

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

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

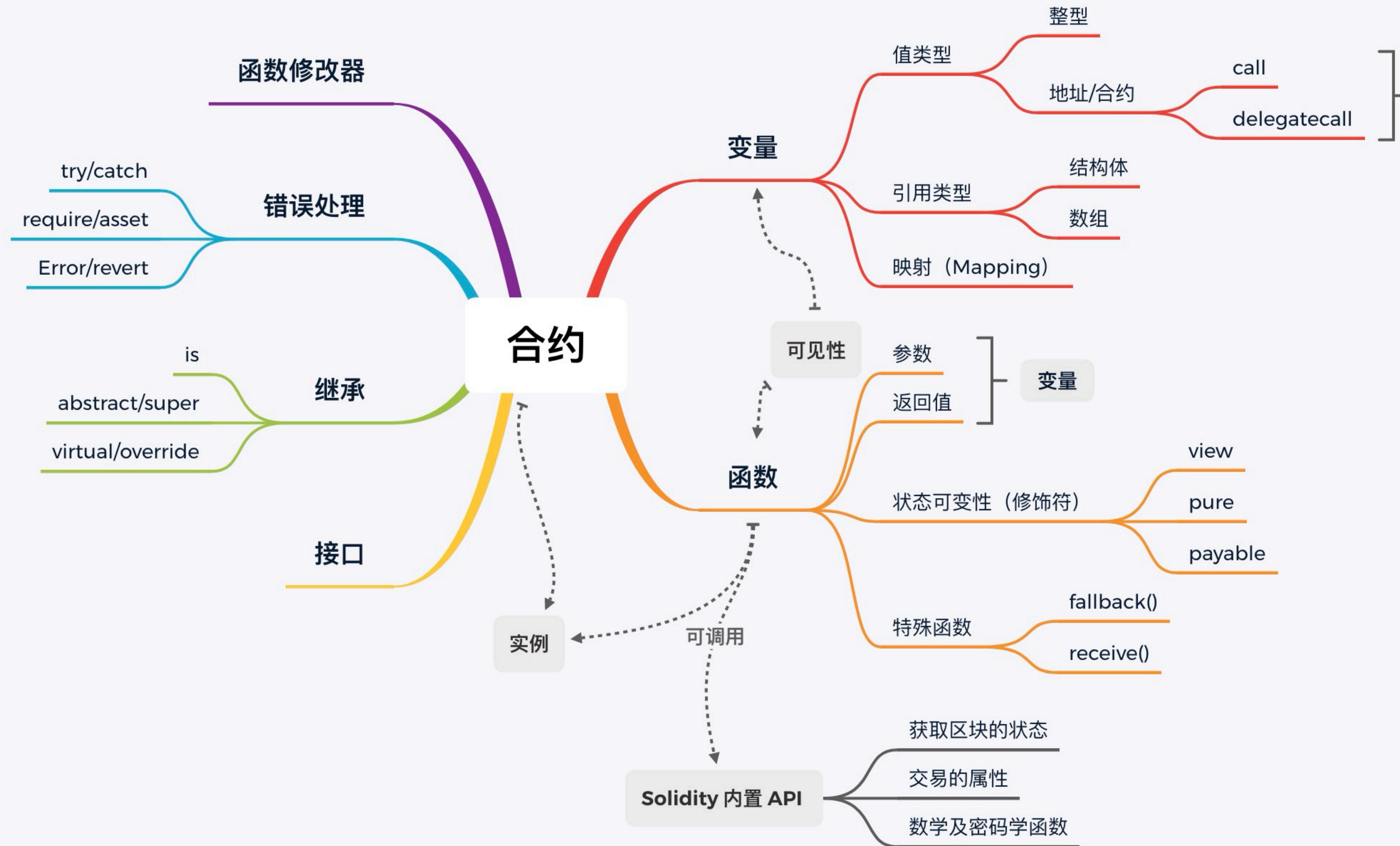
常见问题

- 为什么没有 main 函数?
- 不用保存 total balance
- 不用同时保存 address[] 和 uint[]
- 不用再次执行 payable(address(this)).transfer(msg.value);
- 排序代码复用 (在 receive 及 deposit 中)

DAY4: SOLIDITY

登链社区 - Tiny熊

下节课



自定义修饰符 – 函数修改器

- 用 modifier 定义一个修改器

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}
```

- 用修改器修饰一个函数，用来添加函数的行为，如检查输入条件、控制访问、重入控制

```
function withdraw() public onlyOwner {
    // do something
}
```

testModifier

```
pragma solidity ^0.8.0;
contract testModifier {
    address public owner;
    uint private deposited;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        _;
    }

    function withdraw() public onlyOwner {
        payable(owner).transfer(deposited);
    }
}
```

函数修改器修饰函数时，函数体被插入到“_”

SOLIDITY – 接口

- **类型声明**（函数的抽象），广泛用于合约之间的调用
- 无任何实现的函数
- 不能继承自其他接口
- 没有构造方法
- 没有状态变量

testInterface.sol

```
pragma solidity ^0.8.0;

interface ICounter {
    function count() external view returns (uint);
    function increment() external;
}

contract Counter is ICounter{
    uint public count;

    function increment() external {
        count += 1;
    }
}

contract MyContract {
    function incrementCounter(address _counter) external {
        ICounter(_counter).increment();
    }

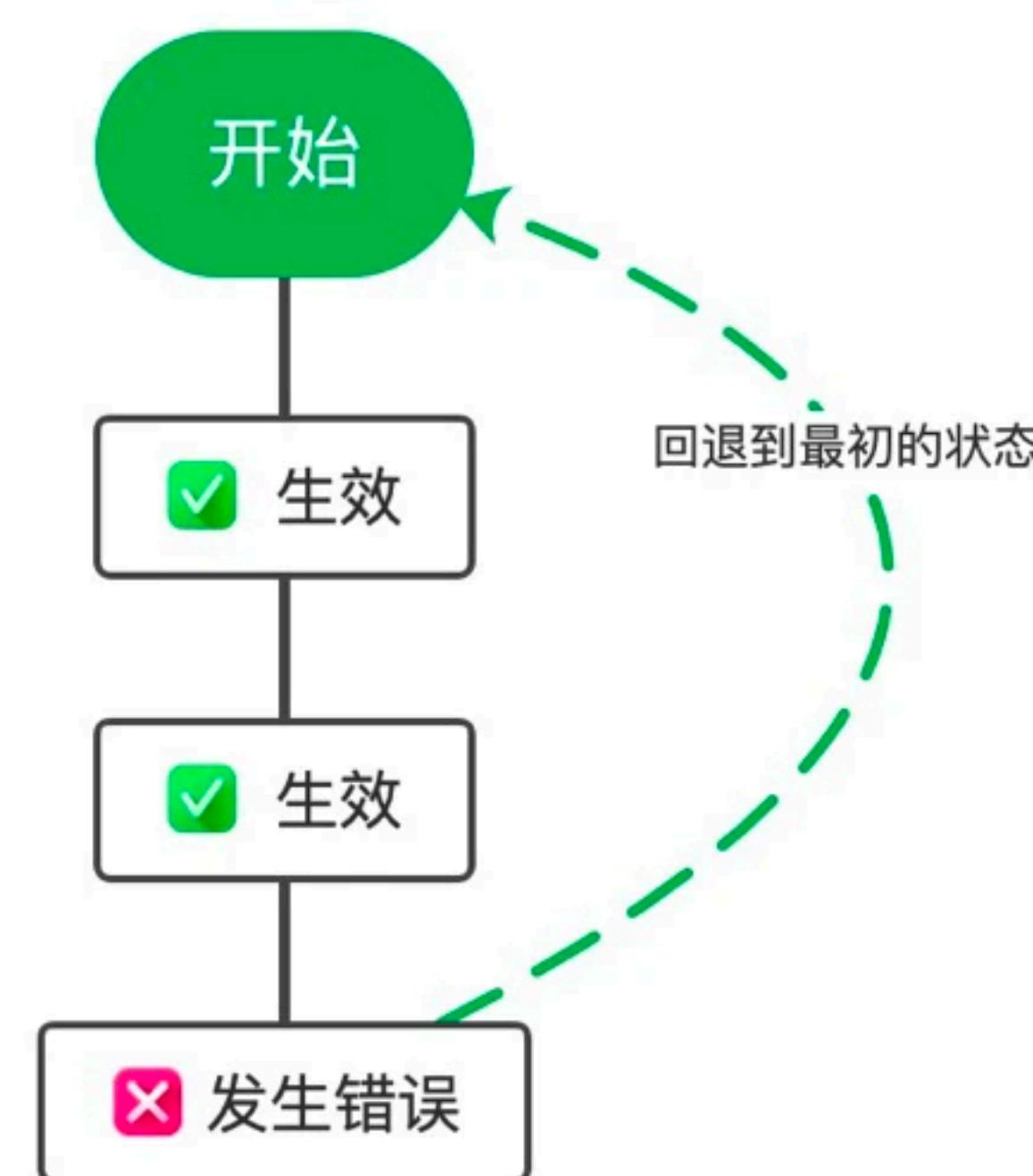
    function getCount(address _counter) external view returns (uint) {
        return ICounter(_counter).count();
    }
}
```

SOLIDITY – 错误处理

Java 等语言错误处理



EVM 错误处理



SOLIDITY – 错误处理

- 在程序发生错误时的处理方式：EVM通过回退状态来处理错误的，以便保证状态修改的**事务性**
- assert()和**require()**用来进行条件检查，并在条件不满足时抛出异常
- revert("msg")：终止运行并撤销状态更改
- Error 定义错误

```
require(msg.sender == owner, "Not owner");
```

```
error NotOwner();
if(msg.sender != owner) revert NotOwner();
```

SOLIDITY – TRY/CATCH

- try/catch: 捕获合约中外部调用的异常
 - 即便是不存在的函数调用也可以捕获
 - 合约不存在则无法捕获
 - 注意: out of gas 错误不是程序异常, 错误不能捕获。

testtrycatch.sol

```
pragma solidity ^0.8.0;

contract Foo {
    function myFunc(uint x) public pure returns (uint ) {
        require(x != 0, "require failed");
        return x + 1;
    }
}

contract trycatch {

    Foo public foo;
    uint public y;
    constructor() {
        foo = new Foo();
    }

    function tryCatchExternalCall(uint _i) public {
        try foo.myFunc(_i) returns (uint result) {
            y = result;
        } catch {
            ..
        }
    }
}
```

SOLIDITY – 继承

- 和大多数高级语言一样，Solidity 也支持继承
- 使用关键字 `is`
- 继承时，链上实际只有一个合约被创建，基类合约的代码会被编译进派生合约。
- 派生合约可以访问基类合约内的所有非私有（`private`）成员，因此内部（`internal`）函数和状态变量在派生合约里是可以直接使用的

testIs.sol

```
pragma solidity ^0.8.0;

contract A {
    uint public a;
    constructor() {
        a = 1;
    }
}

contract B is A {
    uint public b ;
    constructor() {
        b = 2;
    }
}
```

在部署B时候，可以查看到a为1，b为2。

SOLIDITY – 继承、抽象合约

- abstract 抽象合约
 - 不能被部署，可包含没有实现的纯虚函数
- super：调用父合约函数
- virtual：表示函数可以被重写
- override：表示重写了父合约函数

testAbstract.sol

```
pragma solidity ^0.8.0;

abstract contract A {
    uint public a;
    function add(uint x) public virtual ;

contract B is A {
    uint public b ;
    constructor() {
        b = 2;
    }

    function add(uint x) public override virtual {
        b += x;
    }
}
```

练习题

- 编写一个 BigBank 合约继承自 Bank, 要求:
 - 仅 >0.001 ether (用modifier权限控制) 可以存款
- 把管理员转移给 Ownable 合约, 只有 Ownable 可以调用 BigBank 的 withdraw().

<https://decert.me/quests/d0600476-7ce8-4648-aid2-58f15ebac73f>