**solidity文档学习笔记**

本学习笔记结构依照登链社区solidity中文文档编写，以自己在学习过程中加深印象以及方便查找为目的。

**day1**

**合约基本结构**

//SPDX-Licenxe-Identifier: GPL-3.0 (机器可读许可证)

Prama solidity >=0.8.0 (运行代码所需要的编译器版本要求)

Contract Coin{

constructor(){} (构造函数，在代码部署时执行)

//内部添加函数逻辑

}

实践代码

//SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.4;

contract Coin {

address public minter;//一个公开访问的状态变量，且不允许任何算术操作

public关键字会自动生产函数用于访问状态变量的当前值，即我们在访问 状态变量的时候，其实是调用了编译器生成的函数

function minter() external view returns (address) { return minter; }

mapping (address => uint256) public balances;

一个地址到无符号整数的映射，但是不能获得映射的键与值,只能通过地址指针来发现值

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

event表示一个事件，在其所在函数被触发之后，你就能获取监听信息，方便追踪交易。

constructor(){

minter = msg.sender;

}

构造函数constructor是只能在创建合约初期运行的函数

function mint(address reciver, uint amount)public{

require(msg.sender == minter);

balances[reciver] += amount;

}

error insufficientbalance(uint requested,uint available);

error用于向调用者反馈错误信息

function sender(address reciver,uint amount)public {

if(amount > balances[msg.sender])

revert insufficientbalance({

requested:amount,

available:balances[msg.sender]

});

balances[msg.sender] -= amount;

balances[reciver] += amount;

emit Sent(msg.sender,reciver,amount);//事件触发

}

**evm基础知识**

以太坊虚拟机evm是智能合约的运行环境，它是沙盒封装的也是完全隔离的，evm在运行代码时无法访问网络，文件以及其他进程

**账户**

以太坊中有两类账户，外部账户和合约账户

外部账户由公钥私钥对控制

合约账户由账户所存储的代码控制

两类账户类型都是相同的

**交易**

即一个账户向另一个账户发送的消息，包含二进制数据和以太币

**Gas**

每笔交易完成时所收取的费用，由原始交易发起人支付。当evm执行交易时，gas按特定规则消耗，当gas消耗完毕后，当前调用帧所做的所有状态操作都将会回滚到最近的未执行的状态。

**存储，内存，栈**

存储是账户上的一块持久化的内存区，读写效率低下，而且修改开销很高。

内存是一块临时的存储空间，合约会为每一次调用获取一块重新擦拭干净的内存，读长度限制256位，写长度可以为8位和256位

栈是用于合约执行时存储各种临时变量状态的空间，最大容纳1024个元素，每个元素最大长度为256位。

**合约自毁**

合约代码从区块链上移除的唯一方法是在合约中执行自毁操作，即将合约账户中的以太币发送到指定目标，然后将所有存储状态和代码移除。但是有丢失以太币的风险（指定目标位失效账户）

**day2**

**函数修改器modifier**

用于以声明的方式来修改函数语义，

Modifier onlyseller (){

require(){

}

\_;

}

**事件**

一个方便日志调用的接口

Event eventname(参数)；

触发用法即在需要监听的函数或其他操作中使用

Emit eventname(参数)；

**错误error**

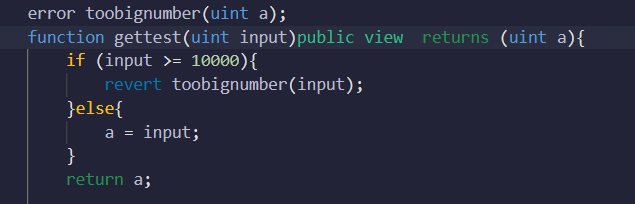
用于用户自定义错误的名称和数据

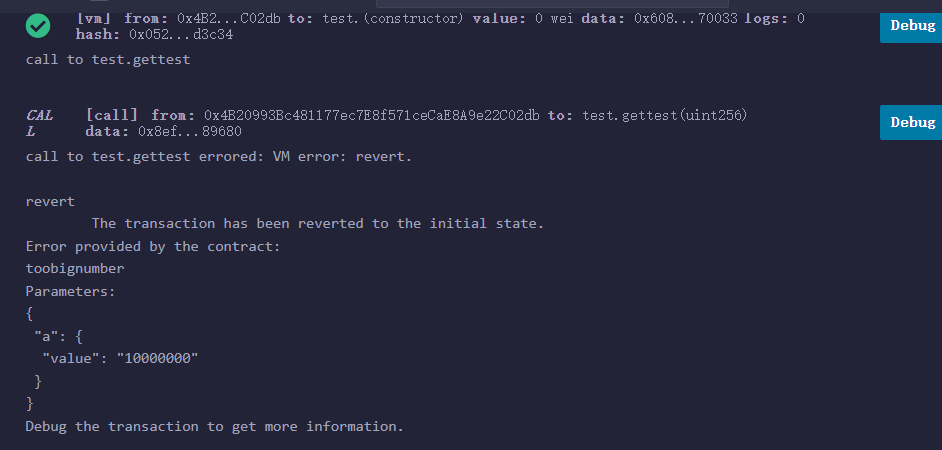
error errorname(参数);

触发时使用revert语句

Revert errorname(参数)

简单的实践代码





**结构体**

类似于java中的抽象类

struct structname{

Xxx

Xxx

xxx

}

**枚举型enum**

enum state{xxx,xxx,xxx}

**类型与逻辑**

solidity是一种静态类型的语言，每个变量都要在编译时指定变量的类型

在solidity中 null与 undefined并不存在，但是变量被声明时总有一个默认值，且默认值总与数据类型有关

**值类型**

以下类型称为值类型，因为这些变量始终按照值来转递，就是说这些变量在函数中作为参数或者用于赋值时，会进行值的复制

**布尔型**

bool 取值为true 或者false

相关的逻辑运算符

！逻辑非

&&逻辑与

||逻辑或

==相等

！=不等

运算符 || 和 && 都遵循同样的短路（ short-circuiting ）规则。就是说在表达式 f(x) || g(y) 中， 如果 f(x) 的值为 true ，那么 g(y) 就不会被执行，即使会出现一些副作用。

**整型**

整型分为有符号整型和无符号整型 int/uint

相关运算符 ：

比较>< <= >= == !=

位运算 &与 |或 ^异或 ~取反

移位 << 左移 >>右移

算术运算 +- \*/ \*\*幂 %取余或者模，幂运算只支持无符号整数

**定长浮点**

fixed/unfixed可以表示各种大小的有符号和无符号定长浮点

在关键字unfixedMxN与fixedMxN中，M表示该类型所占用的位数，M需要被8整除，8-256.N表示可用的小数位数，0-80的某一个数

**地址类型**

address 保存一个20字节的值

address payable 表示可支付地址，但是有成员函数transfer和send

**地址型成员变量**

Balance 和transfer

可以使用 balance 属性来查询一个地址的余额， 也可以使用 transfer 函数向一个可支付地址（payable address）发送 以太币Ether （以 wei 为单位）

send成员

Send是transfer的低级版本，执行失败之后，合约不会因为异常而终止，返回返回false

code和codehash成员

可以通过 .code来访问合约字节码

**定长字节数组**

byte1 byte2 ……byte32

变长字节数组

Bytes

String

他们都不是值类型

**day3**

映射

映射类型的声明结构是：mapping(keytype => valuetype)

keytype是除了数组类型其他的都可以

valuetype是所有类型都可以