DAY3： SOLIDITY

登链社区 - Tiny熊

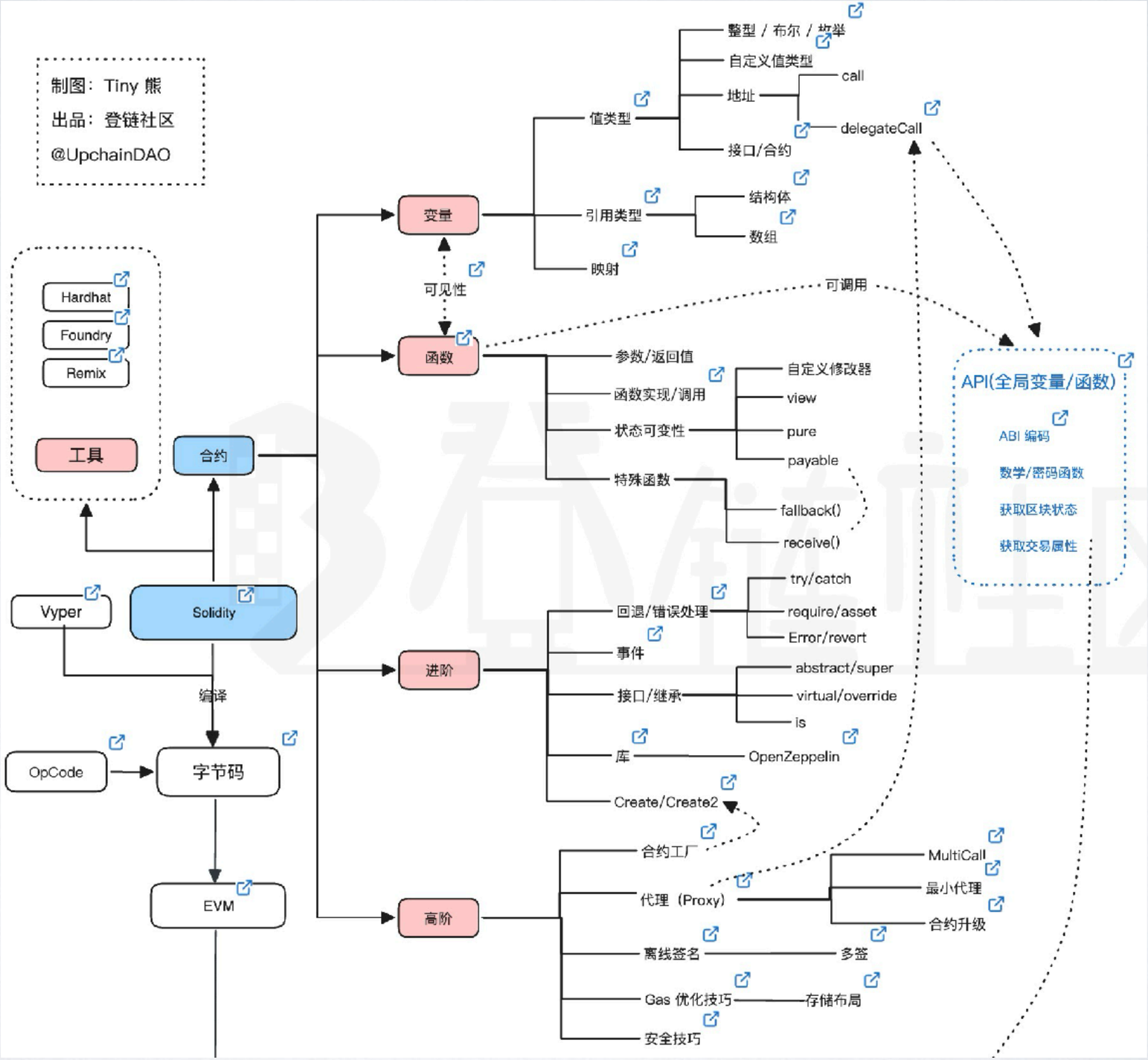
要点

- Solidity 各种不同类型的变量
  - 值类型：整型、Bool、**地址类型、合约类型**
  - 引用类型: 数组、结构体
  - 映射类型
- Solidity 函数： 调用函数、特殊函数：
  - **构造函数**
  - **Receive \ Fallback 函数**

SOLIDITY 语言

- 静态编译型、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/
  - Demo： https://github.com/OpenSpace100/blockchain-tasks/tree/main/solidity_sample_code

Solidity 图谱


制图：Tiny 熊
出品：登链社区
@UpchainDAO
Hardhat
Foundry
Remix
工具
合约
Vyper
Solidity
编译
OpCode
字节码
EVM
变量
函数
进阶
高阶
值类型
整型 / 布尔 / 枚举
自定义值类型
地址
call
delegateCall
接口/合约
引用类型
结构体
数组
映射
可见性
可调用
参数/返回值
函数实现/调用
状态可变性
自定义修改器
view
pure
payable
特殊函数
fallback()
receive()
API(全局变量/函数)
ABI 编码
数学/密码函数
获取区块状态
获取交易属性
回退/错误处理
try/catch
require/asset
Error/revert
事件
接口/继承
abstract/super
virtual/override
is
库
OpenZeppelin
Create/Create2
合约工厂
代理（Proxy）
MultiCall
最小代理
合约升级
离线签名
多签
Gas 优化技巧
存储布局
安全技巧

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;
    }

}
```

定义函数

定义：

function 函数名([参数列表]) 可见性 [状态可变性] [返回值] {
}

```
pragma solidity ^0.8.0;
contract Counter {
    uint public counter;

    // 定义函数
    function count(uint i) public {
        counter = counter + i;
    }

}
```

SOLIDITY – 函数可见性

格式：function 函数名([参数列表]) 可见性 [可变性] [返回值]

- 可见性关键字来控制函数是否可以被外部使用。
- 四种可见性：public（合约内/外）、private（内部）、external（外部访问）、internal（内部及继承）

TestVariable.sol

```
pragma solidity >=0.8.0;


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

函数 – 状态可变性

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

- View：表示函数只读取状态 （不需要支付手续费）
- 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
- 自定义修改器 （modifier）

```
 // view  不修改状态
function addTo(uint y) public view returns (uint){
    return counter + y;
}

 // pure
 function add(uint i, uint j) public pure returns uint){
    return i + j;
 }

 // 存款
 function deposit() public payable {
    deposited += msg.value;
 }
```

testViewPure.sol

函数 – 函数体

由表达式和控制语句构成

for/if/else/while/do/

break/continue/return

没有 goto/switch

TestFuncbody.sol

```
function testFor() public pure {
    for (uint256 i = 0; i < 10; i++) {
        if (i == 3) {
            continue;    // 忽略下一次循环
        }
        if (i == 5) {
            break;    // 退出
        }
    }
}

function testWhile() public pure returns (uint) {
    uint256 i;
    while (i < 10) {
        i++;
        if (i == 5 || i == 8) {
            // …
        }
    }
    return i;
}
```

定义变量

格式：变量类型 [可见性] [可变状态] 变量名 [赋值];

```
pragma solidity ^0.8.0;
contract TestVariable {
    uint public counter;     // 定义状态变量

    function count() public {
        uint256 a = 503;     // 定义本地变量
        counter = a + 1;
    }
}
```

定义变量 – 可见性

格式：变量类型 [可见性] [可变状态] 变量名 [赋值];

- 可见性关键字来控制变量是否可以被外部使用。
- 三种可见性：public（合约内/外）、 private（内部）、 internal（内部及继承）
  - public 的变量会生成一个同名的函数（称为 getter 访问器）

```
pragma solidity >=0.8.0;


contract TestVariable {
    uint public counter;
}
```

定义变量 – 可变状态

格式：变量类型 [变量可见性] [可变状态] 变量名 [赋值];

constant：不可修改的变量，使用硬编码
immutable：不可修改的变量，在构造时赋值

```
pragma solidity ^0.8.0;
contract TestVariable {
    uint private constant TEN = 10;    // 定义常量
    uint public immutable THRESHOLD;

    constructor(uint t) {
        THRESHOLD = t;        // 在构造时赋值
    }
}
```

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**、自定义类型
  - 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**

类型 – 布尔

定义： `bool public isOpen = true;`

支持运算符：

- 比较运算： `==`, `!=`
- 逻辑运算：`!` （取反） ， `&&` （且） ，`||` (或)

常与 If/else 一起使用，注意运算时的短路原则

类型 – 布尔

testBool.sol

```
contract testBool {

    bool public isOpen = true;
    bool public isChanged = false;

    function setOpen(bool o) public  {
        isOpen = o;
        if( isOpen || tryChange() ) {  // 注意短路
            // ....
        }
    }

    function tryChange() internal returns (bool) {
        isChanged = !isChanged;
        return isChanged;
    }
}
```

SOLIDITY – 整型

- 整型： int/uint , uint8、uint16、 … 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 – 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>.code / .codehash: 返回地址的链上代码 / 链上代码hash
  - <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);

     if (myAddress.balance >= 10) {
         x.transfer(10);
     }
  }
}
```

合约转换为地址类型

调用 x 的transfer 方法: 向 x 转10 wei

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。

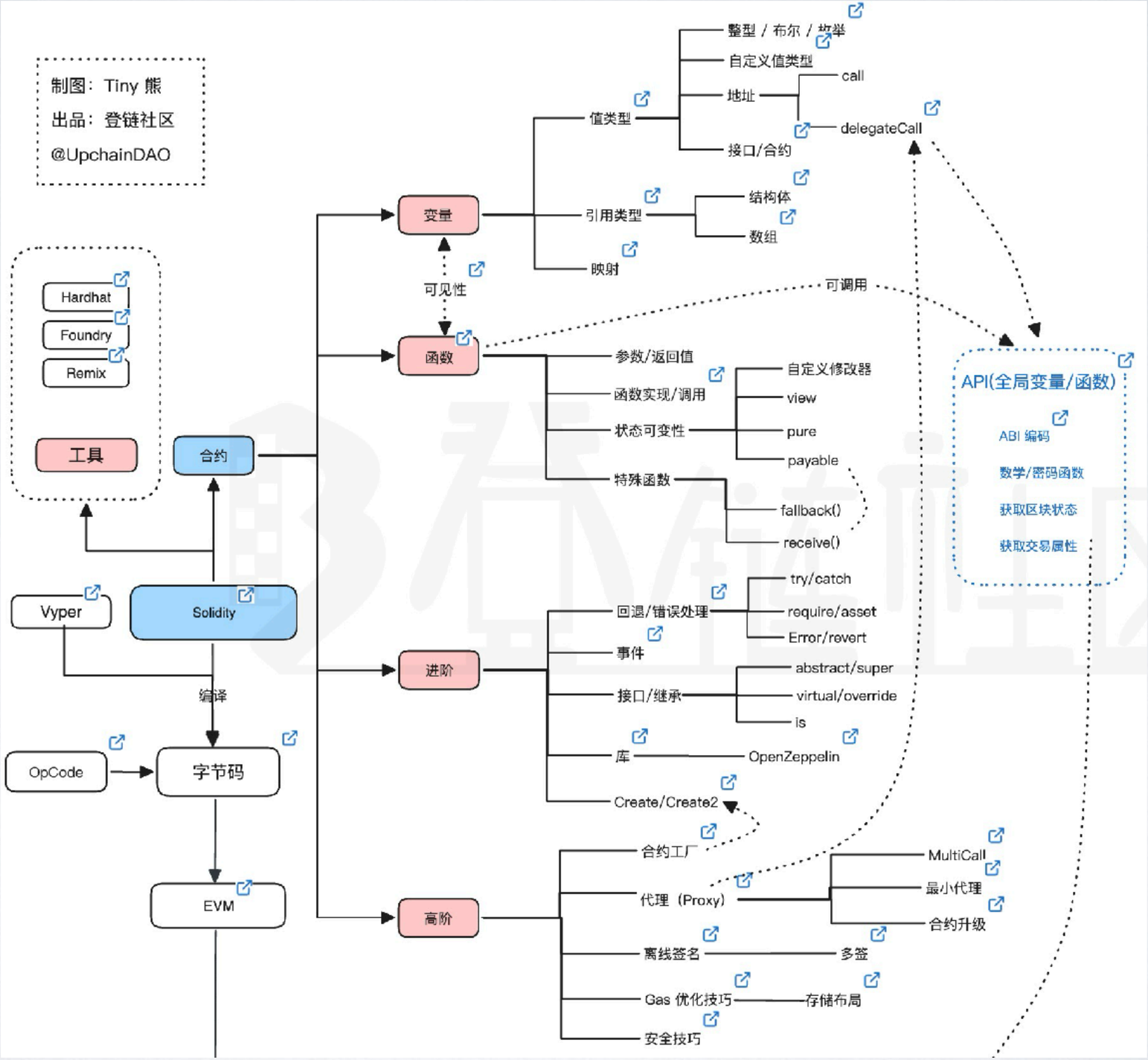
类型 – 枚举类型

testEnum.sol

```
contract testEnum {

    enum Color { Red, White, Blue }
    Color color;

    function setColor(Color c) public {
        color = c;
    }

    function getMaxValue() public pure returns (Color) {
        return type(Color).max;
    }

    function getMinValue() public pure returns (Color) {
        return type(Color).min;
    }
}
```

- 定义一个枚举类型：enum 枚举名 { 成员 }
- 枚举成员从 0 编号、不能多于 256 个成员

Solidity 图谱


制图：Tiny 熊
出品：登链社区
@UpchainDAO
Hardhat
Foundry
Remix
工具
合约
Vyper
Solidity
编译
OpCode
字节码
EVM
变量
函数
进阶
高阶
值类型
整型 / 布尔 / 枚举
自定义值类型
地址
call
delegateCall
接口/合约
引用类型
结构体
数组
映射
可见性
可调用
参数/返回值
函数实现/调用
状态可变性
自定义修改器
view
pure
payable
特殊函数
fallback()
receive()
API(全局变量/函数)
ABI 编码
数学/密码函数
获取区块状态
获取交易属性
回退/错误处理
try/catch
require/asset
Error/revert
事件
接口/继承
abstract/super
virtual/override
is
库
OpenZeppelin
Create/Create2
合约工厂
代理（Proxy）
MultiCall
最小代理
合约升级
离线签名
多签
Gas 优化技巧
存储布局
安全技巧

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（memory、storage、calldata）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
  - transient (瞬态存储）: 类似状态变量，但写入的值仅在一个交易内有效（仅支持值类型）
  - calldata（调用数据）: 用于函数参数不可变存储区域

不同的位置 赋值的时候做拷贝

引用类型 – 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

类型 – 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

类型 – 数组

testArray.sol:

```
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);
    }
}
```

类型 – 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;
    }
}
```

类型 – 数组进阶

- 多维数组 uint[ ][ ] groups;
- bytes： 字节数组类型， 还有定长字节数组 bytes1 、bytes2 … bytes32
  - 写入的字符，会转为字节
  - 使用 .concat 拼接
- string 也是数组类型，但没有长度和下标 （没有char）， 使用 .concat 拼接
- 数组切片，仅支持 calldata，主要用于 bytes
  - x[0:2] x[:2] 取前两个元素
  - x[4:] 取第 4 个到末尾
- pulibc 数组 生成带有参数的 getter 函数（访问器）

testArray2.sol:

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 Person {
        address account;
        bool gender;
        uint8 age;
    }

    Person public chairman;    // 定义变量

    function setChairman(address _acc, bool _g, uint8 _age) public {
        chairman = Person(_acc, _g, _age);  // 创建变量
        // Person memory person =  Person({gender: _g, account: _acc, age: _age}) ;
    }

    function getChairman() public view returns  (address, bool, uint8 ) {
        return (chairman.account, chairman.gender, chairman.age); // 使用 . 获取成员
    }

}
```

类型 – 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];
- 如果访问一个不存在的键，返回的是默认值。
- Public 的映射变量 生成带有参数的 getter 函数（访问器）
- 限制：
  - 只能作为状态变量（storage）
  - KeyType 不能是数组（string 和 bytes 是例外）
  - 映射没有长度、没有 key 的集合或 value 的集合的概念

类型 – 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];
    }

}
```

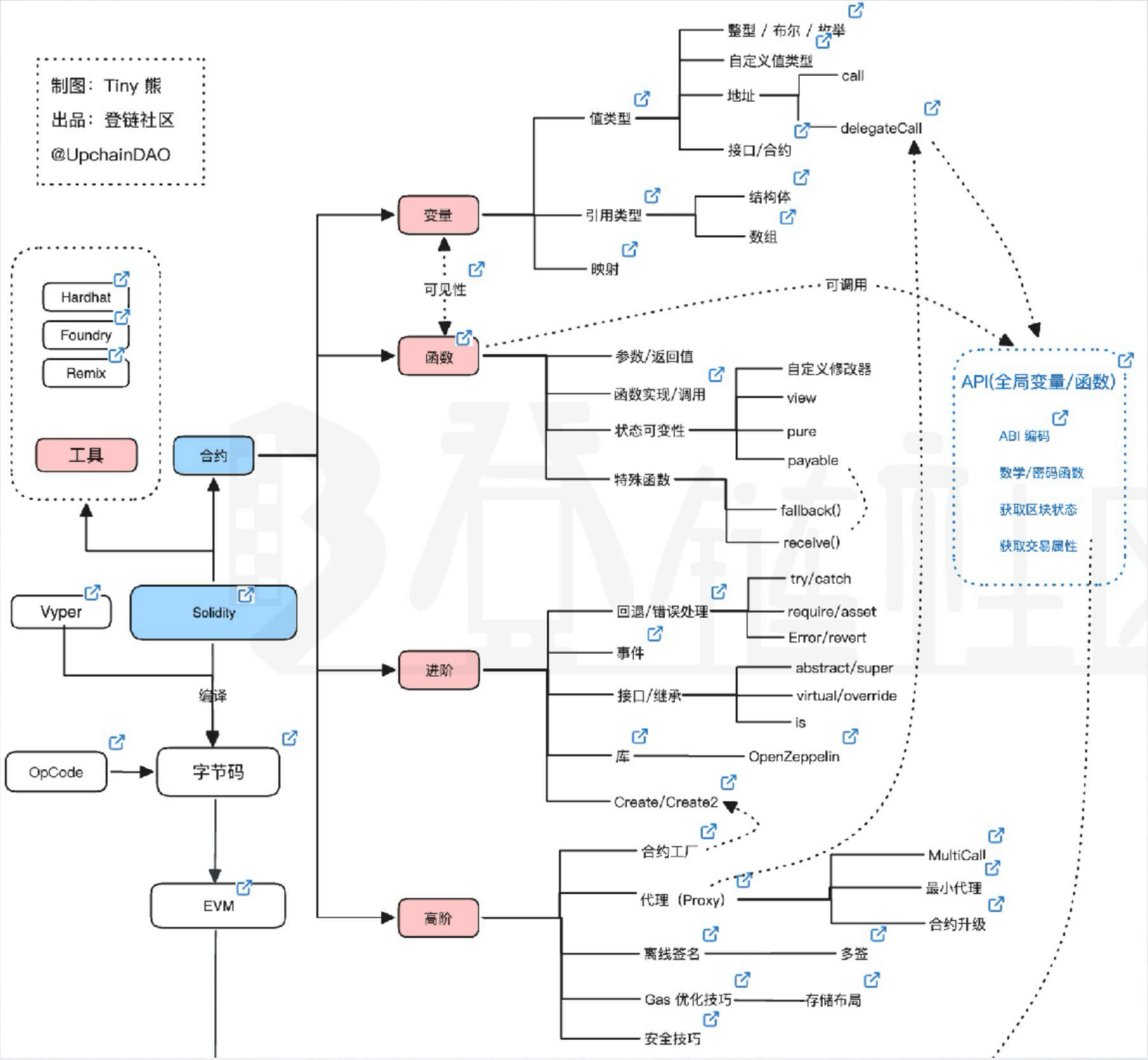
结构体映射组合使用

结构体成员是映射，映射的Value 是结构体

testStruct.sol

```
contract testStruct {

    struct Student {
        string name;
        mapping(string=>uint) score;   //使用较少
        int age;
    }

    struct Funder {
        address addr;
        uint amount;
    }

    mapping (uint => Funder) funders;

}
```

Solidity 图谱


制图：Tiny 熊
出品：登链社区
@UpchainDAO
Hardhat
Foundry
Remix
工具
合约
Vyper
Solidity
编译
OpCode
字节码
EVM
变量
函数
进阶
高阶
值类型
整型 / 布尔 / 枚举
自定义值类型
地址
call
delegateCall
接口/合约
引用类型
结构体
数组
映射
可见性
可调用
参数/返回值
函数实现/调用
状态可变性
自定义修改器
view
pure
payable
特殊函数
fallback()
receive()
API(全局变量/函数)
ABI 编码
数学/密码函数
获取区块状态
获取交易属性
回退/错误处理
try/catch
require/asset
Error/revert
事件
接口/继承
abstract/super
virtual/override
is
库
OpenZeppelin
Create/Create2
合约工厂
代理（Proxy）
MultiCall
最小代理
合约升级
离线签名
多签
Gas 优化技巧
存储布局
安全技巧

函数调用

- 具名调用函数
- 有内部调用和外部调用（跨合约调用）
- 内部函数调用： get();
- 外部函数调用： 合约地址.get(); this.get()
  - 可以指定发送的 value 和 gas： {value: 10, gas: 10000}， 但不建议指定 gas
- 内部函数调用高效在同一个 EVM 环境里，外部调用会切换上下文环境

TestCallFunc.sol

SOLIDITY – 特殊函数

- 有一些不使用 function 来定义的函数：
  - getter 函数(访问器)：不需要定义，所有 public 状态变量自动创建 getter 函数
  - constructor(构造函数)：初始化逻辑
- 回调函数（不可主动调用）
  - receive 函数：接收以太币时被调用。
  - fallback 函数：EVM 没有找到用户要调用的函数时，fallback 会被调用，如果是转账时，没有 receive 也有调用 fallback

特殊函数 – 构造函数

- 构造函数(constructor)：初始化逻辑
- 构造函数仅在部署的时候执行一次，链上的字节码没有构造函数，而是执行构造函数之后的结果

不同的字节码

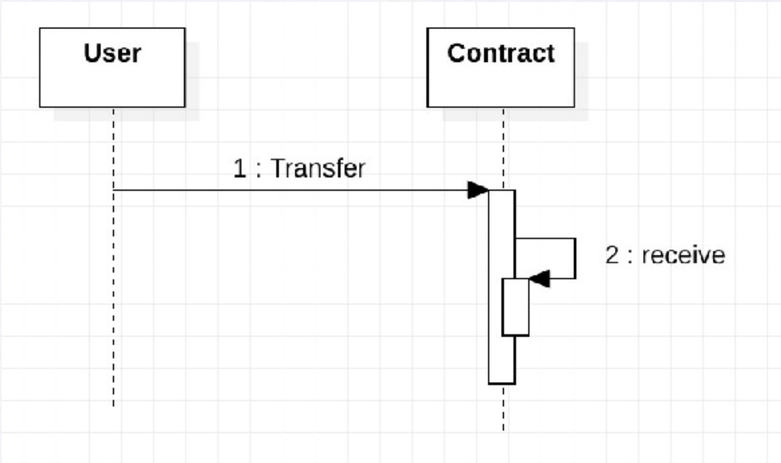
- 编译产生字节码
- 创建字节码：部署合约，发送交易字节码，包含可能的构造参数
- 链上字节码：运行时字节码
  - address.code 返回这个结果

有构造函数时，他们有所区别，我们 Demo 一下

特殊函数 – RECEIVE 回调函数

- 回调函数不可以直接调用（在 Remix 左下角看不到）
- receive 函数：合约地址在接收以太币时被调用

testPayable.sol



接受以太币时执行逻辑

合约向合约转账

回顾地址类型，通过地址类型是可以向合约转账的，此时也会执行回调函数。

testAddr.sol

```
pragma solidity ^0.6.0;

contract testAddr {

  function testTrasfer(address payable x) public {
     address myAddress = address(this);

     if (myAddress.balance >= 10) {
         x.transfer(10);
     }
  }
}
```

合约转换为地址类型

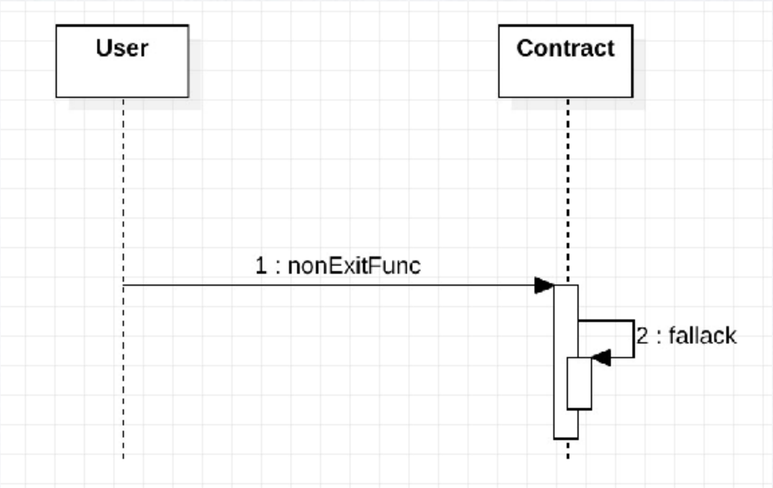
调用 x 的transfer 方法：向 x 转10 wei

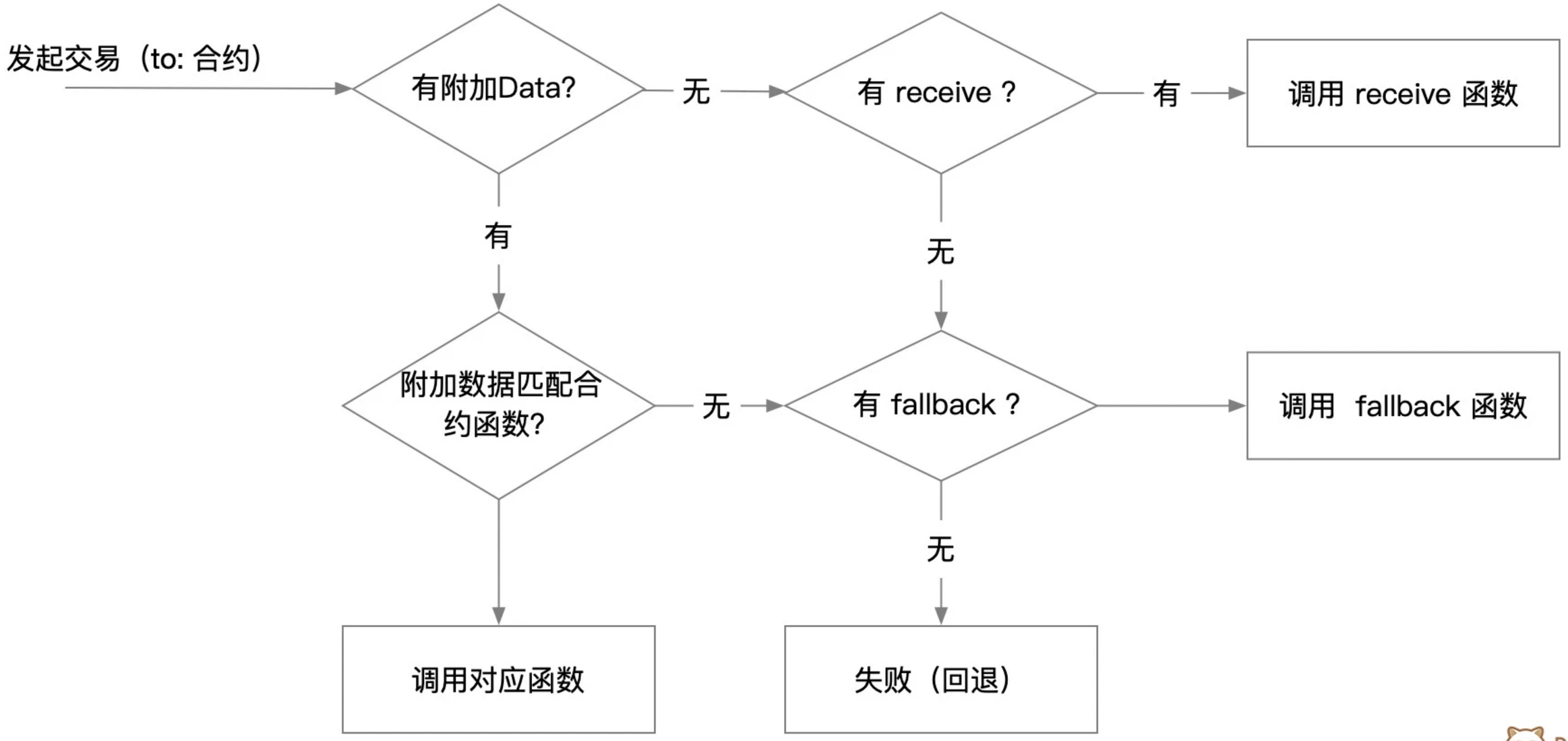
给一个合约地址转账，即上面代码 x 是合约地址时，合约的receive函数或fallback函数会随着transfer调用一起执行

特殊函数 – FALLBACK 回调函数

- fallback 函数: EVM 没有找到用户要调用的函数时，fallback 会被调用， 如果没有 fallback 调用失败
  - 如果是转账时，没有 receive 时，会找 fallback 函数

testPayable.sol





https://decert.me/tutorial/solidity/solidity-basic/receive

回调函数小结

合约接收以太币必须显式实现 receive 或 fallback，否则无法接收。

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（tyep(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
  - 用数组记录存款金额前 3 名
  - 编写 Bank合约 withdraw(), 实现只有管理员提取出所有的 ETH

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

练习

- DEMO 代码，自己运行一次，动手做一些改动

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