',()=>{

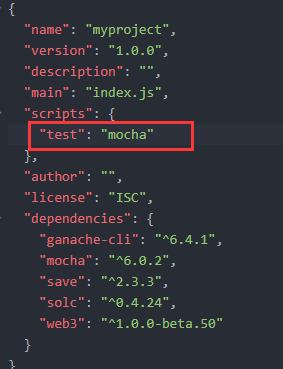
// var me = new Person();

assert.equal('walk',me.walk());

});

});

修改package.json文件的test字段内容， 修改后结果：



在cmd中运行测试指令，将会执行test目录下的js文件：

D:\code-nodejs\myproject>npm test

> myproject@1.0.0 test D:\code-nodejs\myproject

> mocha

person

√ can eat

√ can walk

2 passing (14ms)

D:\code-nodejs\myproject>

4 第3步操作只是验证了mocha可用，现在就要把mocha和ganache测试网络连接，在测试网络上部署合约。

新建文件test/helloworld.test.js, 文件内容如下：

const ganache = require('ganache-cli');

const Web3 = require('web3'); //大写的Web3是导入的类

//这里就是构造函数，把ganache测试网络和web3库连接起来，生成一个对象小写的web3.

const web3 = new Web3(ganache.provider());

/\* 方法1

//这种写法是笨拙的同步方法，强制等待getAccounts()完成后再继续执行打印操作。

//原因是web3.eth.getAccounts()函数是一个异步函数，发起请求后就直接返回了，还没有收到结果。

beforeEach(()=>{

web3.eth.getAccounts().then(fetchAccounts=>{

console.log(fetchAccounts);

});

});

\*/

//方法2 用异步和同步关键字, 先声明beforeEach中包含许多异步函数，再用await强制等待执行完毕再返回。

beforeEach( async()=>{

fetchAccounts = await web3.eth.getAccounts();

console.log(fetchAccounts);

});

describe('HelloWorld',()=>{

it('deploy contract',()=>{});

});

在cmd进行测试npm test, 会同时运行test目录下的两个js文件。一大堆绿色的就是测试网络中自动产生的账户地址。



5 经过上面测试后说明mocha、ganache-cli、web3、solc都运行正常，继续修改helloworld.test.jsl文件，最终内容如下：

const ganache = require('ganache-cli');

const Web3 = require('web3'); //大写的Web3是导入的类

//这里就是构造函数，把ganache测试网络和web3库连接起来，生成一个对象小写的web3.

const web3 = new Web3(ganache.provider());

const assert = require('assert');

// 推荐方法：

const {bytecode, interface} = require('../compile');

//改进的方法3, 用字节码和API接口创建合约对象，指定创建合约的外部账户和燃料限制

var helloworld;

var fetchAccounts;

beforeEach( async()=>{

fetchAccounts = await web3.eth.getAccounts();

// console.log(fetchAccounts);

helloworld = await new web3.eth.Contract(JSON.parse(interface)).deploy({data:bytecode, arguments:['Tom']}).send({from:fetchAccounts[1],gas:'1000000'});

//console.log(helloworld);

});

describe('HelloWorld',()=>{

it('deploy contract', async ()=>{

//如果指定的对象存在就OK，不存在就失败

assert.ok(helloworld.options.address);

});

//调用静态方法

it('call static function', async ()=>{

const msg = await helloworld.methods.getName().call();

assert('Tom',msg);

});

//调用动态方法

it('call dynamic function', async ()=>{

await helloworld.methods.setName('Olaya').send({from:fetchAccounts[1]});

const msg = await helloworld.methods.getName().call();

assert('Olaya',msg);

});

运行测试结果：

