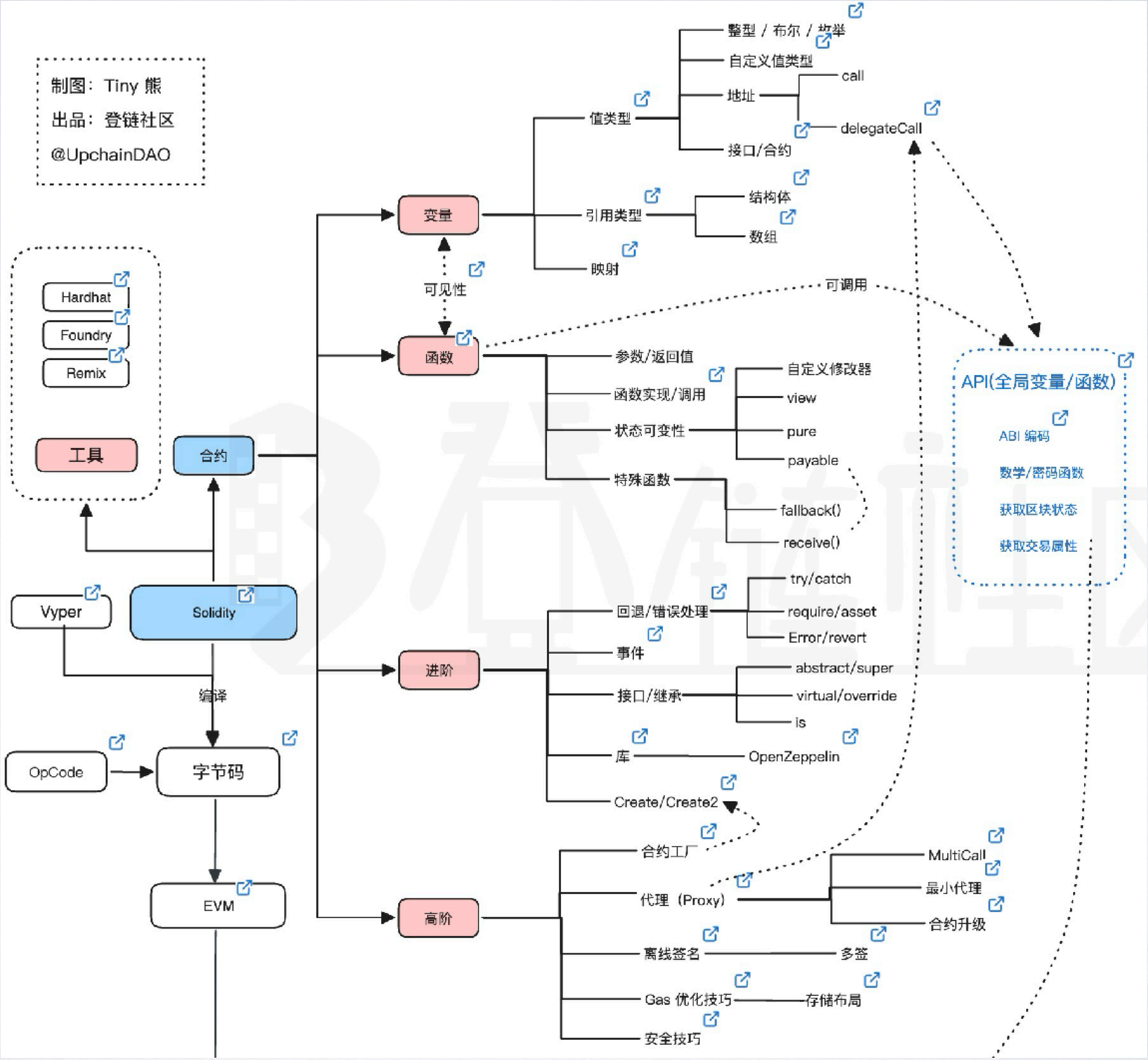
Solidity 图谱


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

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

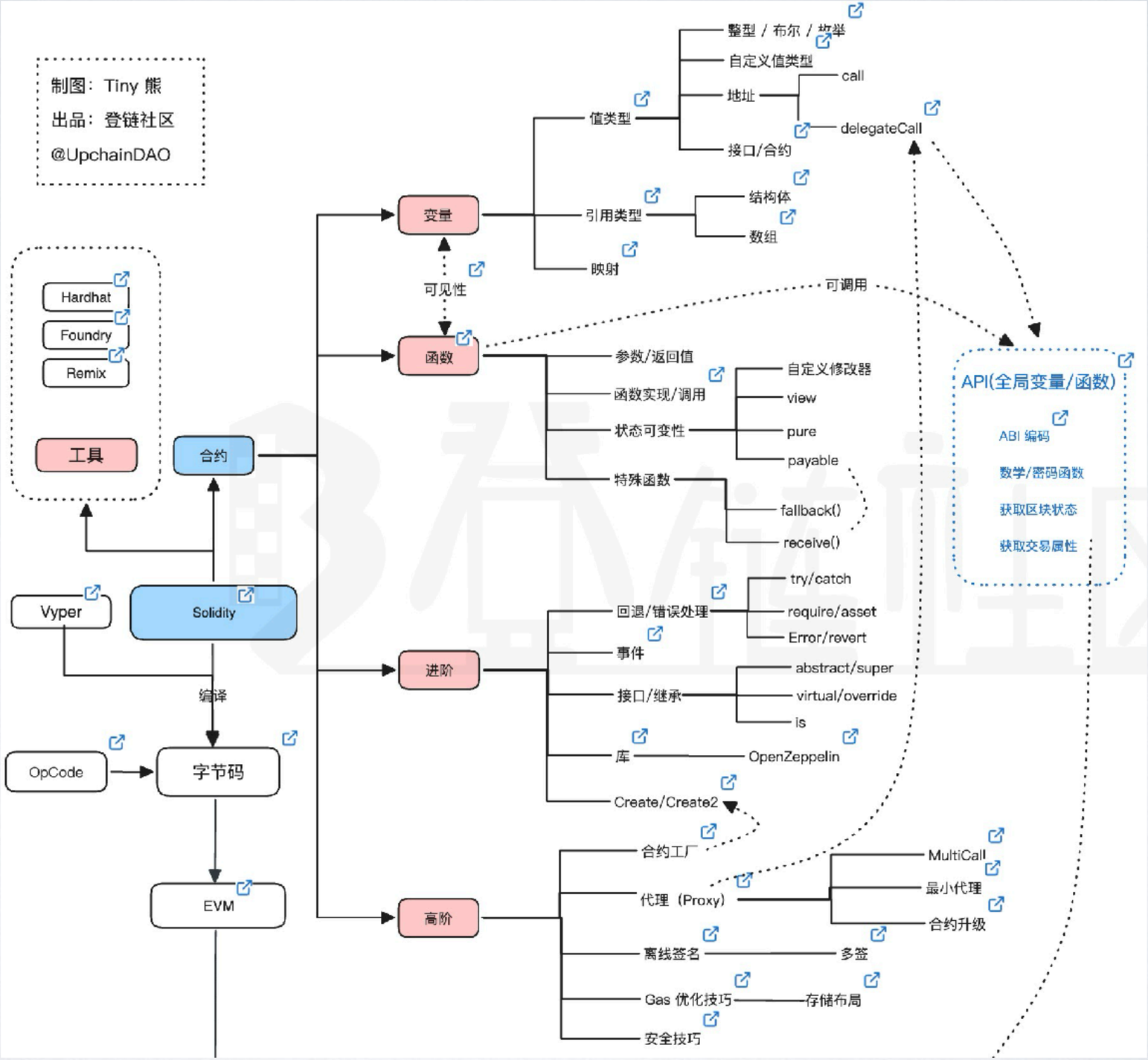
常见问题

- 不用保存 total balance
- 不用同时保存 address[3] 和 uint[3] (复用mapping的数据)
- 不需要保存所有的数组数据
- 排序代码复用 （在 receive 及 deposit 中）
- 排序代码放 View 函数可以么?

DAY4： SOLIDITY

登链社区 - Tiny熊

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

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 “_;”

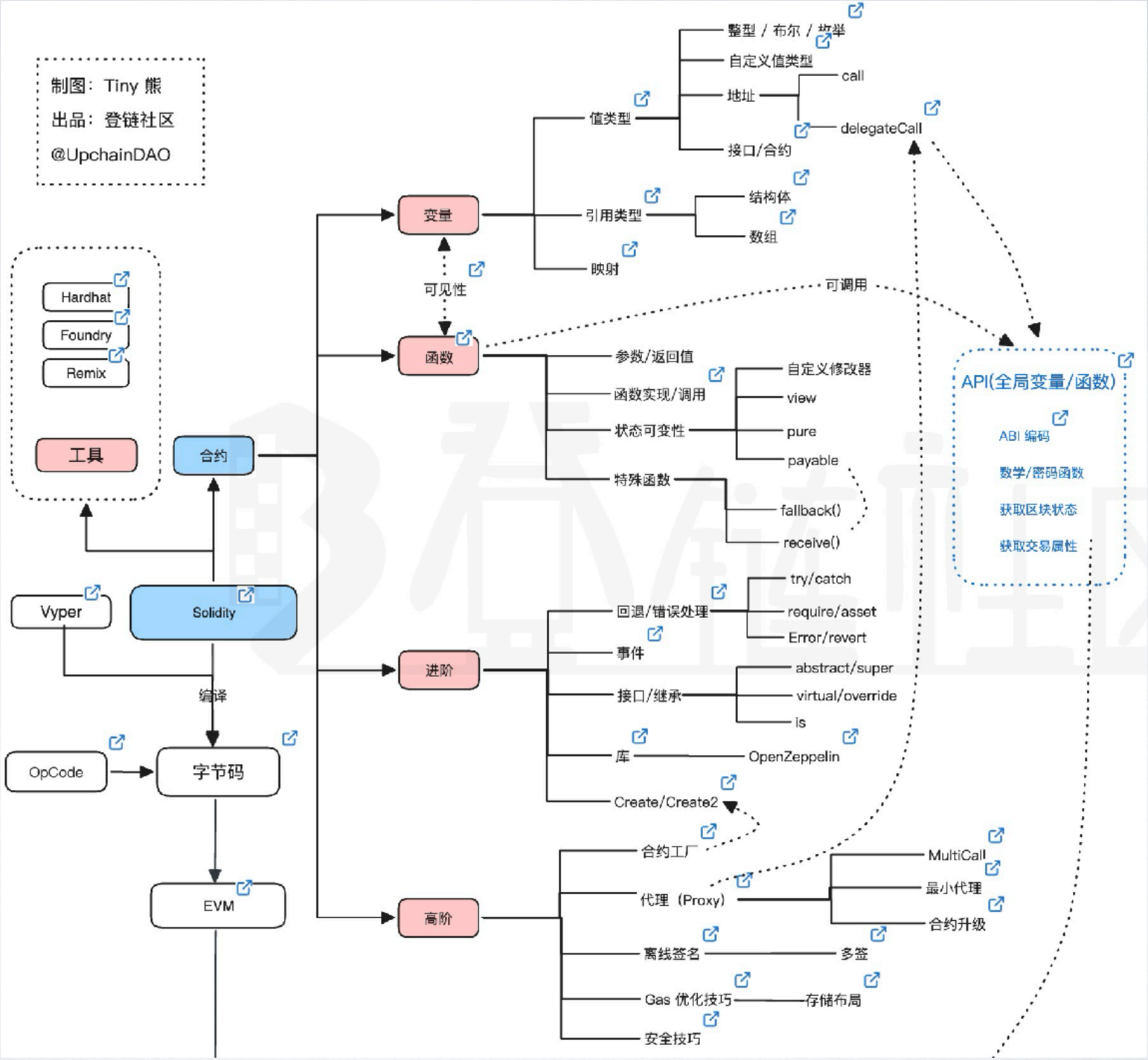
函数修改器带参数

```
modifier over22(uint age) {
    require (age >= 22, "too small age");
    _;
}


 function marry(uint age) public over22(age) {
   // do something
}


 function marry(uint age) public payable over22(age) {
   // do something
 }
```

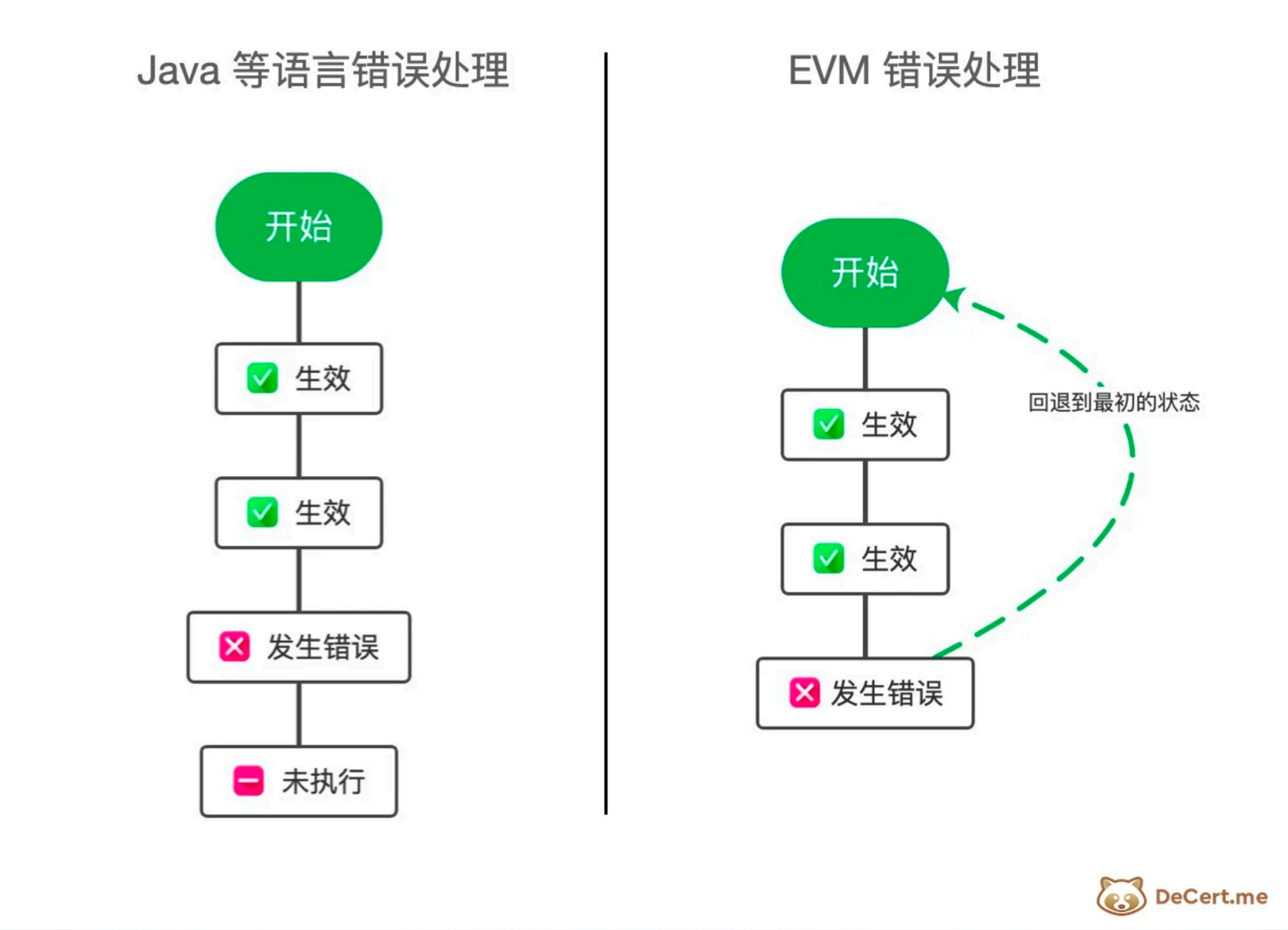
函数修改器 – 还应该了解

- 函数修改器是一个语法糖（编译期间扩展到函数的实现）
  - 可以用私有函数实现类似效果

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

EVM 错误处理


Java 等语言错误处理
开始
生效
生效
发生错误
未执行
EVM 错误处理
开始
生效
生效
发生错误
回退到最初的状态
DeCert.me

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
  - assert(): Panic ， 应对错误的代码
- revert(“msg”) ： 终止运行并撤销状态更改
- Error 定义错误 、 revert Error()

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

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

testError.sol

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
  - 例外：
    - 无法捕获对不存在的合约调用（对一个不存在的合约的调用会成功）
    - 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 – 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 – 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

testls.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
- overide： 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 Admin 合约，Admin 调用 BigBank 的 withdraw().

https://decert.me/quests/063c14be-d3e6-41e0-a243-54e35b1dde58