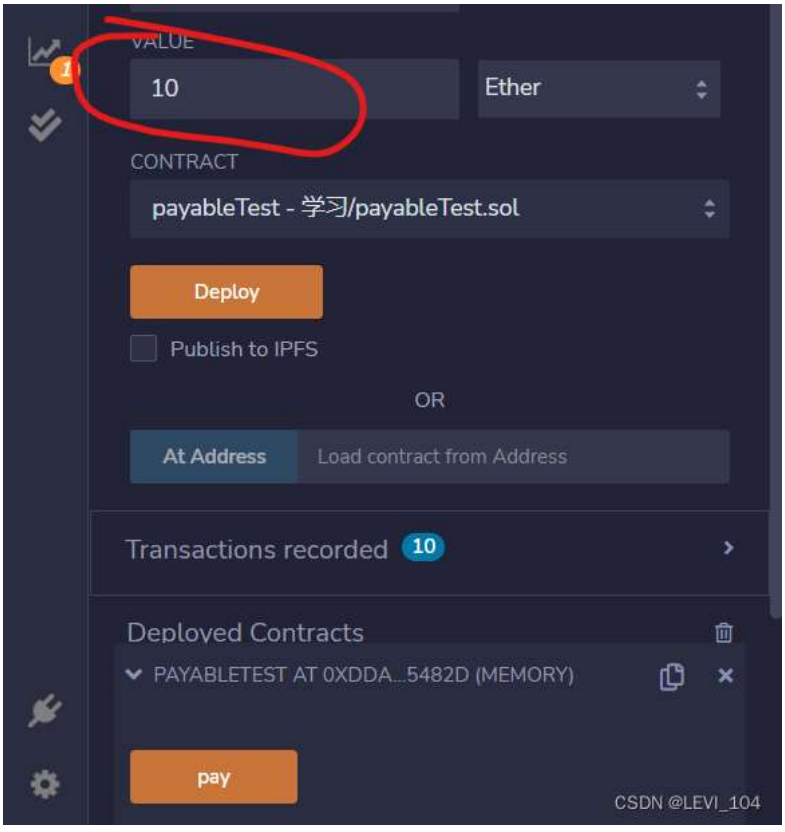
solidity自学：第三天

目录

使用钱包转移资金



当调用pay函数的时候(pay函数必须实现payable)，会向该合约地址发送10个以太币。

用户从99 ehter变成89 ether。

payable： 代表我们可以通过这个函数来给我们的合约地址充值、转账

```
    function pay() payable{
    }
    function getBalance() returns(
        return this.balance;//获取
    }
```

合约与合约账户

```
    //账户地址：0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
    //合约地址：0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
    function getThis() view returns(address){
        return this;
        //返回0xDA0bab807633f07f013f94DD0E6A4F96F8742B53
        //这说明了this表示本合约的地址
        //balanace是this里面的一个属性
    }

    function getRandomBalance() view returns(uint){
        //获得任意一个地址（账户/合约）的金额
        address account = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
        return account.balance;
    }
```

transfer转移资金

```
    //外部地址与外部地址之间的转账
// msg.value是用户，account1.transfer是合约地址。
    function transfer() payable{
        //向合约account1转入一笔钱
        address account1 = 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB;
        //意思是目前的合约会向account1转账：将VALUE设置为10，然后调用transfer
        //本合约：0x5B38Da6a701c568545dCfcB03FcB875f56beddC4从89变为79
        //account1:0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB从99变成109
        account1.transfer(msg.value);
    }
```

```
    function transfer2() payable{
        this.transfer(msg.value);
    }
    function () payable{
        //回滚函数
    }
```

以太坊中的全局属性

msg对象代表调用合约时传递的消息内容。

- msg.data (bytes):完整的calldata。完整的调用数据(calldata)
- msg.gas (uint):剩余的gas量
- msg.sender (address):消息的发送方(合约的调用者，也就是用户)
- msg.sig (bytes4):calldata的前四个字节(即函数标识符)
- msg.value (uint):联盟链中无需使用此数据。这个信息所附带的以太币，单位wei
- block.difficulty(uint):当前块的难度
- block.number(uint):当前块的块号
- block.coinbase(address):当前块矿工的地址(也就是这个块是谁挖出来的)
- now(uint):当前块的时间戳(block.timestamp的别名).

```
    function getGlobal1() view returns(address){
        return msg.sender;
        //返回0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
    }
    function getGlobal2() view returns(uint){
        return block.difficulty;
        //返回69762765929000
    }
    function getGlobal3() view
```

```
        return block.number;11        //返回15。该区块不会变化
    }
```

转账误操作

如果函数实现了payable，但是里面没有指定地址，那么VALUE的默认地址就是本合约的地址

```
function transfer() payable{}
```

奇怪的现象：VALUE设置为20，而transfer()设置为10。运行之后，目标用户地址+10，用户-20，而少了的10则去到了合约地址当中

```
    function transfer() payable{
        //account是目标用户地址
        address account = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
        account.transfer(10 ether);
    }
```

底层sender方法

transfer方法，如果什么都不传递进去，那么会报错。sender方法，什么都不传递进去也都可以运行，但是不会成功转账，它是一个底层函数，存在sender的返回值是bool。

send()方法执行时有一些风险：

- 调用递归深度不能超过1024
- 如果gas不够，执行会失败
- 所以使用这个方法要检查成功与否
- transfer相对send较安全，所以以后用transfer更好

mapping映射——哈希表

```
pragma solidity ^0.4.4;

contract mappingTest{
    //是一对一 一一对应的关系
    //                  id
    mapping(address => uint) idMapping;
    //本用户：0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 => 1
    mapping(uint => string) nameMapping;
    //1 => "陈钦"

    uint sum = 0;//用于编号

    function register(string name){
        address account_ = msg.sender;
        sum++;
        //mapping(address => uint) idMapping:account_会加入到address,sum加入到uint
        idMapping[account_] = sum;
        //mapping(uint => string) nameMapping:sum会加入到uint，name会加入到string
        nameMapping[sum] = name;
    }
    function getIdByAddress(address x) view returns(uint){
        return idMapping[x];
    }
    function getNameById(uint x) view returns(string){
        return nameMapping[x];
    }
}
```

函数重载

以下代码test1(123)发生了错误，因为address和uint160在solidity处理的时候都是一样的，按照uint160处理。无法匹配到对应的test1.两个函数虽然可过，但是无法调用

```
function test1(address _x){}
function test1(uint160 _x){}
function test(){
    test1(123);
}
```

函数命名参数

参数的传入可以不按照顺序，用下面的方法来写

```
function test(){
    setGuy({_name:"123",_phone:6164665});
}
```

返回值

风骚的返回方式

```
function returntest() returns(uint mul){
    mul = 1;
    //return 5;  如果此时有这句话，则返回5，否则返回1
}
```

多返回值

```
function t(uint a,uint b) view returns(uint add,uint mul){
    add = a + b;
    mul = a * b;
    //return(a+b,a*b);  是同样的效果
}
```

constant

在4.0版本中和view等价，5.0版本中被废弃。局部变量中无法使用constant

uint public constant num = 50; =====> 全局变量无法被修改

4.0版本中，支持int, uint, string ,bytes1…bytes32使用constant

构造函数

构造函数只能有一个（和Java不一样！！！）

```
function constructor_(){
    a = 100;
```

```
    }
    //新版本的solidity的构造器如下写法：
    constructor(){//默认是public
        a = 100;
    }
```

modifier

相当于一个判断条件，将判断封装起来

```
contract modifier_{
    address public owner;
    uint public num = 0;

    constructor(){
        owner = msg.sender;
    }

    modifier OnlyOwner{
        require(msg.sender == owner);
        _;
    }

    function changeIt(uint _num) OnlyOwner{
        num = _num;
    }
}
```

```
    uint public level = 0;
    modifier level(uint needLevel){
        require(level >= needLevel);
        _;
    }
    function changeId() level(2){}
    function changeName() level(10){}
```

多重modifier

```
    modifier mod1{
        a = 1;
        _;
        a = 2;
    }
    modifier mod2{
        a = 3;
        _;
        a = 4;
    }
    function m() mod1 mod2{
        a = 100
    }//执行结果：a = 2
    //过程：a = 1,a = 3, a = 100 ,a = 4, a = 2
```

继承inherit

格式：contract A is B{}　　A是孩子，B是双亲

public, interal, external, private中，只有private修饰的不可以被继承

- external：只可以在外部调用，可以被继承
- internal：只可以在内部调用，可以被继承
- private：只可以在内部调用，不可以被继承

> 特殊的，external可以这么调用

```
    function dahan() external pure returns(string){}
    function dahanTest() public view{
        this.dahan();//这里相当于外部调用
    }
```

或者在外部合约中new一个内部合约，然后调用该方法

多重继承

contract son is dad, mom{}

这个合约是覆盖继承，如果dad和mom有重复的属性或者方法，mom作为后者会覆盖dad前者，在合约son中显示mom的属性或者方法

全局变量自动getter函数

public会自动生成getter函数

比如：uint public num = 1;

此时相当于写了下面这个函数：

function num() external view returns(uint){ return num;}

比如：mapping(uint => string) public map;

此时相当于写了下面这个函数：

function map(uint key) external returns(string){}

疯狂的映射：这里在mapping方法中输入0,3,20，则输出陈钦

```
    mapping(uint => mapping(uint => mapping(uint => string))) public map;
    function test(){
        map[0][3][20] = "陈钦";
    }
```

合约的销毁

使用selfdestruct()来销毁

```
pragma solidity ^0.4.0;

contract destruction{
    address owner;
    uint public money = 0;

    constructor(){
        owner = msg.sender;
    }

    function increment(){
        money += 10;
    }
    function kill(){
        if(msg.sender == owner){
            selfdestruct(owner);
        }
    }
}
```