solidity文档链接：<https://solidity.readthedocs.io/en/v0.4.25/>

[remix-ide在线链接:](https://solidity.readthedocs.io/en/v0.4.25/) <http://remix.ethereum.org/#optimize=false>

**solidity**

**solidity in depth**

**layout of a solidity source file**

**version pragma**

pragma solidity ^0.4.0 //不能用低于0.4.0的版本编译,也不能用高于0.4.0的版本编译,版本的范围为 [ 0.4.0, 0.5.0 )

**importing other source files**

导入文件:

import "./demoToken.sol"; //从文件中导入全局符号.

import \* as symbolName from "./demoToken.sol"; //创建一个全局变量symbolName,它的成员是文件demoToken.sol中的所有全局变量. 这个方法与 import "demoToken.sol" as symbolName; 是一样的.

import {symbol1 as alias, symbol2} from "filename"; //从文件中导入symbol1别名为alias,导入别名symbol2.

在导入的时候,除非路径以 . 或者 .. 开始,否则,所有路径都被视为是绝对路径.

例子1:

import "./x" as x; //从当前路径导入文件.

import "x" as x; //从系统的标准include路径导入文件.

solc编译智能合约的时候可以在参数中替换import的前缀, 例如:

可以在源代码中写入 import "github.com/ehereum/dapp-bin/library/iterable\_mapping.sol" as it\_mapping; 在运行编译器的时候可以进行前缀替换以找到本地系统的文件,如下: solc github.com/ethereum/dapp-bin = /usr/local/dapp-in/ source.sol

**comments**

solidity支持三种注释方式:

单行注释 //

多行注释 /\* ... \*/

doxygen样式的注释,能够自动生成说明文档.

**struct of a contract**

状态变量持久的存储在"合约存储"(区块链)中.

**units and globally available variables**

**special variables and functions**

block and transaction properities //块及交易属性

block.number //块的数量,如果当前最新块号是12,那么block.number返回13.

block.blockhash(uint blockNumber) returns(bytes32); 用于小于0.4.22版本的编译器.

blockhash(uint blockNumber) returns(bytes32); 用于大于等于0.4.22版本的编译器.

blockhash只能访问最新的256个块的hash,早于256个块的hash就不能访问到了.

block.coinbase 返回address.

block.gaslimit 返回uint.

block.difficulty 返回uint.

block.timestamp 返回uint.

msg.sender 返回消息发送者的地址.

msg.value 消息要转的钱,合约函数调用的话此域一般为0.

now 返回uint,当前块(准备出的块)的时间戳,是block.timestamp的别名.

tx.gasprice 返回uint, 交易的gasprice.

tx.origin 返回address, 交易的最早发起方(full call chain).(跨合约调用应该为用户账号.)

msg.data 返回消息的calldata字段.

msg.sig 返回消息的calldata字段的前4个字节.

gasleft() 返回uint256, 当前块(准备要出的块)当前剩余的gas数量.

error handing

assert(bool condition); 用于处理合约内部错误, condition为false则返回.

require(bool condition); 如果条件不满足,那么交易回退.

require(bool condition, string message); 条件不满足则交易回退,打印错误信息.

revert() 条件不满足则中断执行,交易回退.

revert(string reaseon); 条件不满足则中断交易,交易状态回退.

address releated

<address>.balance; 查询address的余额.

<address>.transfer(uint256 amount); 给address转amount个wei,失败抛出异常(这个异常会导致合约执行的终止). 如果地址是一个合约地址,那么合约的code也会被执行,这个是由evm虚拟机决定的,不能被阻止.

<address>.send(uint256 amount) returns(bool); 给address转amount个wei,失败返回false.(send是比transfer低级的函数,send失败会返回false,合约不会被终止.)

<address>.call(...) returns (bool); 是一个底层函数,用来向一个合约发送消息,可以带有任意类型的参数. 以下例子演示了如何通过call方法调用合约的函数以及如何向合约传递参数.

合约1:

*pragma solidity ^0.4.0;*

*contract Test1 {*

*uint256 a;*

*function setT1(uint256 num) public {*

*a += num;*

*}*

*function getT1() public view returns(uint256){*

*return a;*

*}*

*}*

*<address>.callcode(...) return (bool);*

*<address>.delegatecall(...) returns (bool);*

合约2:

*pragma solidity ^0.4.0;*

*contract Test2 {*

*function getNum(address addr ) public returns(bool){*

*//call的第一个参数为要执行的函数的签名.后面为函数的参数.*

*//如果第一个参数为bytes4,那么会认为是一个函数签名.然后evm找对应函数去执行.*

*return addr.call(bytes4(keccak256("setT1(uint256)")),333333);*

*}*

*}*

**contracts**

**visibility and getters**

**继承**

solidity支持多继承.当一个合约继承了多个合约的时候,在部署的时候只在链上部署了一个合约,基合约的代码被拷贝到创建的合约中(即以代码拷贝的方式完成了继承).

子合约能够访问到基合约的所有非私有成员,包括internal函数和状态变量.

抽象合约(有的函数只有声明没有实现)是为了让编译器知道接口,这样在跨合约调用的时候才能编译通过.

如果子合约没有指定基合约的全部参数,那么这个子合约被认为是一个抽象合约(不能被实例化).

下面的例子是指定基合约的两种方式:

testBase.sol合约:

*pragma solidity ^0.4.11;*

*contract TestBase {*

*uint256 a;*

*string b;*

*constructor(uint256 \_a, string \_b) public {*

*a = \_a;*

*b = \_b;*

*}*

*function getValues() public view returns(uint256, string) {*

*return (a,b);*

*}*

*}*

testA.sol合约:

*pragma solidity ^0.4.11;*

*import "./testBase.sol";*

*//第一种给基类构造函数赋值的方式.*

*contract TestA is TestBase {*

*constructor(uint256 \_a, string \_b) TestBase(\_a, \_b) public {*

*}*

*}*

*//第二种给基类构造函数赋值的方式.*

*contract TestB is TestBase(4,"aaaa") {*

*}*

**abstract contracts 抽象合约**

当合约中有一个函数只有声明没有实现的时候，这个合约就被称为抽象合约。

抽象合约因为包含未实现的函数所以不能被编译，但是抽象合约可以被当做基类使用。

如果继承了抽象基类的子类没有实现所有抽象函数，那么这个子类也是抽象的。

函数声明和函数类型的区别：

function foo(address) external returns(address); //函数声明

function(address) external returns(address) foo; //foo为函数类型

**interface 接口**

接口与抽象合约类似，但是接口中的所有函数都不能有实现；抽象合约中只需要有一个函数只有声明没有实现即可。换句话说，接口就是特殊的抽象函数。

接口必须满足如下要求：

不能继承其他合约或接口。

不能定义构造函数。

不能定义变量，结构和枚举。

合约也能继承自接口，就行他们继承自其他合约一样。

在跨合约调用的时候,比如A合约调用B合约,这种情况可以在合约A中用抽象合约或者接口声明B的合约原型(不需要声明全部函数,只需要声明要使用到的函数即可).

**验证**

1. 抽象合约能够用于跨合约调用，写代码验证。

2. 接口也能够用于跨合约调用，写代码验证。

3. import的使用.