

DAY3: SOLIDITY

登链社区 - Tiny熊

练习题

- 安装 Metamask、并创建好账号（向水龙头获取代币）
- 向同桌的地址执行一次转账
- 使用 Remix 创建一个Counter合约并部署到测试网:
 - Counter 合约添加一个 add(x) 方法;

作业要求：

1. 使用自己的 github 创建  firstcontract
2. 提交代码、截图、交易 hash 等

习题解答

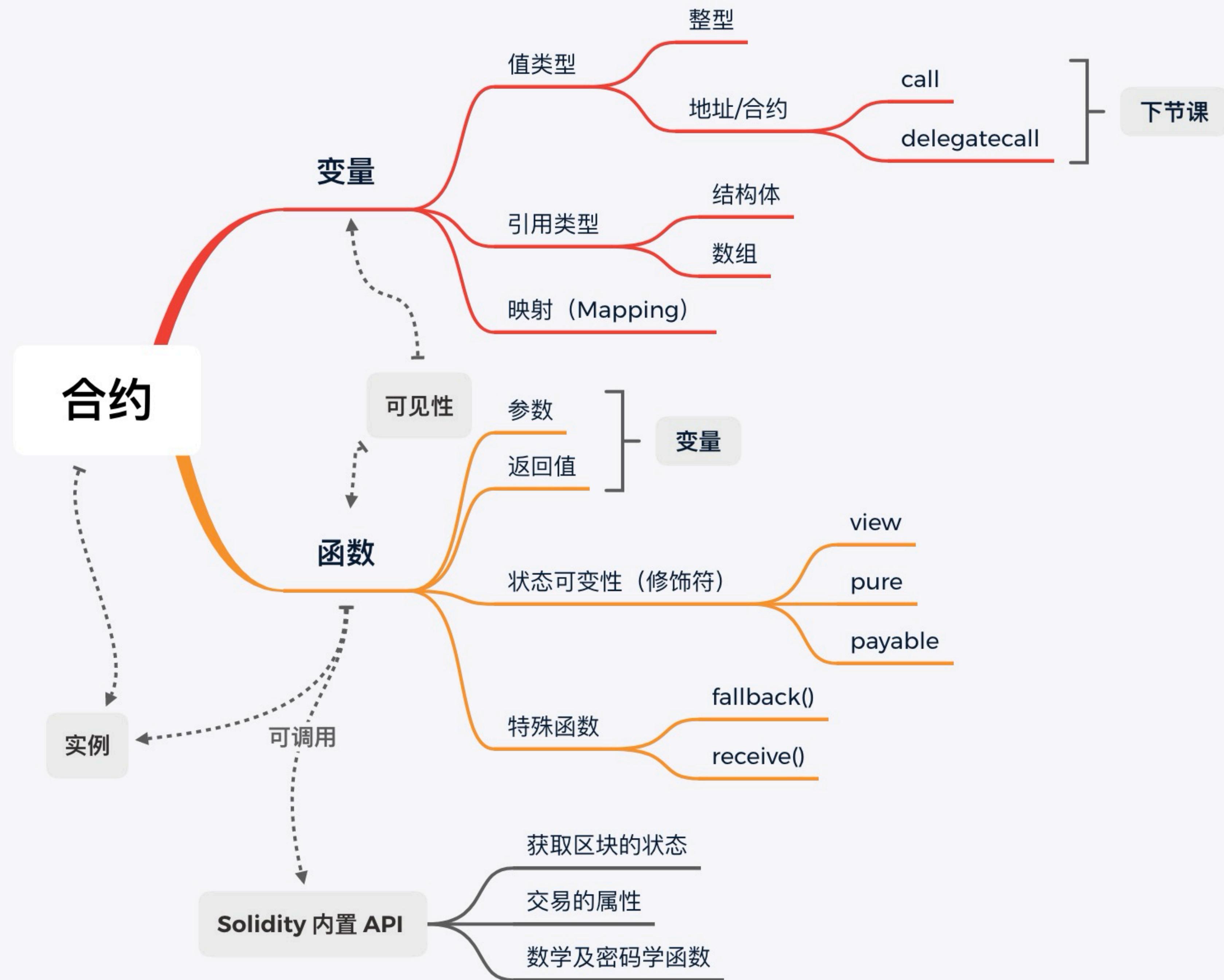
- 账号获取代币（水龙头或购买）
 - <https://chaintool.tech/faucet#Sepolia>
- 使用 Remix 创建一个 Counter 合约并部署：
 - Counter 合约有一个 add(x) 方法；

SOLIDITY 语言

- 静态类型、编译型、高级语言
- 针对 EVM 专门设计
- 受 C++、JavaScript 等语言影响
 - 如：变量声明、for 循环、重载函数的概念 来自于 C++
 - 函数关键字、导入语法来自于 JavaScript
- 文档
 - 中文：<https://learnblockchain.cn/docs/solidity/>
 - 英文：<https://docs.soliditylang.org/>
 - Decert: <https://decert.me/tutorial/solidity/intro/>

合约的组成

```
pragma solidity ^0.8.0; 1. 编译器版本声明
contract Counter { 2. 定义合约
    uint public counter; 3. 状态变量
    constructor() {
        counter = 0;
    }
    function count() public { 4. 合约函数
        counter = counter + 1;
    }
}
```



定义变量

定义函数

SOLIDITY – 可见性

内外

内部

外部访问

内部及继承

- 使用public、private、external、internal 可见性关键字来控制变量和函数是否可以被外部使用。

```
pragma solidity >=0.8.0;

contract Available {
    function cal(uint a) public pure returns (uint b) { return a + 1; }
    function setData(uint a) internal { data = a; }
    uint public data;
}
```

testAvailable.sol

变量 – 类型

- Solidity 值类型
 - 布尔、**整型**、定长浮点型、定长字节数组、枚举、函数类型、地址类型
 - 十六进制常量、有理数和整型常量、字符串常量、地址常量
- Solidity 引用类型
 - **结构体**
 - **数组**
 - **映射类型**

SOLIDITY – 整型

- 整型： int/uint , uint8 ... uint256

支持运算符

- 比较运算： `<=`, `<`, `==`, `!=`, `>=`, `>`
- 位运算： `&`, `|`, `^`(异或), `~`(位取反)
- 算术运算： `+`, `-`, `-(负)`, `*`, `/`, `%`(取余数), `**` (幂)
- 移位： `<<` (左移位), `>>`(右移位)

在使用整型时，要特别注意整型的大小及所能容纳的最大值和最小值，如uint8的最大值为0xff（255），最小值是0。从solidity 0.6.0 版本开始可以通过 `Type(T).min` 和 `Type(T).max` 获得整型的最小值与最大值。

SOLIDITY – 整型

```
pragma solidity ^0.5.0;
// pragma solidity ^0.8.0;

contract testOverflow {
    function add1() public pure returns (uint8) {
        uint8 x = 128;
        uint8 y = x * 2;
        return y;
    }

    function add2() public pure returns (uint8) {
        uint8 i = 240;
        uint8 j = 16;
        uint8 k = i + j;
    }
}
```

预测一下2个函数分别的结果是什么?
为什么?

testOverflow.sol

SOLIDITY – 地址类型

- Solidity 使用地址类型来表示一个账号，地址类型有两种形式
 - address：一个20字节的值。
 - address payable：表示可支付地址，可调用**transfer**和**send**。

类型转换：address payable ap = payable(addr);

为什么做区分？

SOLIDITY – 地址类型

- 成员函数

- <address>.balance(uint256): 返回地址的余额
- <address payable>.transfer(uint256 amount): 向地址发送以太币, 失败时抛出异常 (gas: 2300)
- <address payable>.send(uint256 amount) returns (bool): 向地址发送以太币, 失败时返回false (gas: 2300)

SOLIDITY – 地址类型

testAddr.sol

```
pragma solidity ^0.6.0;

contract testAddr {

    function testTrasfer(address payable x) public {
        address myAddress = address(this); 合约转换为地址类型

        if (myAddress.balance >= 10) {
            x.transfer(10); 调用 x 的transfer 方法: 向 x 转10 wei
        }
    }
}
```

给一个合约地址转账，即上面代码 x 是合约地址时，合约的receive函数或fallback函数会随着transfer调用一起执行

合约类型

testContract.sol

```
pragma solidity ^0.8.0;

contract Counter{
    function count() public {
    }
}
```

- 创建合约可使用 New 关键字
- 每个合约都是一个类型，可声明一个合约类型。
如：Counter c; 则可以使用c.count() 调用函数
- 合约类型可以显式转换为address类型，从而可以使用地址类型的成员函数。

引用类型

- 值类型赋值 = 拷贝
- 引用类型太大 (>32个字节) , 拷贝开销很大, 可使用“引用”方式, 指向同一个变量。
- 引用类型太大, 不同的位置, 不同的gas费用, 需要有一个属性来标识数据的存储位置:
 - memory (内存) : 生命周期只存在于函数调用期间
 - storage (存储) : 状态变量保存的位置, gas开销最大
 - calldata (调用数据) : 用于函数参数不可变存储区域

数组

- $T[k]$: 元素类型为 T , 固定长度为 k 的数组
 - `uint[10] tens;`
- $T[]$: 元素类型为 T , 长度动态调整
 - `uint[] public numbers;`
- 数组通过下标进行访问, 序号是从0开始
 - `tens[0], numbers[1]`
- `bytes`、`string` 是一种特殊的数组

数组

- 成员
 - Length属性：表示当前数组的长度
 - push(): 添加新的零初始化元素到数组末尾，返回引用
 - push(x): 动态数组末尾添加一个给定的元素
 - pop(): 从数组末尾删除元素

数组

```
pragma solidity ^0.8.0;

contract testArray {
    uint[10] tens;
    uint[] public numbers;

    function copy(uint[] calldata arrs) public returns (uint len) {
        numbers = arrs;
        return numbers.length;
    }

    function test(uint len) public pure {
        string[4] memory adaArr = ["This", "is", "an", "array"];
        uint[] memory c = new uint[](len);
    }

    function add(uint x) public {
        numbers.push(x);
    }
}
```

testArray.sol:

数组

- 数组的使用要注意 Gas 问题，你能发现 dosome() 函数有什么问题吗？

```
function dosome() public {
    uint len = numbers.length;
    for (uint i = 0; i < len; i++) {
        // do...
    }
}

function remove(uint index) public {
    uint len = numbers.length;
    if (i == len - 1) {
        numbers.pop();
        break;
    } else {
        numbers[index] = numbers[len - 1];
        numbers.pop();
        break;
    }
}
```

结构体

使用Struct 声明一个结构体， 定义一个新类型。

testStruct.sol

```
contract testStruct {
    struct Funder {
        address addr;
        uint amount;
    }

    mapping (uint => Funder) funders;

    function contribute(uint id) public payable {
        funders[id] = Funder({addr: msg.sender, amount: msg.value});
        funders[id] = Funder(msg.sender, msg.value);
    }
    function getFund(uint id) public view returns (address, uint ) {
        return (funders[id].addr, funders[id].amount);
    }
}
```

映射

- 声明形式： mapping(KeyType => ValueType) , 例如： mapping(address => uint) public balances;
- 使用方式类似数组，通过 key 访问，例如： balances[userAddr];
- 映射没有长度、没有 key 的集合或 value 的集合的概念
- 只能作为状态变量（storage）
- 如果访问一个不存在的键，返回的是默认值。

映射

testMapping.sol

```
pragma solidity >=0.8.0;

contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }

    function get(address key) public view returns(uint)
    {
        return balances[key];
    }
}
```

定义变量

定义函数

函数

- function + 函数名(参数列表) + 可见性 + 状态可变性 (可多个) + 返回值

```
function update(uint a) public {  
}  
  
function get() public view returns(uint){  
    return a;  
}
```

函数 - 状态可变性

- view: 表示函数只读取状态
- pure: 表示函数不读取状态、也不写 (状态) , 仅计算
- payable: 表示函数可接收以太币
 - 接收的 ETH 值, 用 msg.value 获取

testAvailable.sol

```
function get() public view returns(uint){  
    return a;  
}
```

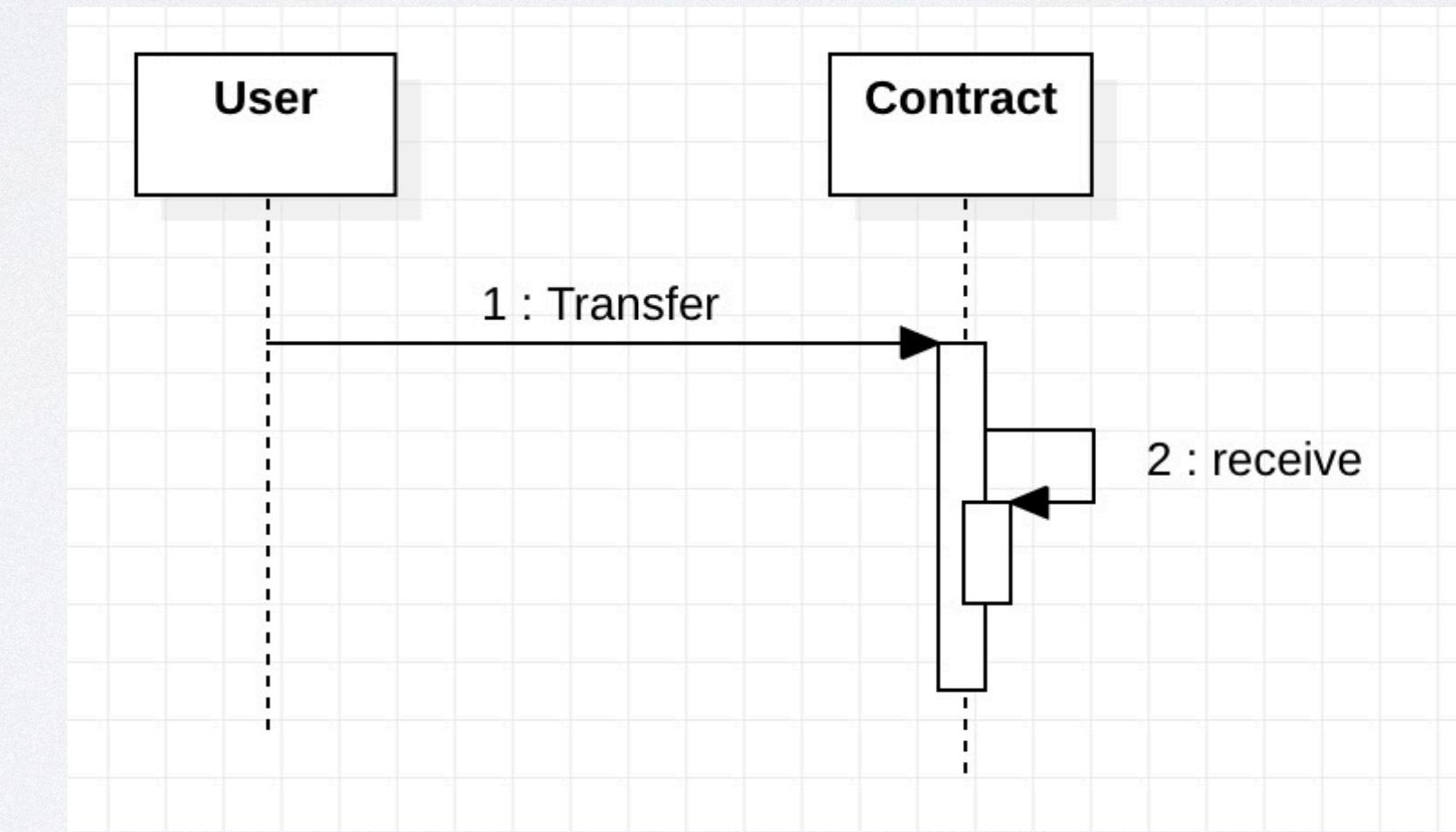
SOLIDITY – 合约特殊函数

- 构造函数(constructor): 初始化逻辑
- getter 函数: 所有 public 状态变量创建 getter 函数
- receive 函数: 接收以太币时回调。
- fallback 函数: 没有匹配函数标识符时, fallback 会被调用, 如果是转账时, 没有 receive 也有调用 fallback

SOLIDITY – 回调函数

- receive / fallback 充当回调函数作用
- 回调函数：有转账了，告诉我（合约）一下。
- fallback : 找不到方法的时候被调用

testPayable.sol



SOLIDITY API – 全局变量及函数

- 区块和交易属性
- 错误处理(require()/revert())
- 数学及密码学函数
- 类型信息 (type(C).creationCode / type(T).min)
- ABI 编码及解码函数
- 地址及合约

<https://learnblockchain.cn/docs/solidity/units-and-global-variables.html#special-variables-and-functions>

区块和交易属性

- `block.number` (`uint`): 当前区块号
- `block.timestamp` (`uint`): 自 unix epoch 起始当前区块以秒计的时间戳
- `msg.sender` (`address`): 消息发送者 (当前调用)
- `msg.value` (`uint`): 随消息发送的 `wei` 的数量
- `tx.origin` (`address payable`): 交易发起者 (完全的调用链)
- ...

testOwner.sol

练习题

- 编写一个Bank 合约（代码提交到github）：
 - 通过 Metamask 向Bank合约存款（转账ETH）
 - 在Bank合约记录每个地址存款金额
 - 编写 Bank合约 withdraw(), 实现只有管理员提取出所有的 ETH
 - 用数组记录存款金额前 3 名

<https://decert.me/quests/c43324bc-0220-4e81-b533-668fa644c1c3>

习题解答

- 理解合约作为一个账号、也可以持有资产
- msg.value / 如何传递 Value
- 回调函数的理解 (receive/fallback)
- Payable 关键字理解
- Mapping 、数组使用