Solidity 简介

2018.10

Solidity是什么

- Solidity 是一门面向合约的、为实现智能合约而创建的高级编程语言。这门语言受到了 C++，Python 和 Javascript 语言的影响，设计的目的是能在以太坊虚拟机（EVM）上运行。
- Solidity 是静态类型语言，支持继承、库和复杂的用户定义类型等特性。
- 内含的类型除了常见编程语言中的标准类型，还包括 address 等以太坊独有的类型，Solidity 源码文件通常以 .sol 作为扩展名
- 目前尝试 Solidity 编程的最好的方式是使用 Remix。Remix 是一个基于 Web 浏览器的 IDE，它可以让你编写 Solidity 智能合约，然后部署并运行该智能合约。

Solidity语言特性

Solidity的语法接近于JavaScript，是一种面向对象的语言。但作为一种真正意义上运行在网络上的去中心合约，它又有很多的不同：

- 以太坊底层基于帐户，而不是 UTXO，所以增加了一个特殊的 address 的数据类型用于定位用户和合约账户。
- 语言内嵌框架支持支付。提供了 payable 等关键字，可以在语言层面直接支持支付。
- 使用区块链进行数据存储。数据的每一个状态都可以永久存储，所以在使用时需要确定变量使用内存，还是区块链存储。
- 运行环境是在去中心化的网络上，所以需要强调合约或函数执行的调用的方式。
- 不同的异常机制。一旦出现异常，所有的执行都将会被回撤，这主要是为了保证合约执行的原子性，以避免中间状态出现的数据不一致。

Solidity源码和智能合约

- Solidity 源代码要成为可以运行在以太坊上的智能合约需要经历如下的步骤：

1. 用 Solidity 编写的智能合约源代码需要先使用编译器编译为字节码（Bytecode），编译过程中会同时产生智能合约的二进制接口规范（Application Binary Interface，简称为 ABI）；
2. 通过交易（Transaction）的方式将字节码部署到以太坊网络，每次成功部署都会产生一个新的智能合约账户；
3. 使用 Javascript 编写的 DApp 通常通过 web3.js + ABI去调用智能合约中的函数来实现数据的读取和修改。

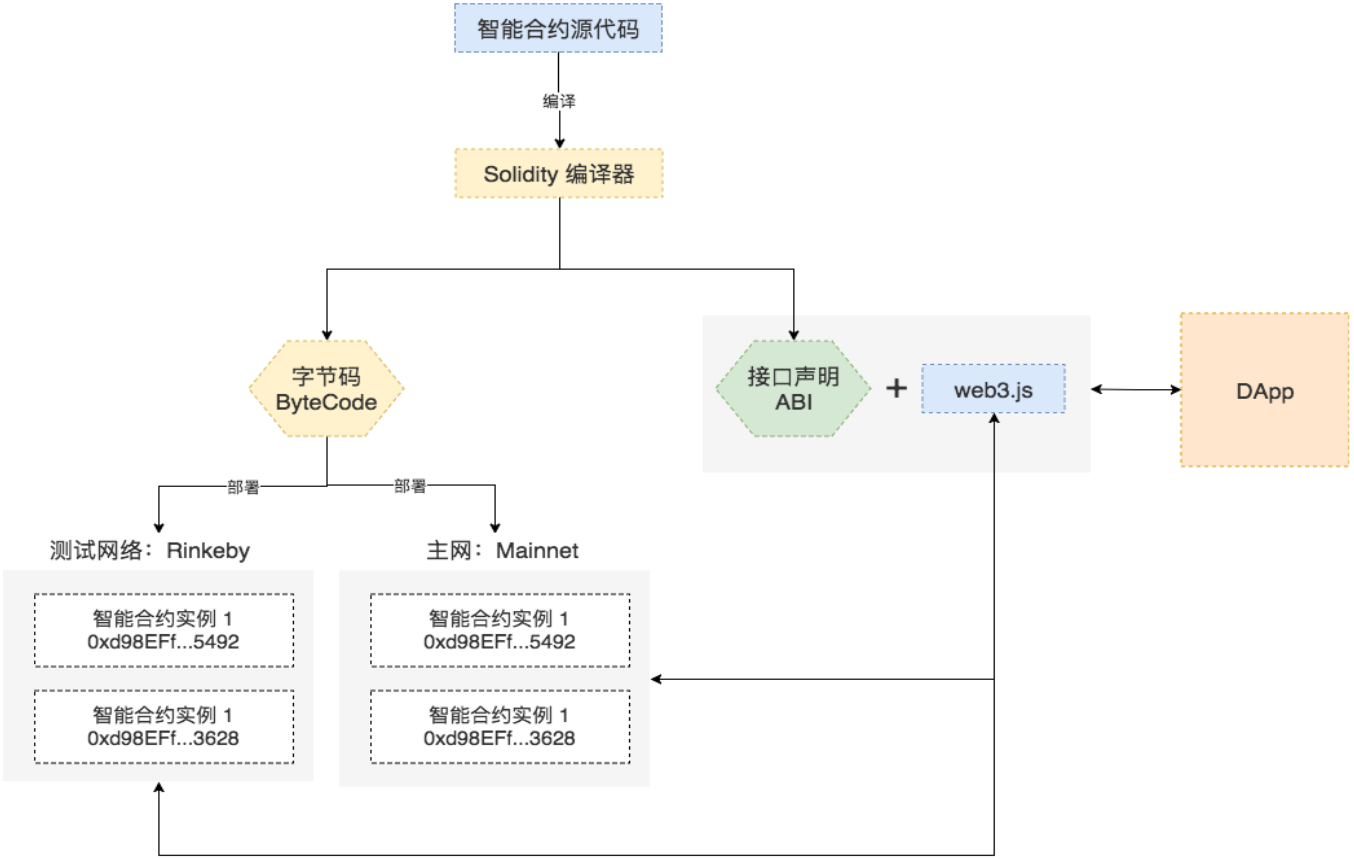
Solidity编译器

Remix

- Remix 是一个基于 Web 浏览器的 Solidity IDE；可在线使用而无需安装任何东西
- http://remix.ethereum.org

solcjs

- solc 是 Solidity 源码库的构建目标之一，它是 Solidity 的命令行编译器
- 使用 *npm* 可以便捷地安装 Solidity 编译器 solcjs
- *npm install -g solc*


智能合约源代码
编译
Solidity 编译器
字节码
ByteCode
接口声明
ABI
+
web3.js
DApp
部署
部署
测试网络：Rinkeby
主网：Mainnet
智能合约实例 1
0xd98EFf...5492
智能合约实例 1
0xd98EFf...3628
智能合约实例 1
0xd98EFf...5492
智能合约实例 1
0xd98EFf...3628

一个简单的智能合约

```
pragma solidity ^0.4.0;
contract SimpleStorage {
        uint storedData;
        function set(uint x) public {
                storedData = x;
        }
        function get() public view returns (uint) {
                return storedData;
        }
}
```

智能合约概述

Solidity中合约

- 一组代码（合约的*函数*)和数据（合约的*状态*），它们位于以太坊区块链的一个特定地址上
- 代码行 uint storedData; 声明一个类型为 uint (256位无符号整数）的状态变量，叫做 storedData
- 函数 set 和 get 可以用来变更或取出变量的值

合约结构

- 状态变量（State Variables）

  作为合约状态的一部分，值会永久保存在存储空间内。

- 函数（Functions）

  合约中可执行的代码块。

- 函数修饰器（Function Modifiers）

  用在函数声明中，用来补充修饰函数的语义。

- 事件（Events）

  非常方便的 EVM 日志工具接口。

智能合约练习

```
pragma solidity >0.4.22;
contract Car {
        string public brand;
        uint public price;
        constructor(string initBrand, uint initPrice){
                brand = initBrand;
                price = initPrice;
        };
        function setBrand(string newBrand) public {
                brand = newBrand;
        }
        function setPrice(uint newPrice)(uint) {
                price = newPrice;
        }
```

另一个例子 —— 子货币

```
pragma solidity >0.4.22  <0.6.0;
contract Coin {
        address public minter;
        mapping (address => uint) public balances;
        event Sent(address from, address to, uint amount);
        constructor() public { minter = msg.sender; }
        function mint(address receiver, uint amount) public {
                require(msg.sender == minter);
                balances[receiver] += amount;
        }
        function send(address receiver, uint amount) public {
                require(amount <= balances[msg.sender]);
                balances[msg.sender] -= amount;
                balances[receiver] += amount;
                emit Sent(msg.sender, receiver, amount);
```

合约代码解读

address **public** minter;

- 这一行声明了一个可以被公开访问的 address 类型的状态变量。
- 关键字 public 自动生成一个函数，允许你在这个合约之外访问这个状态变量的当前值。

mapping(address => uint) public balances;

- 也创建一个公共状态变量，但它是一个更复杂的数据类型，该类型将 *address* 映射为无符号整数。
- *mappings* 可以看作是一个**哈希表**，它会执行虚拟初始化，把所有可能存在的键都映射到一个字节表示为全零的值。

合约代码解读

event Sent(*address* from, *address* to, *uint* amount);

- 声明了一个“事件”（event），它会在 send 函数的最后一行触发
- 用户可以监听区块链上正在发送的事件，而不会花费太多成本。一旦它被发出，监听该事件的listener都将收到通知
- 所有的事件都包含了 from， to 和 amount 三个参数，可方便追踪事务

emit Sent(msg.sender, receiver, amount);

- 触发Sent事件，并将参数传入

事件的监听

```
Coin.Sent().watch({}, '', function(error, result) {
    if (!error) {
    console.log("Coin transfer: " + result.args.amount +
        "coins were sent from " + result.args.from +
        " to " + result.args.to + ".");
    console.log("Balances now:\n" +
        "Sender: " +
Coin.balances.call(result.args.from) +
        "Receiver: " +
Coin.balances.call(result.args.to));
```

```
pragma solidity >0.4.22  <0.6.0;
contract Coin {
        address public minter;
        mapping (address => uint) public balances;
        event Sent(address from, address to, uint amount);
        constructor() public { minter = msg.sender; }
        function mint(address receiver, uint amount) public {
                require(msg.sender == minter);
                balances[receiver] += amount;
        }
        function send(address receiver, uint amount) public {
                require(amount <= balances[msg.sender]);
                balances[msg.sender] -= amount;
                balances[receiver] += amount;
                emit Sent(msg.sender, receiver, amount);
        }
}
```

Ballot -- 一个简单的投票合约

- 电子投票的主要问题是如何将投票权分配给正确的人员以及如何防止被操纵。 这个合约展示了如何进行委托投票，同时，计票又是 **自动和完全透明的**
- 为每个（投票）表决创建一份合约，然后作为合约的创造者——即主席，将给予每个独立的地址以投票权
- 地址后面的人可以选择自己投票，或者委托给他们信任的人来投票
- 在投票时间结束时，winningProposal() 将返回获得最多投票的提案

Q&A


尚硅谷