以太坊智能合约编写——从入门到放弃

1. 前言
2. 区块链基础
   1. 为什么使用区块链

区块链用在需要在多方之间共享数据以及传递价值而不需要信任彼此的场景下。

金融世界把这种信任叫做交易对手风险：这种风险下另一方不愿意承担风险。区块链通过一个具有数学，密码学，对等网络的革命性系统革除了这种风险。

**第一个数据库**

20世纪60年代第一个计算机数据库诞生了。随着硬件的分布式以及互联网历史进程的发展，数据自然而然地存在了中心化的物理环境。这种中心化的方法意味着数据的存储访问需要被一个中心化的权威控制。

中心化系统可以在内部或者外部被访问，因此我们不得不相信系统的拥有者有足够的意愿和资源来整合我们的数据并保证数据安全。中心化数据库现在仍然很普遍，支配者我们线上线下的大部分应用。

Self-hosted 日志是中心化数据库的一个常见例子。所有者可以在事后编辑帖子或对用户进行审查，而无需求助。相应地黑客也可以恶意攻击系统。如果没有数据库备份那么恢复被破坏的数据变得不可能。

**数据共享的需求**

共享大数据既昂贵又很不方便。我们可以通过分布式数据来减轻这种负担。读写数据被一个或多个实体控制，但是仍然存在中心化数据库中诸如腐败等问题。现代共享数据库采用技术来减少这种问题。其中的一些想法与区块链不谋而合。基于共享数据库的系统可能有以下特点：

* 不变性：复制一份旧数据作为历史记录而不是覆盖旧数据。可以访问数据来证明某些数据在特定时间存在过。
* 一致性：对于一份要被共享的数据，各方必须认同其内容。有很多方法达成一致性，比如将要讨论的POW。

区块链继承发扬这些方法来解决这些信任问题。

* 1. 区块链是什么

根本上说区块链是一份共享数据，包含了交易的账本。很像银行，简单区块链的账簿记录了货币(在这种情况下，是加密货币)的所有权。每一个连接到链的设备都有一个账本的备份，这叫做节点。区块链通过以下方式消除了影响数据库信任的问题：

* 彻底的去中心化：读写数据库彻底去中心化并且安全。没有一个个人或者组织可以控制区块链。
* 高度的容错能力：容错能力不是区块链所独有的，区块链通过每个账户共享数据来实现这个逻辑。
* 独立验证：交易可以被每个人验证而不需要第三方。所谓的脱媒。
  1. 区块链如何工作

区块链网络账户之间的交互叫交易。可以是货币交易比如发送以太。也可以是数据交易比如一条评论。打捆的交易叫块(block)。

区块链上每一个账户都有一个独一无二的签名，让所有人知道谁创建的交易。在一个公链上每个人都可以读写数据，读数据免费写数据收费。这种消费叫做“gas”，用以太币购买，有助于网络安全。

**挖矿**

网络上的任何节点都可以通过一个名为“挖掘”的过程来参与保护网络。成为矿工的节点需要解决一个数学问题来获得记账权。

因为挖矿需要算力，矿工会获得补偿。赢得记账权的矿工可以获得加密货币作为回报。这就激励了矿工的工作，以确保网络的安全，防止太多的控制权落入任何单一的采矿者手中。

**哈希**

一旦一个新块被挖出来，其他的矿工就会被通知到这个消息然后开始验证并把这个新块复制到自己链上。这通过密码学的哈希完成。哈希是一种单向函数，这种函数输入数据并返回一个固定长度的字符串。

然而并不能从哈希函数中推断出原始数据，相同的输入一定有相同的输出。这样可以哈希未经验证的数据并和原数据比较。如果匹配那么验证通过。

一旦一半以上的矿工验证了新块，就认为网络达成了公式，这个块就永久成为区块链历史的一部分。现在这个数据可以被所有的节点下载，因为已经被核实有效。

1. 以太坊区块链
   1. 以太坊是什么

以太坊是一个允许你跑程序的可信环境。比特币区块链的合约只能让你管理加密货币。

为了达到目的，以太坊有一个虚拟机，叫做EVM。EVM允许核实代码并且在区块链上执行，确保在每个人的机器上都以相同的方式执行。这段代码叫做“智能合约”。

除了追踪账户余额以外，以太坊维持区块链上EVM的状态。所有节点都处理智能合约来验证合约和输入的完整性。

* 1. 智能合约是什么

智能合约是运行在EVM上的代码。智能合约可以接受存储以太币，数据或者二者的打包。然后，根据合同中的逻辑，它可以将该以太币分配给其他帐户，甚至是其他智能合约。

以太坊智能合约用solidity语言编写。Solidity是静态类型的，它支持继承、库和复杂的用户定义类型。Solidity语法类似于JavaScript。

* 1. 以太坊网络

到现在为止我们已经以太坊公链的主要部分。在主网上，数据在链上——包含了账户余额和交易——这些是公开的，任何人都可以创造节点并开始验证交易。网络上的以太币有市场机制并且可以用其他加密货币或者诸如美元的发币交换。

但是也有别的网络。事实上每个人都可以创造自己的以太坊网络。

**最新的测试网络**

以太坊区块链可以在本地模拟一个环境。本地测试网络可以即时处理交易，以太坊可以按需分配。有一些以太坊模拟器，[Ganache](http://truffleframework.com/ganache/)(<http://truffleframework.com/ganache/>)

**公共测试网络**

开发者可以使用公开测试网络来测试以太坊应用在最终部署到主网络之前。网络上的这些以太币只是用来测试但是没有什么价值。

有三种广泛使用的测试网络：

* Ropsten：官方测试网络，以太坊基金会发布，功能上类似主网络。
* Kovan：一个使用“POA”一致性算法的网络。这意味着交易可以被选出的其他成员验证，持续4秒。这个测试网络上的以太坊应用也用来减轻供给。
* Rinkeby：一个也用POA的测试网络，以太坊基金会发布。

**私有网络/企业网络**

私有的Ethereum网络允许各方共享数据而不使其公开访问。一个私有区块链是一个很好的选择:

* 共享敏感数据，比如健康记录。
* 由于网络规模较小，扩展处理更高的读/写吞吐量。
  1. DApp

一个用智能合约处理程序的应用叫分布式应用，或者叫dapps。这些dapps的用户界面包括大家熟悉的语言，如HTML、CSS和JavaScript。应用程序本身可以托管在传统的web服务器上，也可以在分布式文件服务(如群集或IPFS)上。考虑到Ethereum区块链的好处，dapp可以成为许多行业的解决方案，包括但不限于：

* 记账
* 金融
* 供应链
* 不动产
* 市场

创建dapp的最好方法是测试一下，并且部署到你自己选的以太坊网络。当然是[Truffle](http://truffleframework.com/docs/getting_started/project)。

1. Solidity介绍与开发环境搭建
   1. Solidity介绍
   2. Solidity开发环境搭建

下文的指令均在ubuntu环境下执行，没有试过linux的其他发行版。

git clone --recursive https://github.com/ethereum/solidity.git

cd solidity

mkdir build

cd build

cmake .. && make

sudo ./scripts/build.sh

输入 solc –help看到帮助信息说明已经装好了

安装**最新版**的npm和nodejs。

官方推荐的IDE是Remix。Remix可以以在线方式使用，地址为<https://remix.ethereum.org/>。也可以本地方式，不过需要预先安装，安装过程（ide随时会更新，建议下载前看一下readme的说明，以前发生过ide弃坑另立github仓库的事情）：

git clone <https://github.com/ethereum/remix-ide.git>

cd remix-ide/

npm install

npm run pullremix

npm run linkremixcore

npm run linkremixlib

npm run linkremixsolidity

npm run linkremixdebugger

* 1. Solidity使用（在线版）

以下面这段代码为例：

pragma solidity ^0.4.0;

contract HelloWorld{

uint balance;

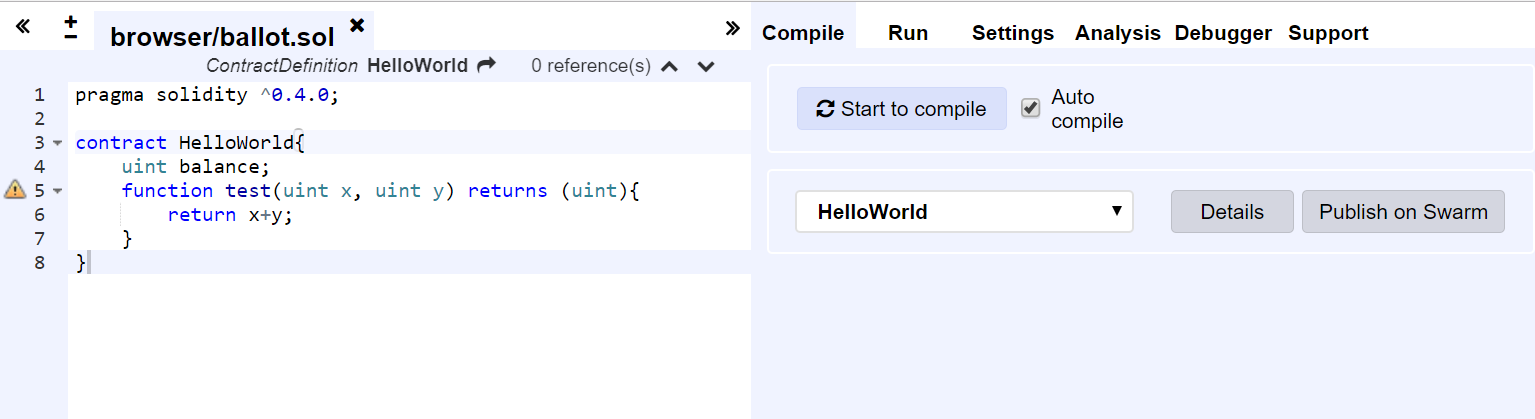
function test(uint x, uint y) returns (uint){

return x+y;

}

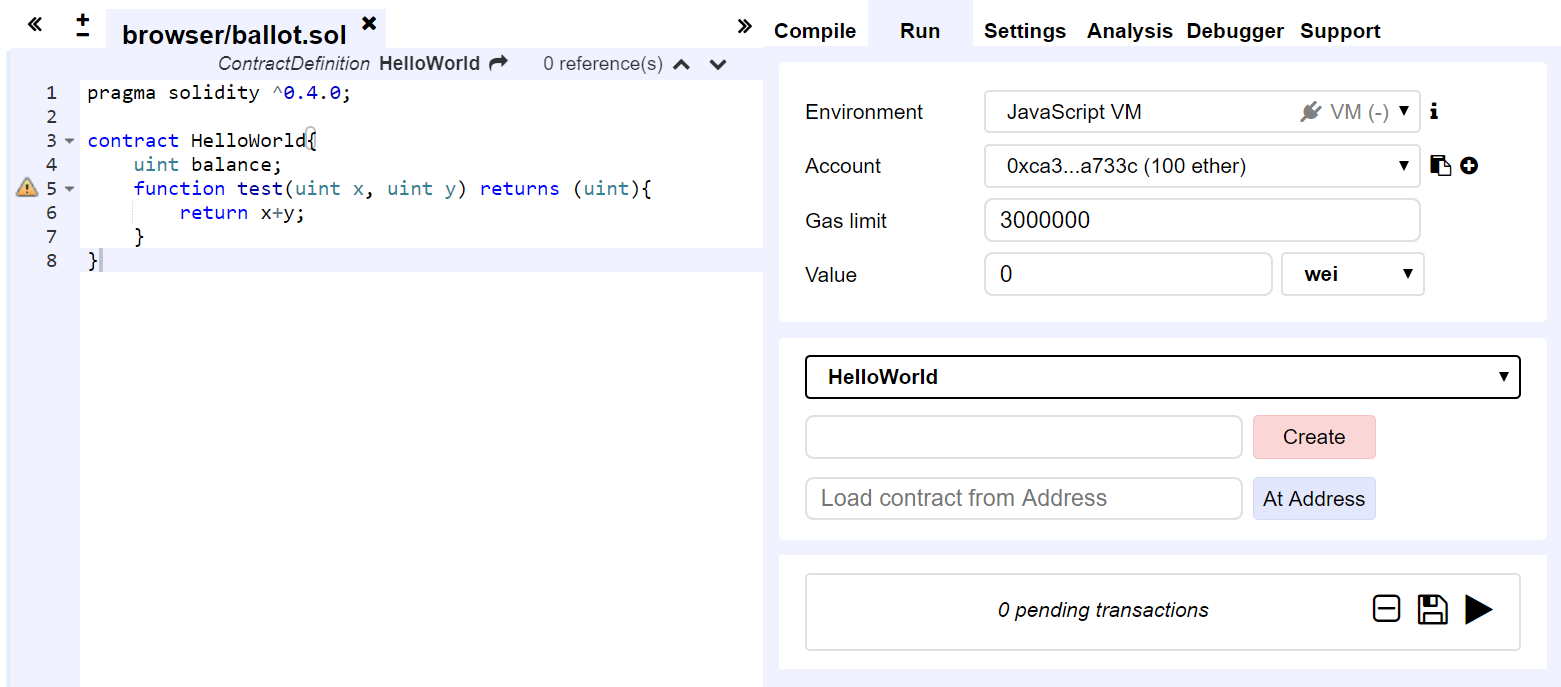
}

进入编译器ol页面。

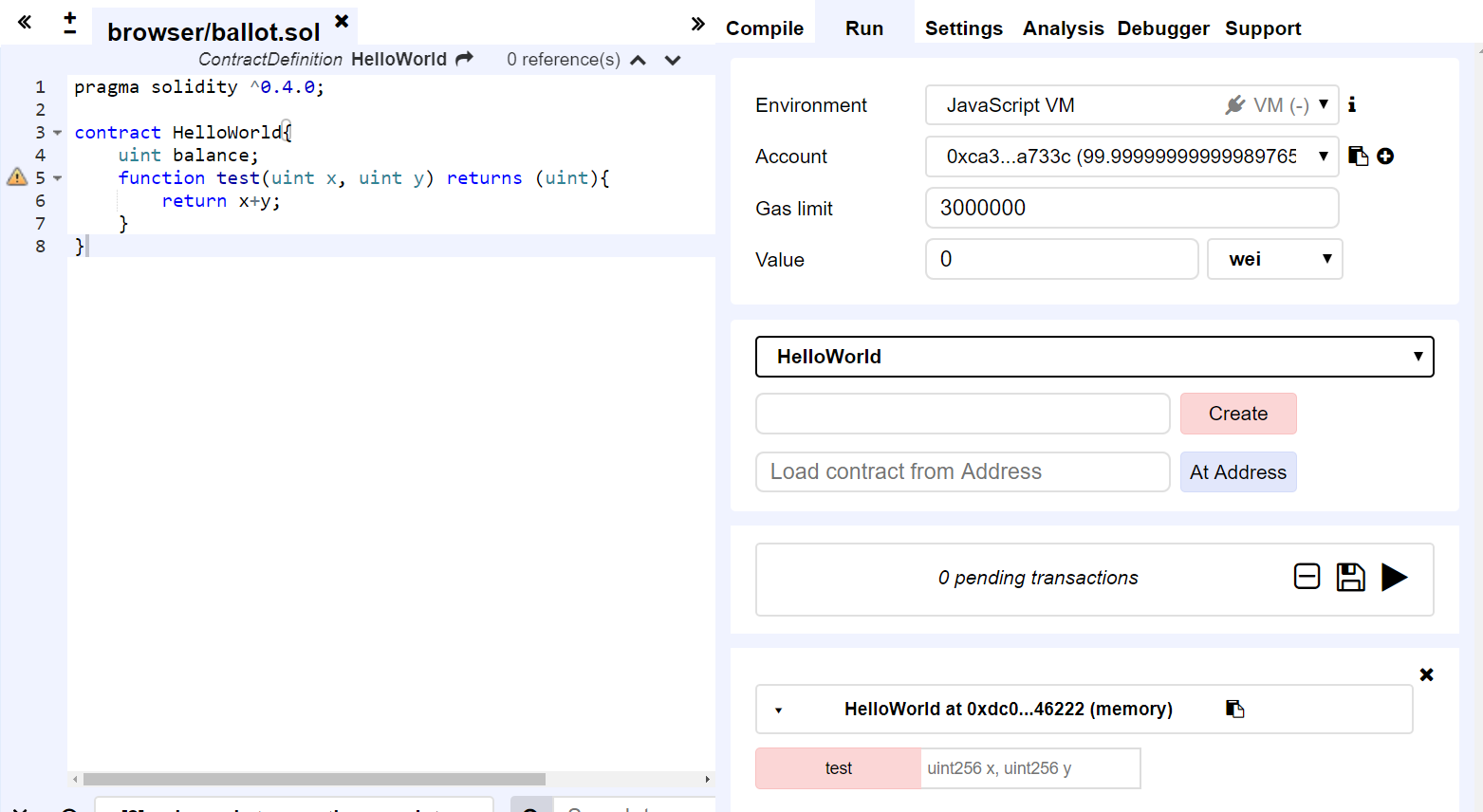


左边的框框输入代码，右边点击**start to compile.**

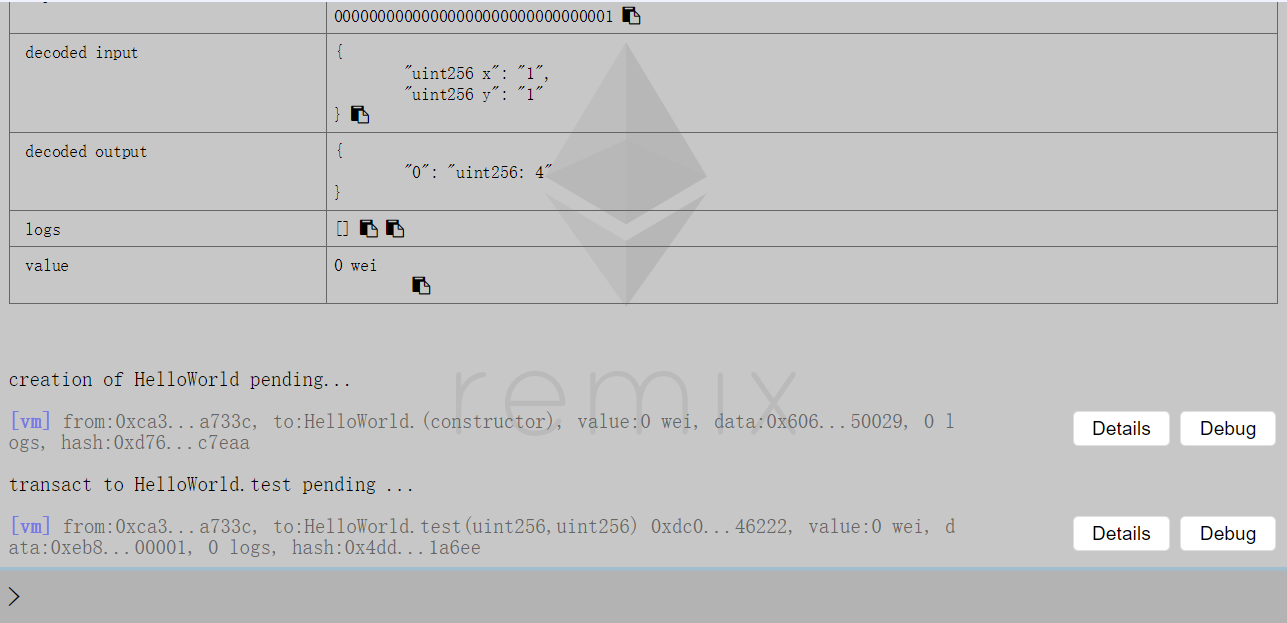
如果想要测试数据，选择**run**选项卡。点击**Create。**



发现下边多了一个输入框 test，也就是代码中的test函数。



在test中输入数据，比如1,1，点击test。在底部控制台可以看到



点击**detail**就可以看到输出结果。如果修改了代码就把上述过程再执行一遍。

1. Solidity基础
   1. Solidity源代码文件结构

一个sol文件的代码大概可以分成四个部分。

1. 指定solidity版本号（必须）

e.g. pragma solidity ^0.4.0;

1. 导入所需的其他源代码文件（非必须）。

e.g. import "filename";这种形式会把所有的filename中的全局变量导入

e.g. import \* as symbol from “filename”; 建立一个全局变量symbol，其成员全部来自filename的全局变量

e.g. import {symbol1 as alias, symbol2} from "filename"; 创建新的全局变量alias 和symbol2

导入文件时注意路径前的点。.代表当前路径 ..代表上一级路径

1. 注释（非必须），用来说明文件内容作者日期一类的信息。

e.g.

*// This is a single-line comment.*

*/\**

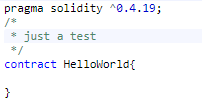
*This is a*

*multi-line comment.*

*\*/*

1. 合约体代码
   1. 合约

Solidity的代码都要在一个合约（contract）中实现，contract类似Java中class的概念



* 1. 基本数据类型
* **bool**：值为true false。支持 && || != ==
* **int/uint:** uint有uint8,uint16，uint24，……uint256，uint默认是256.支持比较，位运算，四则运算，取模，乘方（\*\*）.除法截断。
* **fixed/ufixed：**小数与无符号小数。
* **address：**以太坊地址，160bit也就是20字节，可用uint160表示。支持数学上的比较运算符。

**属性：**balance：获取address余额。如果获取当前合约余额只要this.balance.

**函数：**send：给指定地址打钱

* **字节数组：**byte1,byte2……byte32.支持比较运算符和位运算。可以用.length获取长度。bytes和string是一种特殊的数组。前者类似byte[]。而string不支持获取成都和按序号访问。bytes和string可以自由切换，
* **定长字节数组：**
* **枚举：**关键字enmu。可以将整数转为枚举元素或者反过来。例如:

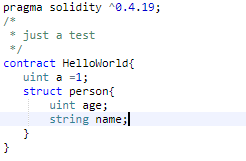
enum coin{btc,eth,eos,xrp}

* 1. 控制语句

支持if else while do for break continue return 不支持switch

* 1. 结构体

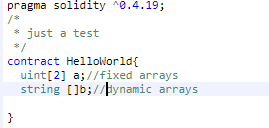
Solidity中结构体类似C/C++中的结构体。



* 1. 数组

Solidity有两种数组，静态数组和动态数组。

静态数组在声明时必须指定数组长度。而动态数组长度不固定可以动态添加元素。



对于一个动态数组如果我们想要加入数据可以使用push

可以用new创建一个memory数组。不能通过.length长度修改数组大小属性。

* 1. 函数与函数访问控制

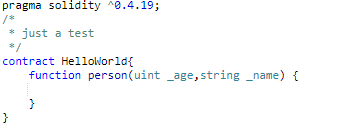
Solidity定义函数格式为：

function 函数名（参数列表）修饰符 **returns**(返回值类型){

函数体内容；

}//注意返回用的是returns而不是return，在函数体返回值时用return

按照惯例函数参数名最好以\_开头以区分全局变量，如果不按照这个做也不会报错。



函数权限控制符：

* **external:**

可以从其它合约或通过交易来发起调用。一个外部函数不能通过内部的方式来发起调用。this.表示外部访问。

* **public:**

可以通过内部，或者消息来进行调用。public是默认访问属性。

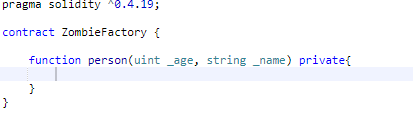
* **internal：**

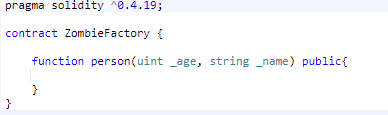
只能通过内部访问。即当前合约或继承的合约里调用。

* **private：**

私有函数和状态变量仅在当前合约中可以访问，在继承的合约内，不可访问。

同C++/Java的函数修饰符不同，Solidity 的修饰符要放在函数定义的中间，位于括号和returns之间。



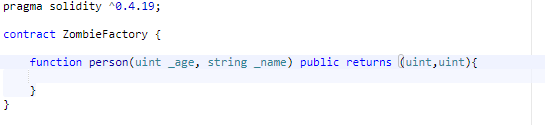


除此之外与C++/Java不同，Solidity还有view与pure属性，internal与external属性。

View属性：函数只能读取数据而不会改变应用里的数据。而pure意味着函数不能读取应用里的数据，数据只能来源于输入的参数。

如果一个函数用internal修饰，那么继承该合约的合约可以访问这个函数。而external只能被合约之外调用。

最后说一下返回值，Solidity的返回值关键字是returns，后边加上括号表示返回值类型，支持多值返回。



* 1. 数学和加密函数

keccak256:

使用以太坊的（Keccak-256）计算HASH值。紧密打包。

sha3:

等同于keccak256()。紧密打包。

sha256:

使用SHA-256计算HASH值。紧密打包。

ripemd160:

使用RIPEMD-160计算HASH值。紧密打包。

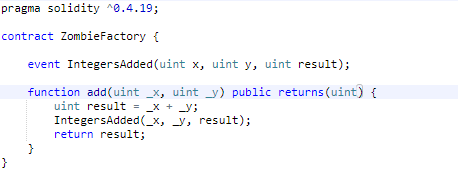
ecrecover:

通过签名信息恢复非对称加密算法公匙地址。如果出错会返回0，附录提供了一个例子[1](http://www.tryblockchain.org/Solidity-MathematicalAndCryptographicFunction-%E5%86%85%E7%BD%AE%E5%87%BD%E6%95%B0.html#fn1).

**紧密打包（tightly packed）**：

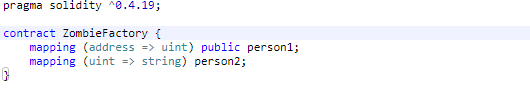
* 1. 事件

事件是合约与区块链通讯的一种机制。一旦监听到某些事件就可以做出反应。事件发生后会被记录到区块链上成为日志。事件关键字是event。



* 1. 映射

Solidity的映射也是一种kv的数据存储方式，不过形式与其他语言不太一样。mapping( key的type => value 的type) 修饰符 变量名。如下图：



* 1. delete符

将元素充值，如果是整数就是0，变长数组则长度为0。

* 1. 类型推断

有些时候不需要指定一个变量的类型，比如在循环节中。关键字为var。

例如：

uint count = 0;

for (var i = 0; i < 20; i++) {

count++;

}

但是函数的参数和返回参数不能使用var。

* 1. 时间单位

时间单位有seconds,minutes,hours,days,weeks,years,默认seconds。

* 1. 特殊的全局变量和全局函数

主要用于提供区块信息。

* block.blockhash(uint blockNumber) returns (bytes32)，给定区块号的哈希值，
* block.coinbase (address) 当前块矿工的地址。
* block.difficulty (uint)当前块的难度。
* block.gaslimit (uint)当前块的gaslimit。
* block.number (uint)当前区块的块号。
* block.timestamp (uint)当前块的时间戳。
* msg.data (bytes)完整的调用数据（calldata）。
* msg.gas (uint)当前还剩的gas。
* msg.sender (address)当前调用发起人的地址。
* msg.sig (bytes4)调用数据的前四个字节（函数标识符）。
* msg.value (uint)这个消息所附带的货币量，单位为wei。
* now (uint)当前块的时间戳，等同于block.timestamp
* tx.gasprice (uint) 交易的gas价格。
* tx.origin (address)交易的发送者（完整的调用链）
  1. 异常

有些情况需要跑出异常但是不能捕获异常。举例：

function test(uint i) payable returns (uint ret) {

if(i == 0)

throw;

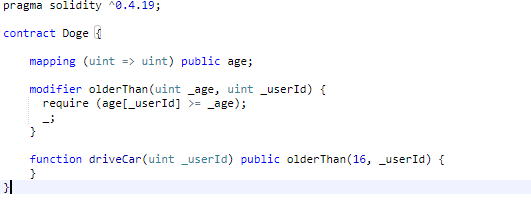
return i+1;

}

* 1. modifier

有些资料翻译成修改器，不过我觉得这个东西类似权限修饰符，但是又不符合modifier的原意，索性不翻译了。

比如我们写一个开车的函数，设定必须18岁以上的才能开车。就可以像下图这样写：



1. Solidity进阶
   1. address

以太坊区块链有很多账户，每个账户有一个自己的地址，通过地址与其他账户交付接受以太币，这一点类似银行转账。

* 1. msg

msg.data (bytes):完整的calldata

msg.gas (uint):剩余的汽油

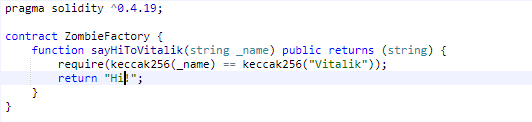
msg.sender (address):消息的发送方(当前调用)

msg.sig (bytes4):calldata的前四个字节(即函数标识符)

msg.value (uint):所发送的消息中wei的数量

* 1. require

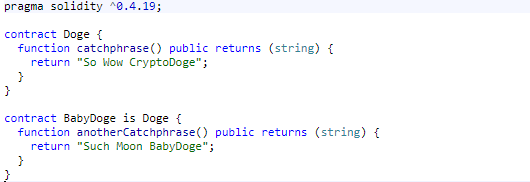
require判断参数内的条件是否满足，如果满足则继续执行后边的代码否则函数执行停止。



注意，solidity不支持原生字符串直接比较，因此只能比较其hash值。

* 1. 继承

Solidity继承的关键字不是extend而是is。



* 1. storage与memory

solidity有两个地方可以存储变量。永久存储在区块链和临时存储。

Storage:永久存储在区块链上，存了就不能动。这需要在每个节点都存入数据，代价是相当大的。

Memory:临时存储，如同计算机中的ram。

在声明变量时候把storage或者memory放在变量类型与变量名之间。

* 1. 与其他合约交互

如果想要与链上的其他合约交互就要定义一个接口。

* 1. fallback函数

每个合约都有一个无名无参无返回的三无函数，如果调用合约时没有匹配上任何一个函数就会调用这个三无函数。这个三无函数叫fallback函数。

1. Solidity高级编程
   1. 智能合约永恒性

合约一旦上传到以太坊以后就会不可更改。这也意味着代码无法更新调整。

* 1. 智能合约所有权
  2. Gas及其优化

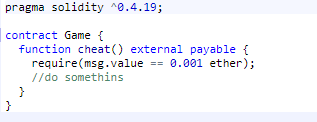
Gas是以太坊特有概念。用户每次执行DApp都需要一定的gas，gas可以通过以太币购买，gas的收费多少取决于程序的复杂程度。因此对程序的优化显得格外重要。

以太坊为什么这样做呢？当前的以太坊运行速度还很慢，一旦有用户写了死循环出来那么整个网络就拥塞了或者用户用密集运算占用资源也会影响网络。

使用view可以优化gas消耗，从外部调用view函数不需要支付gas，因为只是读取，运行这个函数只需要在本地以太坊节点查询而不需要在区块链上创建事物。

* 1. 支付与提现

以太坊是一个如此庸俗的东西甚至有些函数需要付费才能运行，这就是RMB玩家和普通玩家的区别。所以有些时候你的问题在于钱没冲够。如果我们想过滤掉穷B玩家那么就需要一个payable函数。



玩家充值后其以太坊会被冻结到合约的以太坊账户里，因此要设法将其提取出来。

* 1. 事件与日志

事件发生后被记录到链上成为日志。事件也可以用来实现交互功能。当定义的事件触发时，我们可以将事件存储到EVM的交易日志中，日志是区块链中的一种特殊数据结构。日志与合约关联，与合约的存储合并存入区块链中。只要某个区块可以访问，其相关的日志就可以访问。但在合约中，我们不能直接访问日志和事件数据（即便是创建日志的合约）。

1. ERC721与ERC20
2. 动手写一个智能合约
   1. 环境安装

首先安装最新版的nodejs和npm，过程略。其次安装truffle。

npm install -g truffle

truffle version

看到如下版本号说明安装成功。编辑器我用的Visual Studio Code。

https://steemit.com/cn/@lucia3/ethereum-pet-shop

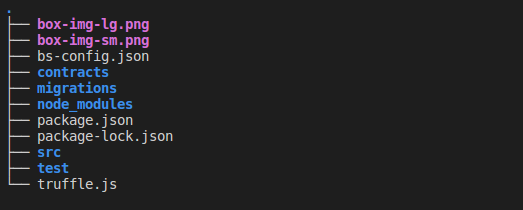
* 1. 使用Truffle Box创建一个Truffle项目

新建一个叫pet-shop的文件夹

cd pet-shop/

truffle unbox pet-shop

等待项目ubox完成。看一下项目目录（我用的vsCode）



contracts:智能合约

migrate:移植合约

src:和前端有关的资源

test:测试代码

truffle.js :truffle配置文件

* 1. 合约编写

合约功能是模拟领养宠物狗，如果你选择了领养，那么别人就不能再领养。一共16只，编号0-15.

在contracts下新建一个文件，文件名Adoption.sol。文件代码如下：

pragma solidity ^0.4.16;//指定版本

contract Adoption {

address[16] public adopters;//领养者的address

function adopt(uint petId) public returns (uint) {

require(petId >= 0 && petId <= 15);

adopters[petId] = msg.sender;//记录领养者address

return petId;

}

function getAdopters() public view returns (address[16]) {

return adopters;

}

}

* 1. 合约迁移

Truffle Develop是Truffle的内置开发平台，可以用来测试部署合同，还可以在上边执行truffle命令。

首先我们用truffle把solidity编译成EVM能读懂的东西。

在工程目录下执行以下指令：

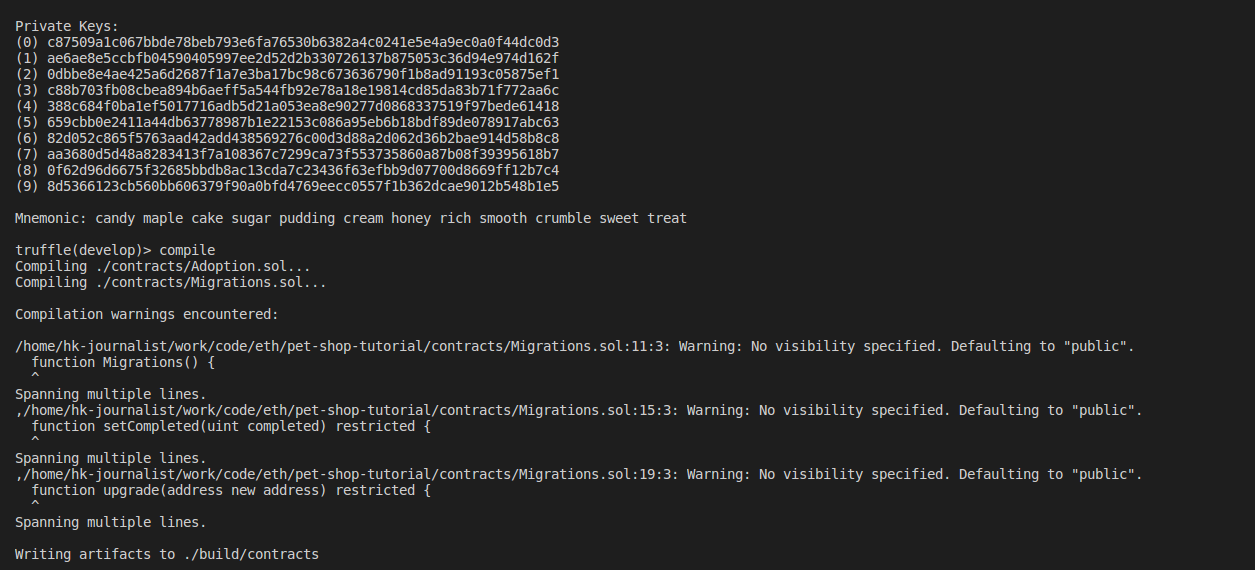
truffle develop

屏幕会出现truffle(develop)>：。恭喜你打开了新的世界。接下来的指令无特殊说明都在这个下面执行。

在控制台输入：

compile

系统将会进行编译，如果编译成功看到：



忽略那个警告。

下面把合约迁移到链上。

在migrations下新建一个2\_deploy\_contracts.js。内容如下：

var Adoption = artifacts.require("Adoption");

module.exports = function(deployer) {

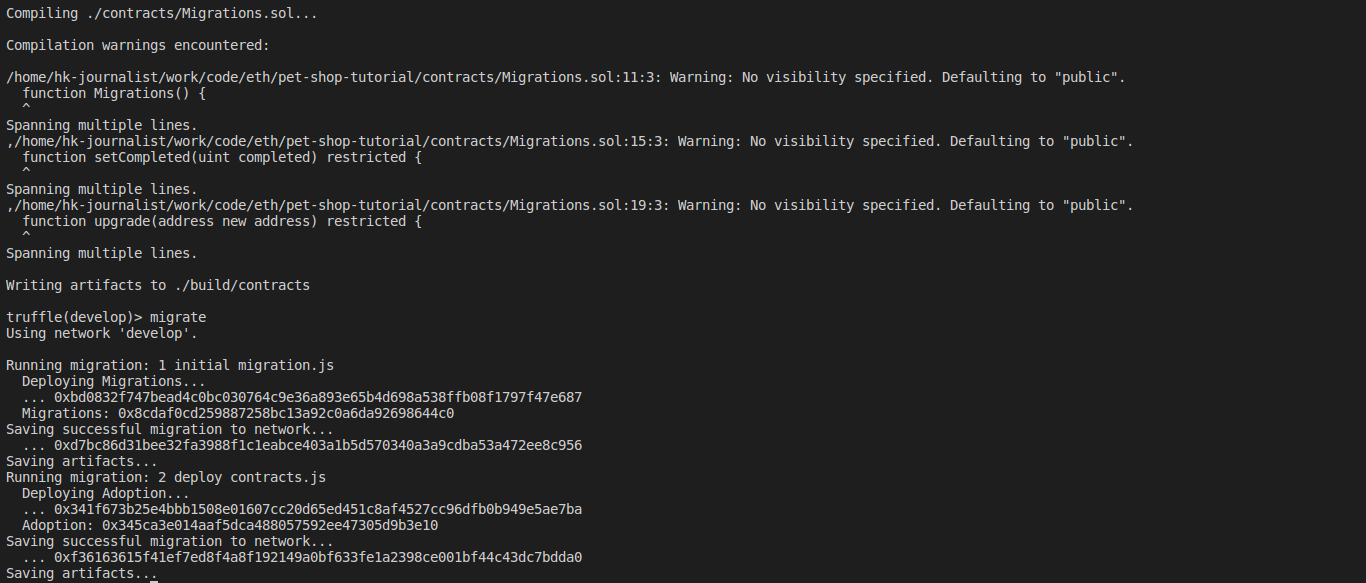
deployer.deploy(Adoption);

};

回到控制台，迁移一下，输入：

migrate

看到输出



* 1. 合约测试

在test下新建TestAdoption.sol。

pragma solidity ^0.4.16;

import "truffle/Assert.sol";

import "truffle/DeployedAddresses.sol";

import "../contracts/Adoption.sol";

contract TestAdoption {

Adoption adoption = Adoption(DeployedAddresses.Adoption());

function testUserCanAdoptPet() {

uint returnedId = adoption.adopt(8);

uint expected = 8;

Assert.equal(returnedId, expected, "Adoption of pet ID 8 should be recorded.");

}

function testGetAdopterAddressByPetIdInArray() {

address expected = this;

address[16] memory adopters = adoption.getAdopters();

Assert.equal(adopters[8], expected, "Owner of pet ID 8 should be recorded.");

}

}

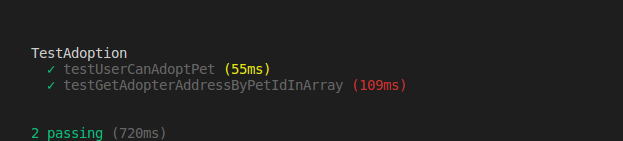
在import truffle的两个文件那里可能会提示找不到文件，不过没关系。这个东西是全局的，不是这个鼠目寸光的编译器能够理解得了的。

导入的两个truffle文件是全局的，虽然在VSCode中报错但是不用管他。

来到truffle终端输入

test

看到结果2 passing。



* 1. UI

仅仅有黑框框是不够的，现在需要一个交互页面，不过我们已经给你写好了。交互页面逻辑大概这样：

1

打开src/js/app.js做一些修改

添加一些函数体

* **initWeb3：**//初始化web3

if (typeof web3 !== 'undefined') {

App.web3Provider = web3.currentProvider;

} else {

App.web3Provider = new Web3.providers.HttpProvider('http://localhost:8545');

}

web3 = new Web3(App.web3Provider);

return App.initContract();

* **initContract：**//初始化智能合约

$.getJSON('Adoption.json', function(data) {

var AdoptionArtifact = data;

App.contracts.Adoption= TruffleContract(AdoptionArtifact);

App.contracts.Adoption.setProvider(App.web3Provider);

return App.markAdopted();

});

return App.bindEvents();

* **markAdopted：**//完成领养后更新一下UI

var adoptionInstance;

App.contracts.Adoption.deployed().then(function(instance) {

adoptionInstance = instance;

return adoptionInstance.getAdopters.call();

}).then(function(adopters) {

for (i = 0; i < adopters.length; i++) {

if(adopters[i]!== '0x0000000000000000000000000000000000000000') {

$('.panel-pet').eq(i).find('button').text('Success').attr('disabled', true);

}

}

}).catch(function(err) {

console.log(err.message);

});

* **handleAdopt：**//处理领养程序

event.preventDefault();

var petId = parseInt($(event.target).data('id'));

var adoptionInstance;

web3.eth.getAccounts(function(error, accounts) {

if (error) {

console.log(error);

}

var account = accounts[0];

App.contracts.Adoption.deployed().then(function(instance) {

adoptionInstance = instance;

return adoptionInstance.adopt(petId, {from: account});

}).then(function(result) {

return App.markAdopted();

}).catch(function(err) {

console.log(err.message);

});

});});

* 1. 在浏览器中创建DAPP

在chrome装一个MetaMask插件。装好后启动，接受协议，进入一个页面叫你输入new password和comfirm password。先不要输入。点击下边的import existing Den。在wallet seed框中输入：

candy maple cake sugar pudding cream honey rich smooth crumble sweet treat

点击ok。回到输入密码页面，输入密码和确认密码。点击ok。

点击Main Network，选Custom RPC。在NetWork框输入：

http:// localhost:9545 并save

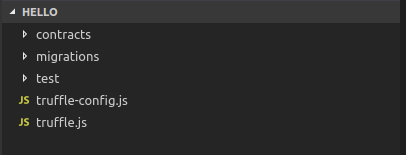
点击setting旁边的的左箭头，返回账户页面。

回到终端，不是truffle终端。输入 npm run dev。

将会自动开启一个网页，然后你就会看到……

1. 尝试篡改第九章的Demo

创建一个文件夹 truffle init



1. 创建过程中的一些坑
   1. **Assert找不到equal函数**

报错输出：TypeError: Member "equal" is not available in type(library Assert) outside of storage.

原因：你所测试的函数的返回值是string，但是目前为止solidity不支持合约之间返回string。因为人呐就不知道就不可以预料一个字符串有多长。所以改成byte32.

* 1. **error Invalid address**

Metatask没登录……

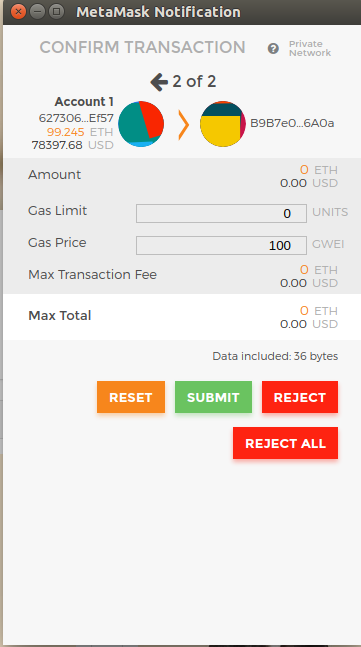
* 1. **Using network 'develop'.Network up to date.**

migrate –reset

* 1. **Attempting to run transaction which calls a contract function, but recipient address**

migrate –reset

* 1. **cannot connect to unknown private network**
  2. **base fee exceeds gas limit**



在gas limit里设置一下gas，推荐的最低值不一定够，要续一下。

* 1. **Contract has not been deployed to detected network (network/artifact mismatch)**

Migrations文件夹下的2\_deploy\_contracts.js文件存在问题。修复并重新编译。

* 1. **Uncaught Error: the tx doesn't have the correct nonce. account has nonce of: X tx has nonce of: Y**

1. 一些很难归类的问题

现在的web3不资瓷编译功能，那么就要直接找到abi。这个东西经过compile直接生成，并保存在json中。所以js读取这个json。

与以太坊的智能合约交互，除了使用web3.js，还可以使用另外一个Javascript库，就是[truffle-contract](https://github.com/trufflesuite/truffle-contract)。truffle-contract对以太坊的智能合约做了更好的抽象，相比于web3.js，使用truffle-contract操作智能合约更加方便。