4.4 智能合约Solidity编程初探

01 Solidity概述

Solidity是面向合约的高级语言，它的语法与JavaScript类似，被设计运行于以太坊虚拟机上(EVM)。

Solidity是静态类型的，支持继承、库和复杂的用户自定义类型。

Solidity与其它语言相比有许多的不同，下面列举论证：

•**以太坊底层是基于帐户**，而非UTXO的，所以有一个特殊的Address类型。

•由于语言内嵌框架是**支持支付的**，所以Solidity语言提供了一些关键字，如payable，可以在语言层面直接进行支付。

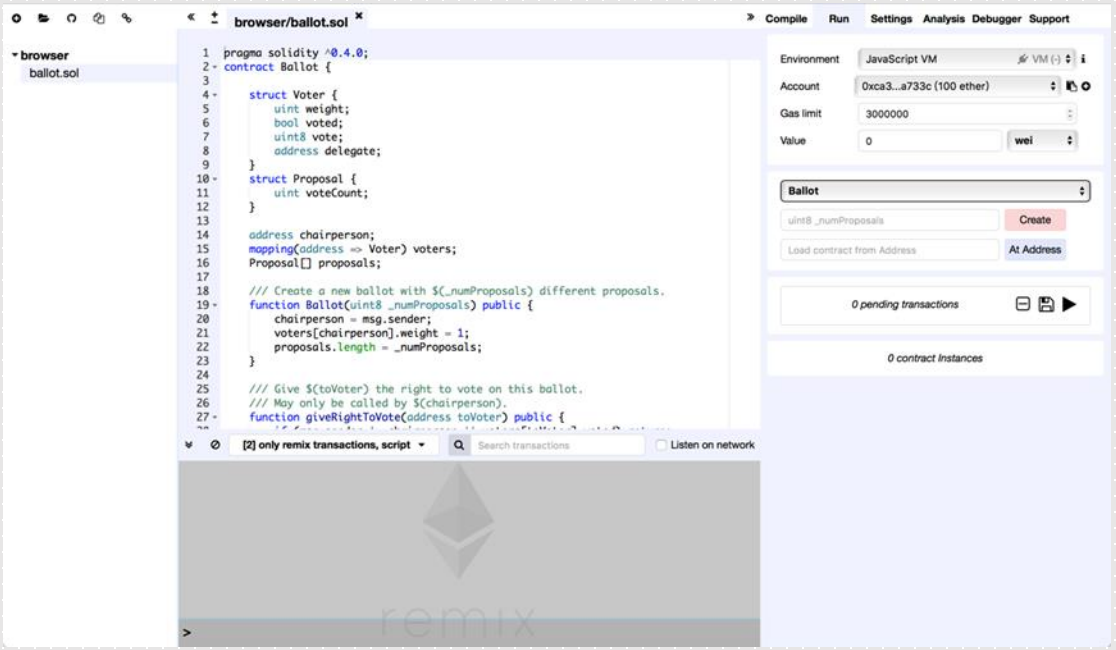
•**存储是使用网络上的区块链。**数据的每一个状态都可以永久存储，所以需要确定变量使用内存，还是使用区块链。

•**运行环境是在去中心化的网络上**，会比较强调合约或函数执行调用的方式。

•最后一个非常大的不同则是它的**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。

02 Solidity 概述

开始学习Solidity最好的方式是使用Remix（https://remix.ethereum.org）。Remix是基于浏览器的IDE，可以直接在浏览器上打开使用，它集成了Solidity编译和运行时的环境，不需要服务端组件。下图就是Remix打开的界面：



03 Solidity源代码文件结构

- 源文件可以包含任意数量合约的定义，包括指令和编译指令。以下是一个示例，定义了一个简单的存储合约:

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;
    function set(uint x) {
        storedData = x;
    }

    function get() constant returns (uint) {
        return storedData;
    }

}
```

04 Solidity源代码文件结构

1.版本申明

源文件的第一行需要进行版本申明，版本申明的格式为:

```
pragma solidity ^0.4.0;
```

进行这样版本申明的源文件表示它不能被0.4.0版本之前的编译器编译，也不会被0.5.0及之后的版本编译，也就是说需要高于0.4.0并且低于0.5.0的版本才可以进行编译。

05 Solidity源代码文件结构

2.引用其他源文件

全局引入。从“filename”中导入所有全局符号到当前源文件的全局范围:

```
import “filename”;
```

自定义命名空间引入。创建一个新的全局符号symbolName，它的成员均是从“filename”文件引

```
import * as symbolName from “filename”
```

分别定义引入。创建新的全局符号alias和symbol2，它们分别从”filename”中引用symbol1和symbol2:

```
import {symbol1 as alias, symbol2} from “filename”
```

06 Solidity源代码文件结构

3.路径相关

路径名**以.开头代表当前目录**，**以..开头代表父目录**，否则代表绝对路径。因此为了引入当前目录下的文件，使用 import "./x" as x;。如果使用 import "x" as x;，则可能会引入一个不同的文件(在一个全局的include目录下)。

为什么会有这个区别呢?

解析路径的时候，目录层级结构并不与我们本地的文件一一对应，它非常有可能是通过ipfs、http或git建立的一个网络上的虚拟目录。

07 Solidity源代码文件结构

4.在实际的编译器中使用

编译器引用文件的机制包括：

- 可以将一个域名下的文件映射到本地，从而从本地的某个文件中读取；
- 提供对同一实现的不同版本的支持（可能某版本的实现前后不兼容，需要区分）；
- 如果前缀相同，取最长；
- 有一个”fallback-remapping”机制，空串会映射到“/usr/local/include/solidify”；

08 Solidity源代码文件结构

5.代码注释

单行注释使用（//）,多行注释使用(/*...*/)。

6.文档注释

文档注释用于书写文档，使用三个斜杠///或者/**...*/。可以使用Doxygen语法，来生成对文档的说明、参数注解或者是当用户调用某个函数时，弹出来的确认内容。

09 合约的结构

使用Solidity编写的合约类似于面向对象语言中的类，并且支持继承其他的合约。

每个合约中可包含**状态变量**(State Variables)、**函数**(Functions)、**函数修饰符**（Function Modifiers）、**事件**（Events）、**结构类型**(Structs Types)和**枚举类型**(Enum Types)。

10 合约的结构

1.状态变量

状态变量的变量值会永久存储于智能合约的存储空间中，示例如下：

```
pragma solidity ^0.4.0;

contract simpleStorage{

    uint valueStore; //状态变量

}
```

11 合约的结构

2.函数

函数是智能合约中的一个可执行单元，示例如下：

```
pragma solidity ^0.4.0;

contract SimpleAuction {

    function bid() payable { // 函数

        // ...
    }
}
```

函数调用可以设置为内部（Internal）的和外部（External）的。同时可以设置对于其它合约不同级别的可见性和访问控制(Visibility and Accessors)。

12 合约的结构

3.函数修饰符

函数的修饰符用于在函数声明时修改函数的语义。示例如下:

如果函数abort()使用修饰符“onlySeller”，则该函数会首先使用“require(msg.sender == seller);”语句；如果执行通过，则继续执行onlySeller函数体里面的内容。

```
pragma solidity ^0.4.11;

contract Purchase {
    address public seller;

    modifier onlySeller() { // 修饰符
        require(msg.sender == seller);
        _;
    }

    function abort() onlySeller { // 使用修饰符
        // ...
    }
}
```

13 合约的结构

4.事件

事件是以太坊虚拟机(EVM)日志基础设施提供的一个便利接口。下面是一个简单的示例:

```
pragma solidity ^0.4.0;

contract SimpleAuction {
    //事件
    event HighestBidIncreased(address bidder, uint amount);

    function bid() payable {
        // ...
        //触发事件
        HighestBidIncreased(msg.sender, msg.value);
    }
}
```

14 合约的结构

5.结构体类型

结构体是将一些变量组合在一起的自定义类型。以下是一个简单的结构体示例：

```
pragma solidity ^0.4.0;

contract Ballot {

    struct Voter { // 结构体

        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }
}
```

15 合约的结构

6.枚举类型

枚举可以用来创建包含一组有限值的自定义类型。以下是一个简单的结构体示例：

```
pragma solidity ^0.4.0;

contract Purchase {
    enum State { Created, Locked, Inactive } // 枚举
}
```