

# 区块链集训营

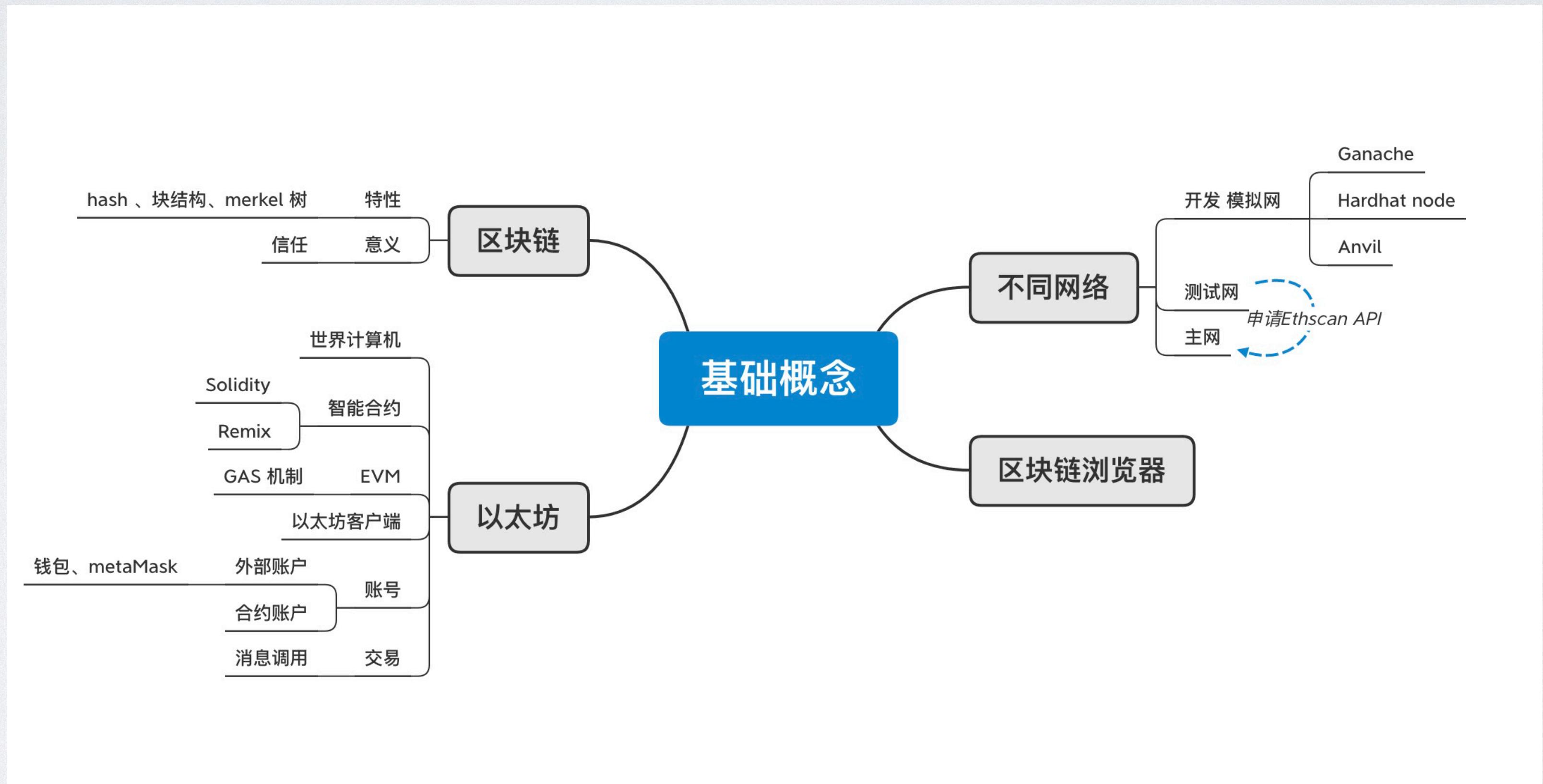
## 二期

登链社区 - Tiny熊

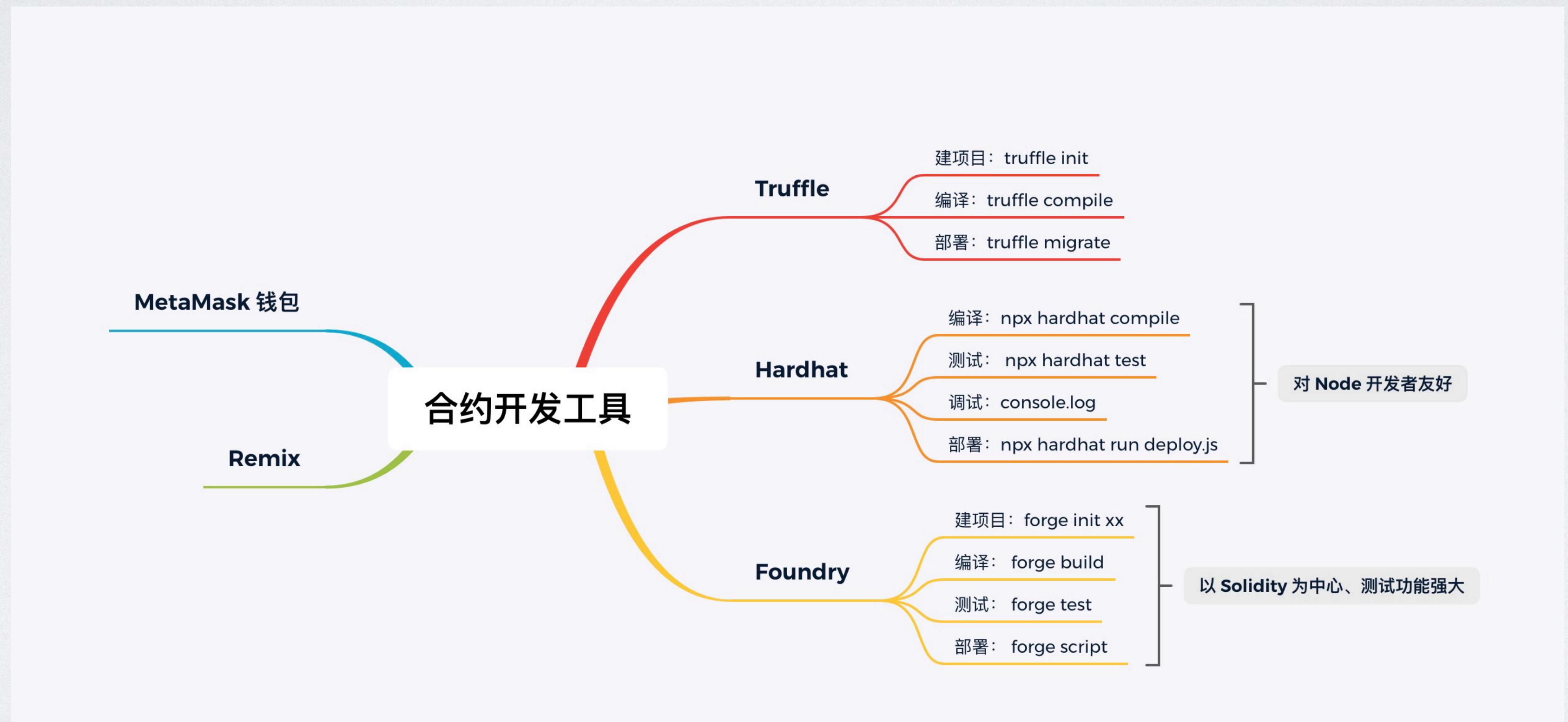
# 课程遇到的问题

- 课件在哪里
  - <https://learnblockchain.cn/course/28>
- 录播在哪里
  - <https://learnblockchain.cn/course/28>
- 代码在哪里
  - [https://github.com/xilibi2003/training\\_camp\\_2](https://github.com/xilibi2003/training_camp_2)
- 预习 + 复习：<https://decert.me/tutorial/solidity/>
- **技能 = 知识 + 实践**
- **坚持 就是 胜利**

# 复习



# 复习



一定要多看文档 看文档 看文档

# 习题解答

- 修改 Counter 合约，仅有部署者 可以调用 count();
- 使用 Hardhat 部署修改后的 Counter
- 使用 Hardhat 测试 Counter:
  - Case 1: 部署者成功调用 count()
  - Case 2: 其他地址调用 count() 失败
- 代码开源到区块浏览器 (npx hardhat verify ...) / 写上合约地址

代码库：[https://github.com/xilibi2003/training\\_camp\\_2](https://github.com/xilibi2003/training_camp_2)  
answer 分支

# 作业遇到问题

- 水龙头获取代币
- Etherscan API 申请（到对应主网）
- 项目配置项（hardhat.config.js，私钥、助记词、etherscan key 配置）
- 网络问题（代码验证、编译器下载）
- 其他问题...

代码库：[https://github.com/xilibi2003/training\\_camp\\_2](https://github.com/xilibi2003/training_camp_2)  
answer 分支

# SOLIDITY语言主要特性

- Solidity 基本类型、数组、结构体、映射
- Solidity API 介绍
- 合约函数、函数修改器、函数修饰符，及各类特殊函数
- 错误处理、合约继承、接口、库及 Openzeppelin 合约库
- 理解合约事件
- 理解ABI

# 学习导图



# 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;
    }
}
```

# 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/>

# SOLIDITY 数据类型

- 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

预测一下2个函数分别的结果是什么?  
为什么?

testOverflow.sol

```
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;
    }
}
```

# SOLIDITY – 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

# SOLIDITY – 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

# SOLIDITY – 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

# SOLIDITY – 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:

# SOLIDITY – 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;
    }
}
```

# SOLIDITY – 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);
    }
}
```

# SOLIDITY – 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。

# SOLIDITY – 映射

```
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];
    }
}
```

testMapping.sol

# SOLIDITY – 全局变量及函数

- 区块和交易
- 错误处理(require()/revert())
- 数学及加密(sha256)
- 类型信息 (type(C).creationCode / type(T).min)
- **ABI 编码**
- 地址及合约

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

# SOLIDITY – 全局变量及函数

- `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)
- ...

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

# ABI

- ABI: Application Binary Interface 应用程序二进制接口
- ABI 接口描述: 定义如何与合约交互
- ABI 编码
  - 函数选择器: 对函数签名计算Keccak-256哈希, 取前 4 个字节
  - 参数编码

# ABI

调用一个合约函数 = 向合约地址发送一个交易

交易的内容就是 ABI 编  
码数据

# Calling a Smart Contract



Address: 0x0123456.....

## 1 Convert function call to HEX

myFunction(parameters) → → 0xabcd0123456789.....

## 2 Put the Information into Transaction object

```
{  
  "to": "0x0123456.....",  
  "value": 0, // No need to send money here  
  "data": "0xabcd0123456789....."  
}
```

## 3 Sign the Transaction with your Private key

{ ... } + Private Key → → 0xfedcba9876...

Signed Transaction  
Can only be decrypted with YOUR public key  
Only you can have sent this transaction

## 4 Send the transaction to the Ethereum Network



第2周

# ABI

Tx Hash: <https://mumbai.polygonscan.com/tx/0x6d45ded0fa162200c9b589ded76697830ea69652f5235d66ff671932d4aee51d>

<https://chaintool.tech/convertABI>

bytes4(keccak256("count()")) = 0x06661abd

Gas Price:	0.00000002 Ether (20 Gwei)
Gas Limit & Usage by Txn:	41,614   41,614 (100%)
<hr/>	
Others:	Nonce: 1 Position: 1
<hr/>	
Input Data:	Function: count() ***  MethodID: 0x06661abd
<hr/>	
View Input As ▾	

ABI 编码 API : abi.encodeWithSignature(string sigs)

# 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**。
- 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

# 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

# 地址类型 – 底层调用

- 底层函数: **call**, **delegatecall**, **staticcall** (不修改状态)
  - <address>.call(bytes memory) returns (bool, bytes memory)

调用合约的count() 方法:

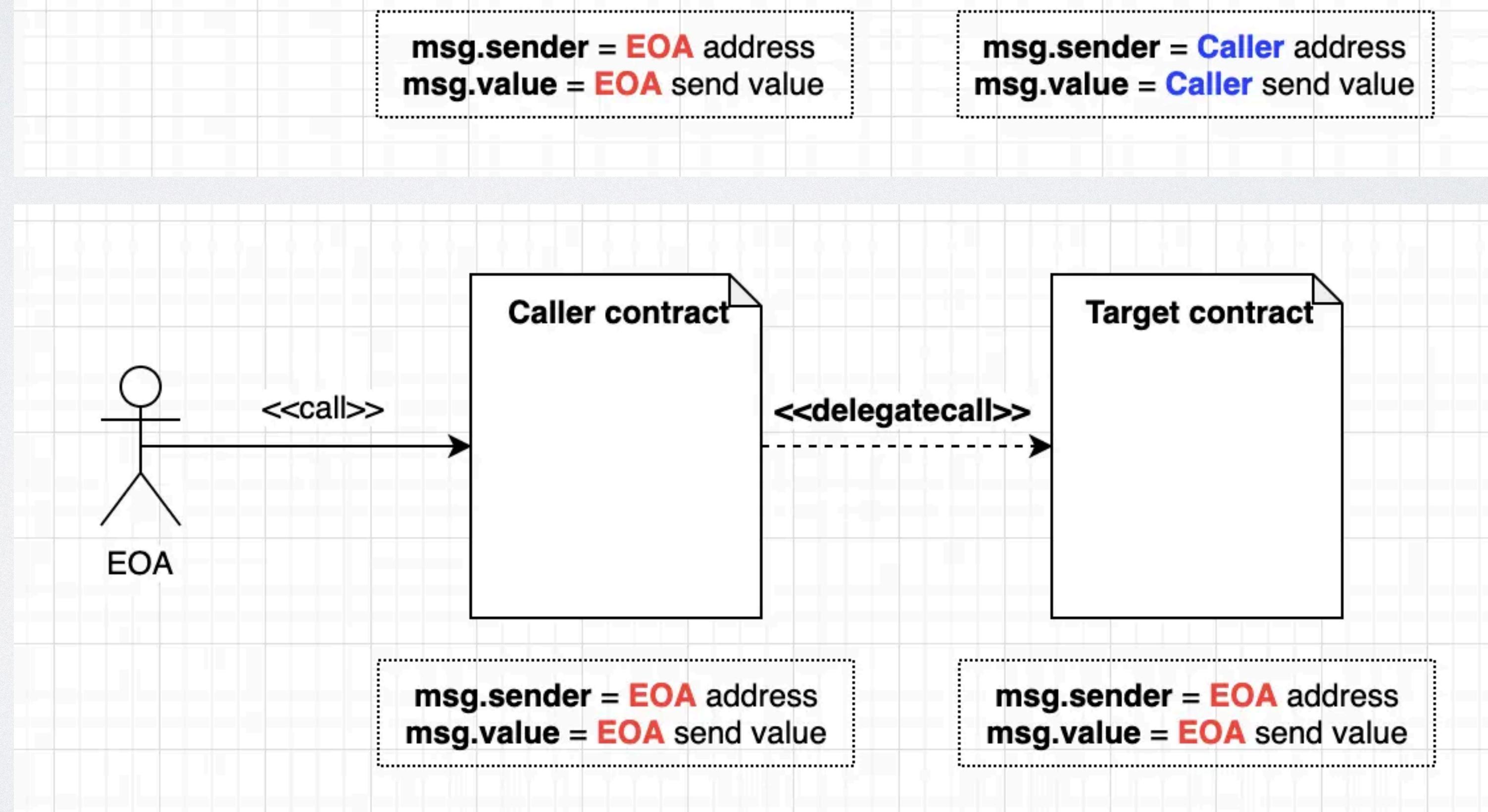
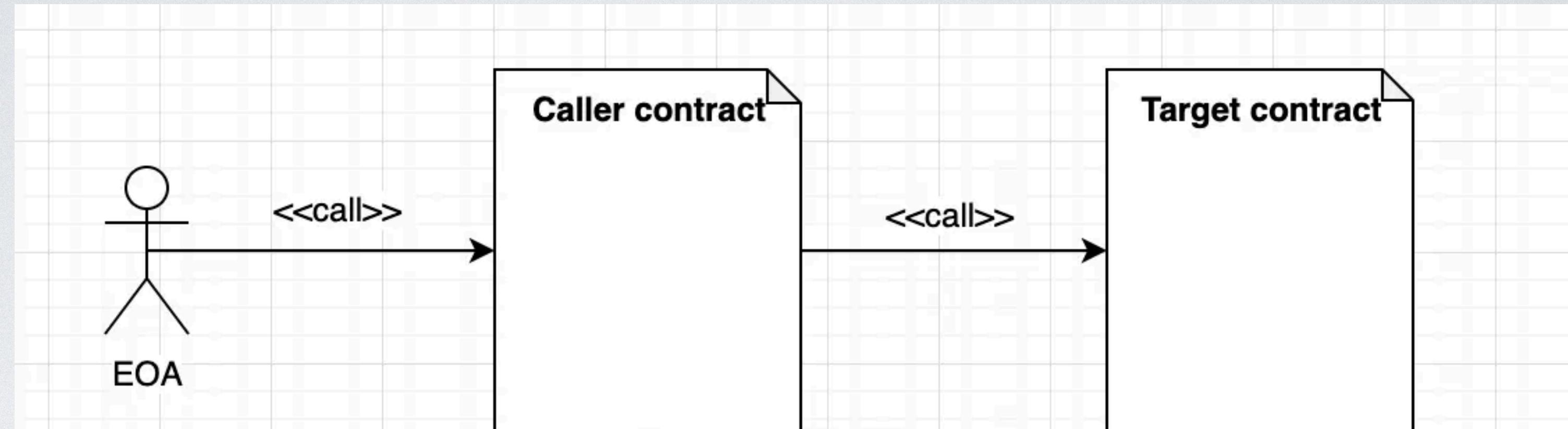
```
function callCount(Counter c) public {  
    c.count();  
}
```

```
function lowCallCount(address addr) public {  
    bytes memory methodData = abi.encodeWithSignature("count()");  
    addr.call(methodData);  
}
```

# 地址 – 底层调用

- 底层调用失败不是发生异常 (revert) , 而是用返回值表示
- call(): 切换上下文
- delegatecall(): 保持上下文

testCall\_Delegate.sol



# 地址 – CALL 转账

- <addr>.transfer()
- gas 限制几乎无法向合约转账，怎么办？

```
addr.call{value: 1 ether}(new bytes(0)) = addr.transfer(1 ether)
```

<https://github.com/Uniswap/solidity-lib/blob/master/contracts/libraries/TransferHelpers.sol>

# SOLIDITY – 合约类型

testABI.sol

```
pragma solidity ^0.8.0;

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

- 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。

# Q & A

# 练习题

- 编写一个Bank合约：
  - 通过 Metamask 向Bank合约转账ETH
  - 在Bank合约记录每个地址转账金额
  - 编写 Bank合约 withdraw(), 实现提取出所有的 ETH

# 帮助改进课程

- 花 2 分钟填写问卷



<https://www.wenjuan.com/s/UZBZJvuLKtf/#>