Solidity编程的高级话题

郑嘉文 2020/09

前 言

这里自由添加文字

目录

CONTENTS

LOGO

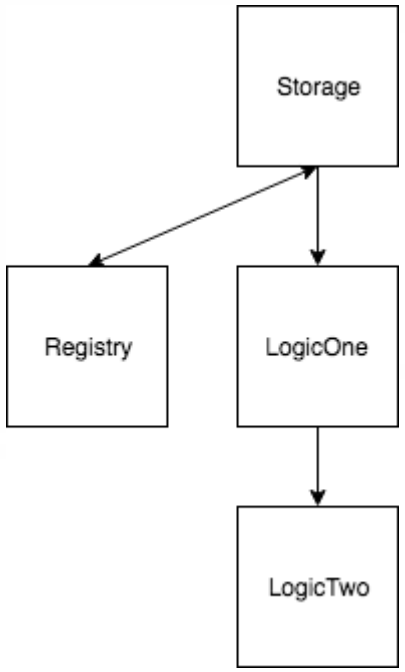
Solidity合约的继承升级

策略	优点	缺点
Proxy合约	升级合约不需要重新设计	- Proxy合约代码并不反映它存储合约的状态 - 不能改变现有目标合约的域但是可以增加新的域
分隔逻辑和数据	数据可以从数据合约里读取 数据合约里的数据结构可以被修改和添加	- 合约需要分拆成数据合约和程序逻辑合约 - 复杂的数据类型，比如struct可以被修改，但是修改起来比较复杂
用键值对来区隔逻辑和数据	键值对更为通用和简单 数据合约里的数据结构可以被修改和添加	- 合约需要分拆成数据合约和程序逻辑合约 - 访问数据是非常抽象，因为访问是都是通过键值对来访问的
部分可升级的合约系统	合约里简单的部分可以不变以获取信任	不可升级的合约代码就永远不能升级了

LOGO

Solidity合约的通用代理模式

- 可升级的合约
- 节省GAS
- 汇编代码
- 合约间调用
- ABI编程
- 智能合约工作机制




```
function (){
        assembly{
            let g := and(gas,0xEFFFFFFF)
            let o_code := mload(0x40) //获得内存空闲指针
            let addr := and(sload(0),0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)
            calldatacopy(o_code, 0, calldatasize)

            let retval := call(g
                    , addr //目标合约地址
                    , 0 //value
                    , o_code //calldata
                    , calldatasize //calldata大小
                    , o_code //返回地址
                    , 32) //32字节的返回地址

            // Check return value
            // 0 == 如果返回值为0，就跳到bad destination (02)
            jumpi(0x02,iszero(retval))
            return(o_code,32)
        }
}
```

LOGO

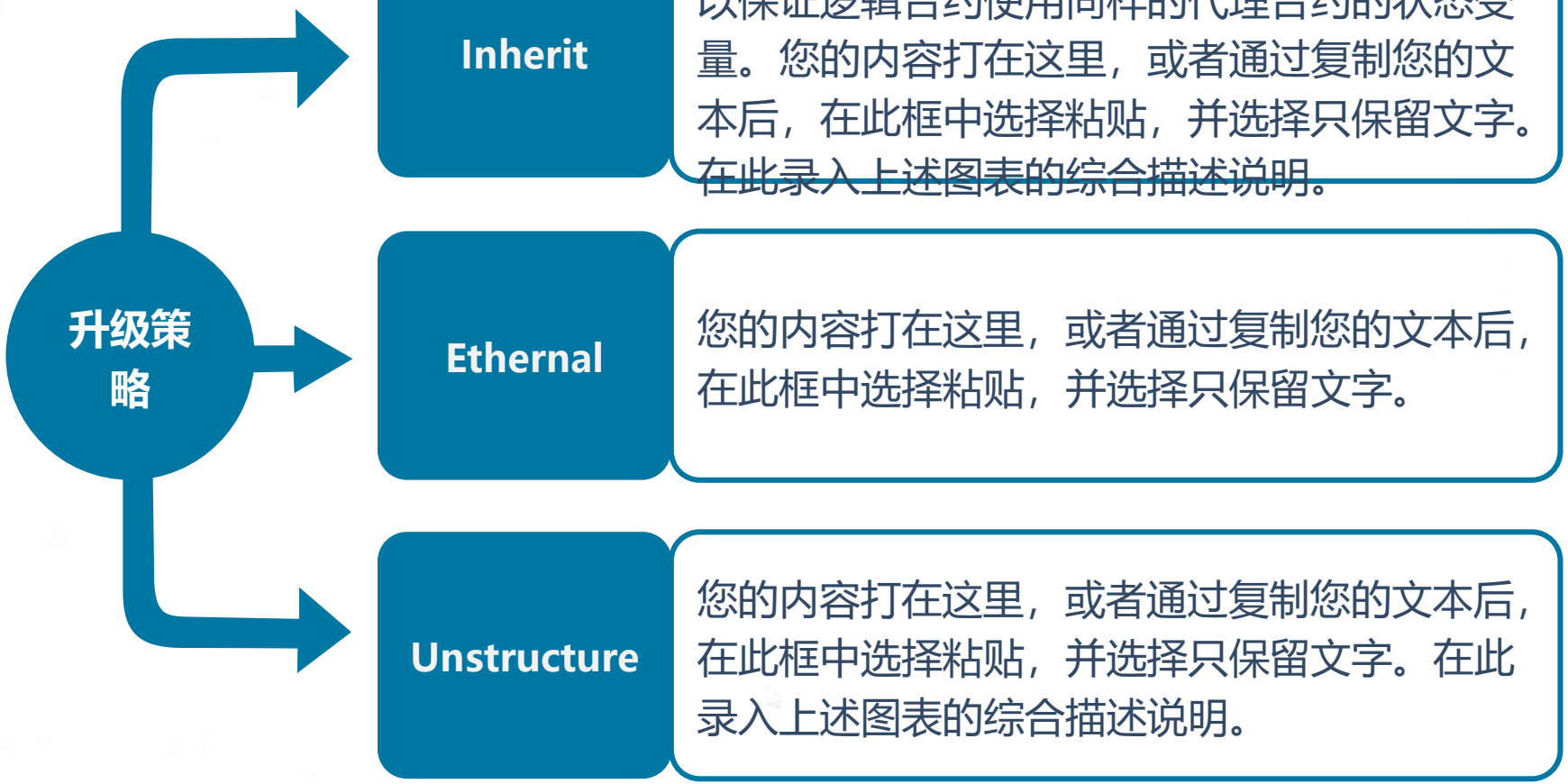
Solidity合约的通用代理模式


```
function (){
    assembly {
        let ptr := mload(0x40)
        calldatacopy(ptr, 0, calldatasize)
        let result := delegatecall(gas, _impl, ptr, calldatasize, 0, 0)
        let size := returndatasize
        returndatacopy(ptr, 0, size)

        switch result {
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
}
```

LOGO

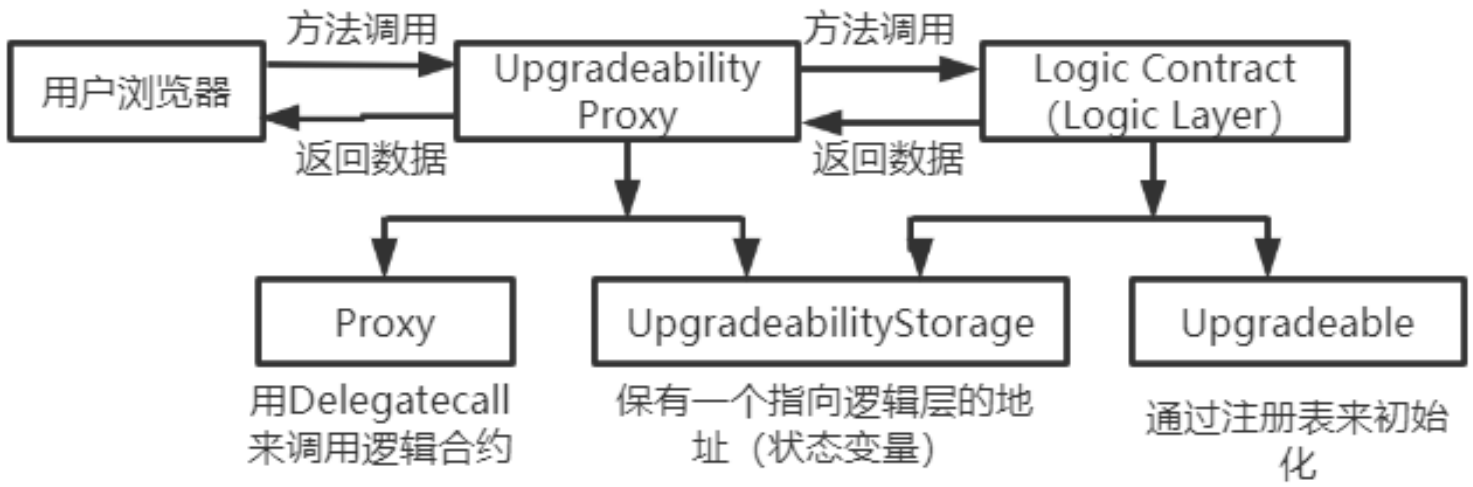
Solidity合约数据的3种升级方式



为了升级，如何让逻辑实现合约不覆盖在代理合约里使用的状态变量

LOGO

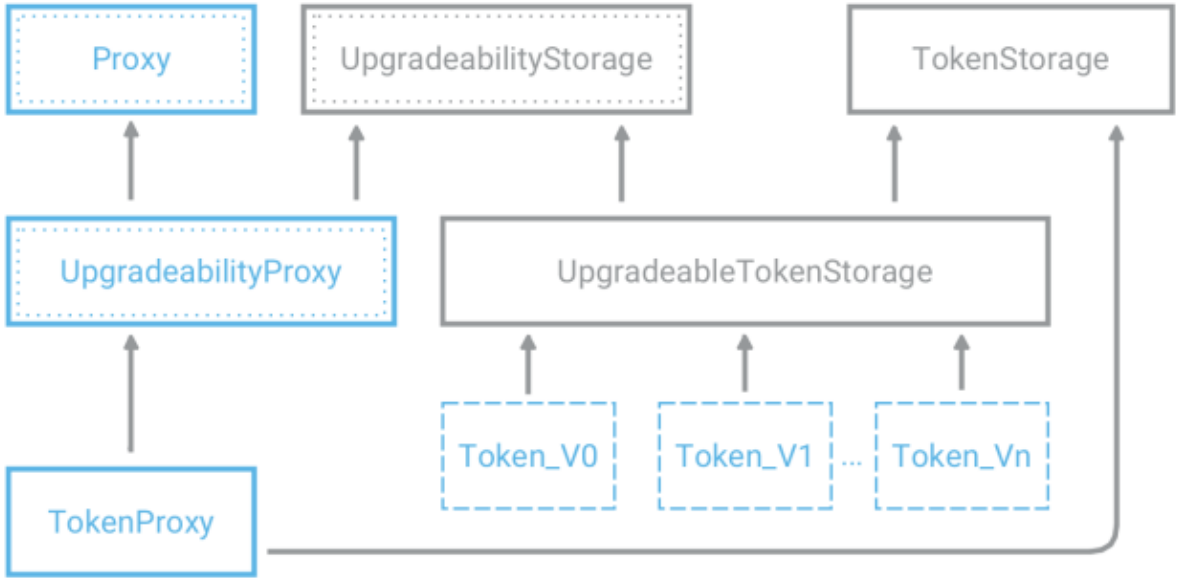
Solidity合约数据的继承升级



- 代理Method call的代码定义在Proxy合约里
- 使用的状态变量（State Variable）定义在UpgradeabilityStorage里
- 因为Upgradeability Proxy合约和最初的逻辑实现合约都继承自UpgradeabilityStorage合约，所有升级的合约都必须继承自最初的逻辑实现合约，升级后的合约都会使用同样的状态变量（State Variable）

LOGO

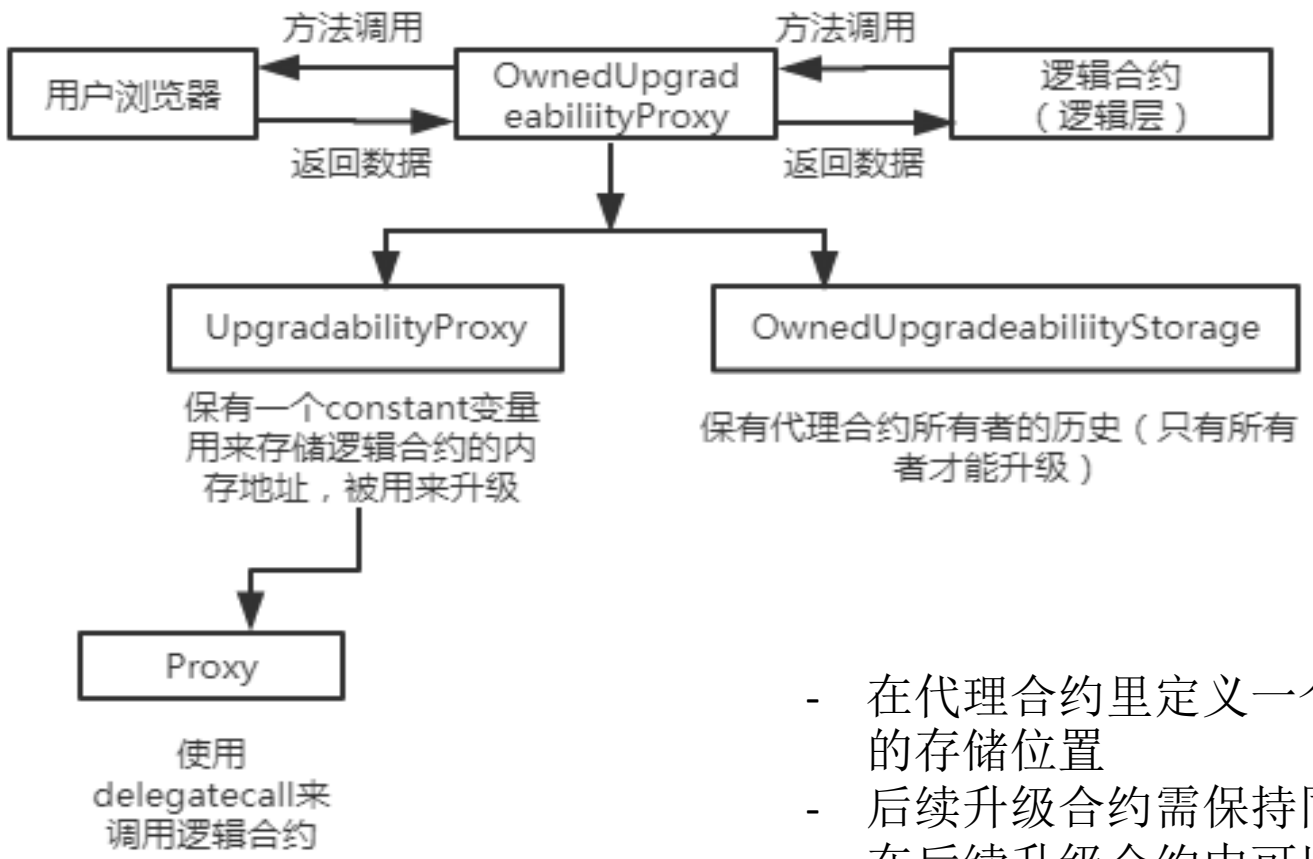
Solidity合约数据的永久升级



- 存储架构定义在一个独立的存储合约里，代理合约和逻辑实现合约都继承它
- 使用的状态变量（State Variable）定义在存储合约里
- 所有以后的升级的合约都不能再定义新的状态变量（State Variable）

LOGO

Solidity合约数据的非结构化升级

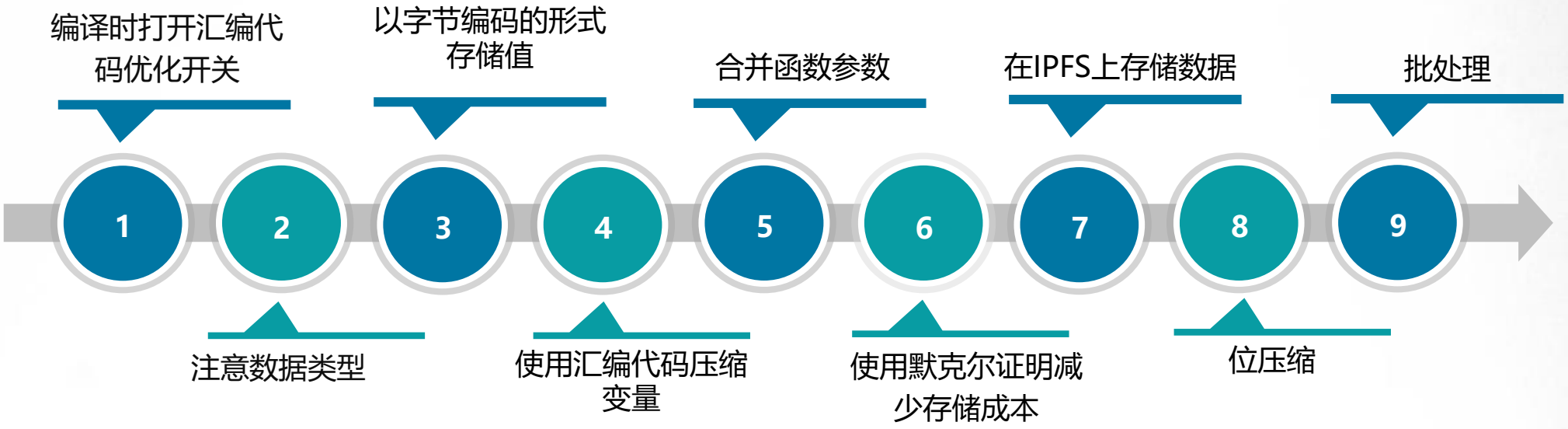


- 在代理合约里定义一个常量，作为逻辑实现合约的存储位置
- 后续升级合约需保持同样的存储结构
- 在后续升级合约中可以自由的定义新的变量

LOGO

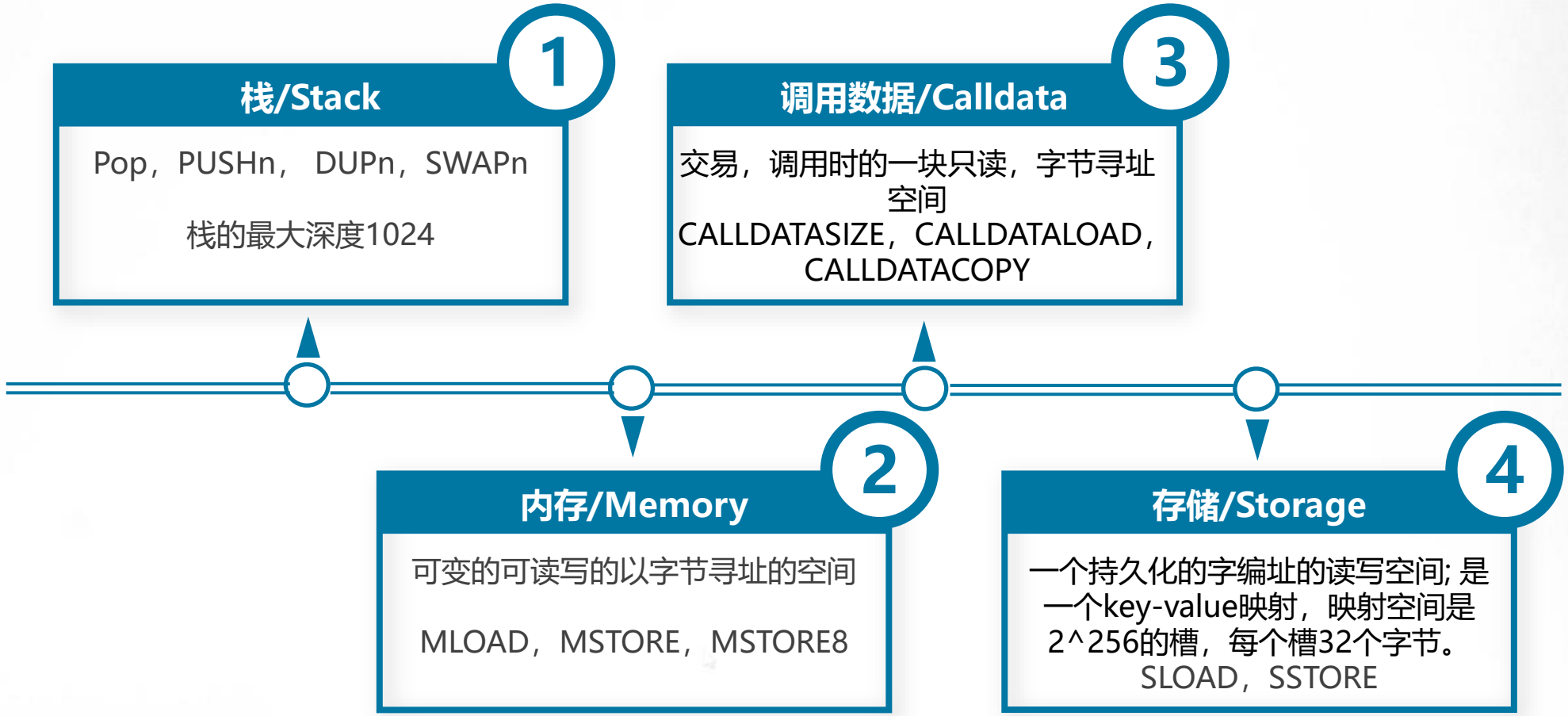
Solidity合约编程节省GAS的方法

- 可升级的合约
- 节省GAS
- 汇编代码
- 合约间调用
- ABI编程
- 智能合约工作机制



LOGO

汇编指令



LOGO

合约间调用的两种主要方式

call

执行其他合约的代码。

VS

delegatecall

执行其他合约的代码，
使用调用合约的存储。
保留调用合约的
msg.sender 和
msg.value

合约间调用必须要检查返回值。自从2017年10月17日的拜占庭的硬分叉以后，可以使用resultdatasize和resultdatacopy这两个新指令来拷贝call/delegatecall的返回值到内存。而此前，只能使用汇编代码获取返回值

LOGO

EVM里的数据表示

固定长度类型

除了Mapping和动态长度类型以外都是固定长度类型，比如 Struct等

- 固定长度类型存在位置为0开始的存储槽，连续存放。
- 初始化值为0。只有在存储值非零的情况下，用户才需要付费

映射

Mapping

- 存储位置slot(p)没有用
- 键为K的映射的值存放在keccak256(k,p)
- keccak256(k,p)函数参数k和p必须32字节对齐

LOGO

EVM里的数据表示

数组

动态数组

- 存储方式和映射一样
- 数组访问更严格，更复杂。提供数组长度以及数组边界检查
- 数组元素都会被对齐到32字节。
- 比映射更贵

特殊情况

字节数组bytes，string

- 如果字节数组长度小于31字节

这种情况下，只占用1个存储槽。其他和字符串数组一样。

- 如果字节数组长度大于等于31字节

字节数组就跟[]byte一样。数组元素的地址计算方式同字符串和映射

LOGO

Calldata里的数据表示

固定大小的数据类型

uint256

>>> encode_abi(["uint256", "uint256", "uint256"],[0xA, 0xB, 0xC]).hex()

'0000000000000000000000000000000000000000000000000000000000000a0000000000000000000000000000000000000000000000000000000000000b0000000000000000000000000000000000000000000000000000000000000c'

重新格式化:

0000000000000000000000000000000000000000000000000000000000000a
0000000000000000000000000000000000000000000000000000000000000b
0000000000000000000000000000000000000000000000000000000000000c

LOGO

Calldata里的数据表示

固定大小的数据类型

不同类型的固定长度数据类型

```
>>> encode_abi(["int8", "uint32", "uint64"],[0xA, 0xB, 0xC]).hex()

'0000000000000000000000000000000000000000000000000000000000000a0000000000000000000000000000000000000000000000000000000000000b0000000000000000000000000000000000000000000000000000000000000c'
```

重新格式化：

```
0000000000000000000000000000000000000000000000000000000000000a
0000000000000000000000000000000000000000000000000000000000000b
0000000000000000000000000000000000000000000000000000000000000c
```

LOGO

Calldata里的数据表示

固定大小的数据类型

固定大小的数据

```
>>> encode_abi(["int8[3]", "int256[3]"],[[0xA, 0xB, 0xC], [0xD, 0xE,
0xF]]).hex()

'0000000000000000000000000000000000000000000000000000000000000
0a0000000000000000000000000000000000000000000000000000000000000
000b0000000000000000000000000000000000000000000000000000000000000
00000c0000000000000000000000000000000000000000000000000000000000000
0000000d0000000000000000000000000000000000000000000000000000000000
000000000e0000000000000000000000000000000000000000000000000000000
00000000000f'
```

重新格式化：

```
0000000000000000000000000000000000000000000000000000000000000
0a
0000000000000000000000000000000000000000000000000000000000000
0b
0000000000000000000000000000000000000000000000000000000000000
0c
0000000000000000000000000000000000000000000000000000000000000
0d
0000000000000000000000000000000000000000000000000000000000000
0e
0000000000000000000000000000000000000000000000000000000000000
0f
```

LOGO

Calldata里的数据表示

- 可升级的合约
- 节省GAS
- 汇编代码
- 合约间调用
- ABI编程
- 智能合约工作机制

动态的数据类型

采用Head-Tail模型

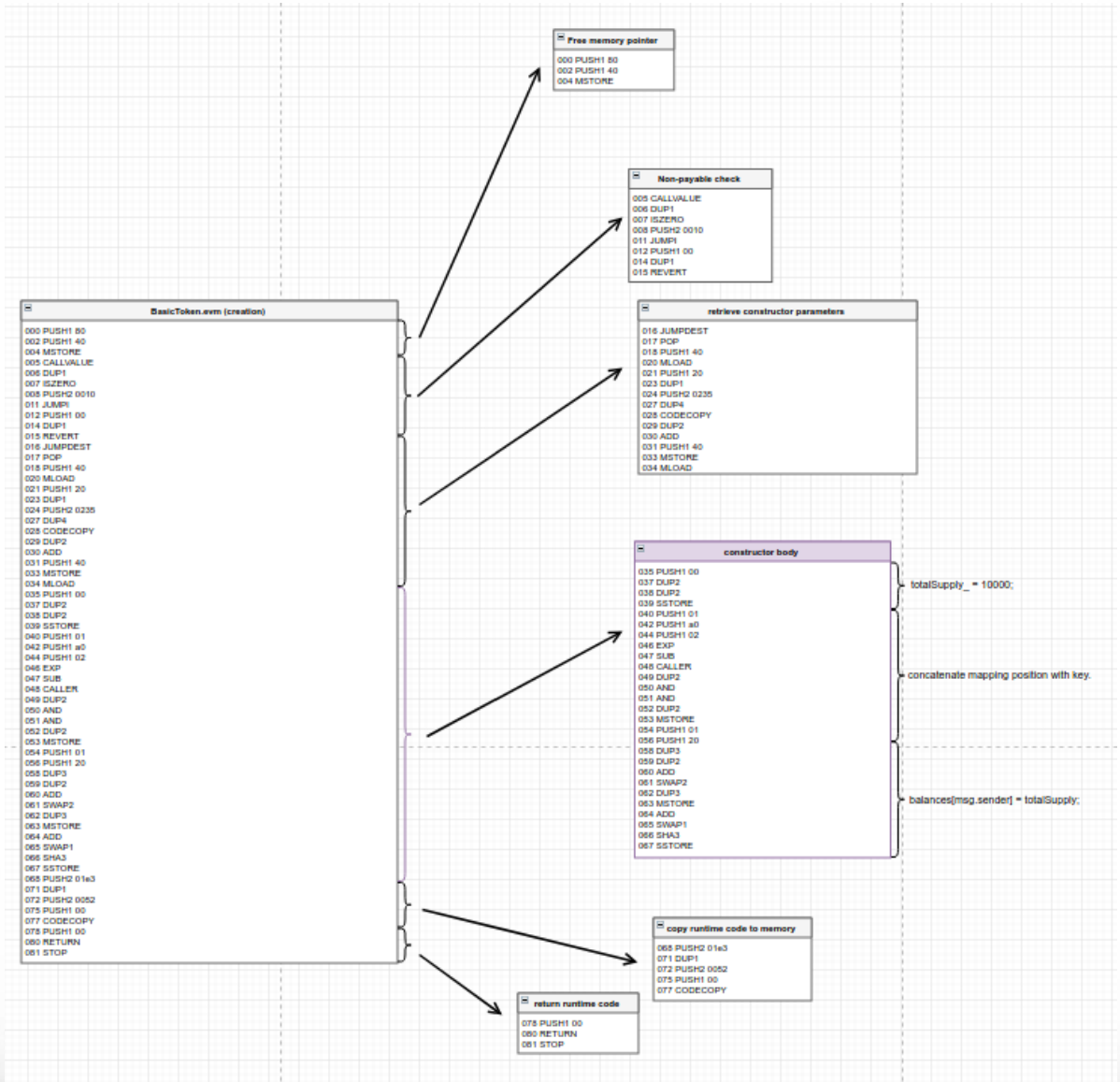
```
>>> encode_abi(["uint256[]", "uint256[]", "uint256[]"],[[0xa1, 0xa2, 0xa3],
[0xb1, 0xb2, 0xb3], [0xc1, 0xc2, 0xc3]]).hex()

'0000000000000000000000000000000000000000000000000000000000000
060000000000000000000000000000000000000000000000000000000000000
000e0000000000000000000000000000000000000000000000000000000000
0000160000000000000000000000000000000000000000000000000000000
0000000030000000000000000000000000000000000000000000000000000
000000000a1000000000000000000000000000000000000000000000000000
0000000000a2000000000000000000000000000000000000000000000000000
00000000000a3000000000000000000000000000000000000000000000000
0000000000000030000000000000000000000000000000000000000000000
000000000000000b1000000000000000000000000000000000000000000000
0000000000000000b2000000000000000000000000000000000000000000
00000000000000000b3000000000000000000000000000000000000000000
0000000000000000000030000000000000000000000000000000000000000
000000000000000000000c1000000000000000000000000000000000000
0000000000000000000000c2000000000000000000000000000000000000
00000000000000000000000c3'
```

简单的例子程序

```
pragma solidity ^0.4.24;
contract SampleERC20{
  uint256 _totalSupply;
  mapping(address => uint256) balances;
  constructor(uint256 _initialSupply) public {
    _totalSupply = _initialSupply;
    balances[msg.sender] = _initialSupply;
  }
  function totalSupply() public view returns (uint256) {
    return _totalSupply;
  }
  function balanceOf(address _owner) public view returns (uint256) {
    return balances[_owner];
  }
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);
    balances[msg.sender] = balances[msg.sender] - _value;
    balances[_to] = balances[_to] + _value;
    return true;
  }
}
```

汇编程序分解


Free memory pointer
000 PUSH1 80
002 PUSH1 40
004 MSTORE
Non-payable check
005 CALLVALUE
006 DUP1
007 ISZERO
008 PUSH2 0010
011 JUMPI
012 PUSH1 00
014 DUP1
015 REVERT
BasicToken.evm (creation)
000 PUSH1 80
002 PUSH1 40
004 MSTORE
005 CALLVALUE
006 DUP1
007 ISZERO
008 PUSH2 0010
011 JUMPI
012 PUSH1 00
014 DUP1
015 REVERT
016 JUMPDEST
017 POP
018 PUSH1 40
020 MLOAD
021 PUSH1 20
023 DUP1
024 PUSH2 0235
027 DUP4
028 CODECOPY
029 DUP2
030 ADD
031 PUSH1 40
033 MSTORE
034 MLOAD
035 PUSH1 00
037 DUP2
038 DUP2
039 SSTORE
040 PUSH1 01
042 PUSH1 a0
044 PUSH1 02
046 EXP
047 SUB
048 CALLER
049 DUP2
050 AND
051 AND
052 DUP2
053 MSTORE
054 PUSH1 01
056 PUSH1 20
058 DUP3
059 DUP2
060 ADD
061 SWAP2
062 DUP3
063 MSTORE
064 ADD
065 SWAP1
066 SHA3
067 SSTORE
068 PUSH2 01e3
071 DUP1
072 PUSH2 0052
075 PUSH1 00
077 CODECOPY
078 PUSH1 00
080 RETURN
081 STOP
retrieve constructor parameters
016 JUMPDEST
017 POP
018 PUSH1 40
020 MLOAD
021 PUSH1 20
023 DUP1
024 PUSH2 0235
027 DUP4
028 CODECOPY
029 DUP2
030 ADD
031 PUSH1 40
033 MSTORE
034 MLOAD
constructor body
035 PUSH1 00
037 DUP2
038 DUP2
039 SSTORE
040 PUSH1 01
042 PUSH1 a0
044 PUSH1 02
046 EXP
047 SUB
048 CALLER
049 DUP2
050 AND
051 AND
052 DUP2
053 MSTORE
054 PUSH1 01
056 PUSH1 20
058 DUP3
059 DUP2
060 ADD
061 SWAP2
062 DUP3
063 MSTORE
064 ADD
065 SWAP1
066 SHA3
067 SSTORE
totalSupply_ = 10000;
concatenate mapping position with key.
balances[msg.sender] = totalSupply;
copy runtime code to memory
068 PUSH2 01e3
071 DUP1
072 PUSH2 0052
075 PUSH1 00
077 CODECOPY
return runtime code
078 PUSH1 00
080 RETURN
081 STOP

汇编程序分解

Free memory pointer

```
000 PUSH1 80
002 PUSH1 40
004 MSTORE
```

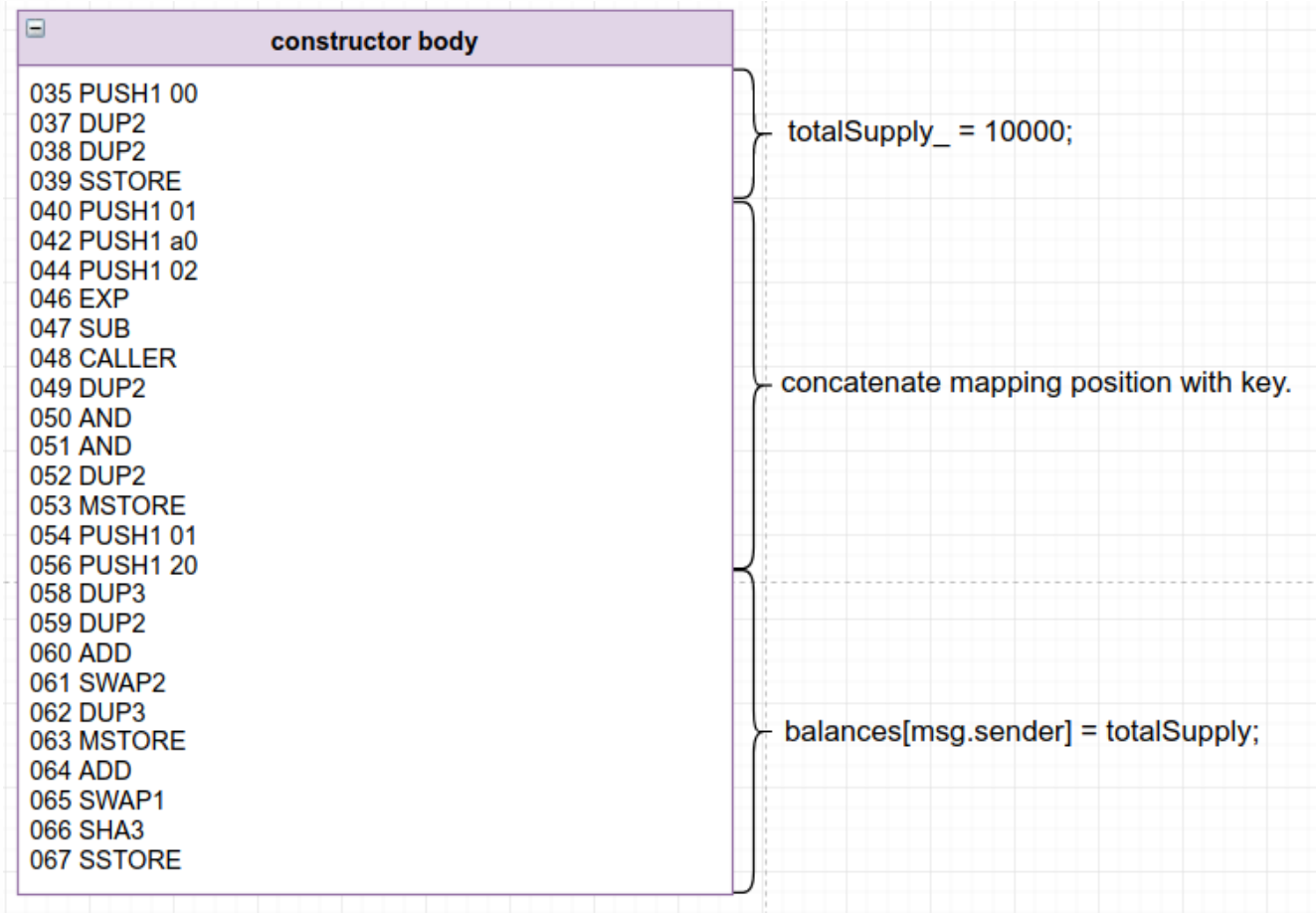
Non-payable check

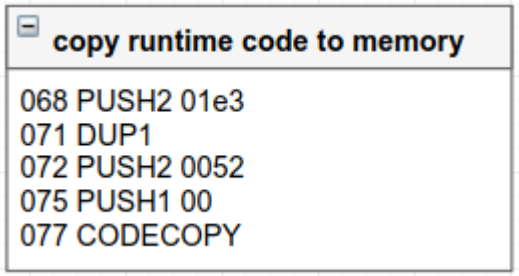
```
005 CALLVALUE
006 DUP1
007 ISZERO
008 PUSH2 0010
011 JUMPI
012 PUSH1 00
014 DUP1
015 REVERT
```

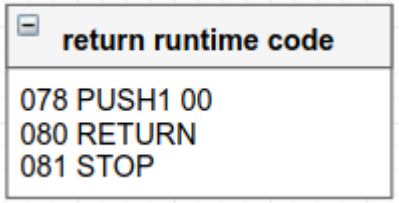
retrieve constructor parameters

```
016 JUMPDEST
017 POP
018 PUSH1 40
020 MLOAD
021 PUSH1 20
023 DUP1
024 PUSH2 0235
027 DUP4
028 CODECOPY
029 DUP2
030 ADD
031 PUSH1 40
033 MSTORE
034 MLOAD
```

汇编程序分解


constructor body
035 PUSH1 00
037 DUP2
038 DUP2
039 SSTORE
040 PUSH1 01
042 PUSH1 a0
044 PUSH1 02
046 EXP
047 SUB
048 CALLER
049 DUP2
050 AND
051 AND
052 DUP2
053 MSTORE
054 PUSH1 01
056 PUSH1 20
058 DUP3
059 DUP2
060 ADD
061 SWAP2
062 DUP3
063 MSTORE
064 ADD
065 SWAP1
066 SHA3
067 SSTORE
totalSupply_ = 10000;
concatenate mapping position with key.
balances[msg.sender] = totalSupply;


copy runtime code to memory
068 PUSH2 01e3
071 DUP1
072 PUSH2 0052
075 PUSH1 00
077 CODECOPY


return runtime code
078 PUSH1 00
080 RETURN
081 STOP

汇编程序 — 函数选择子

Function selector

```
013 PUSH4 ffffffff
018 PUSH29 0100000000000000000000000000000000000000000000000000000000
048 PUSH1 00
050 CALLDATALOAD
051 DIV
052 AND
053 PUSH4 18160ddd
058 DUP2
059 EQ
060 PUSH2 005b
063 JUMPI
064 DUP1
065 PUSH4 70a08231
070 EQ
071 PUSH2 0082
074 JUMPI
075 DUP1
076 PUSH4 a9059cbb
081 EQ
082 PUSH2 00b0
085 JUMPI
086 JUMPDEST
087 PUSH1 00
089 DUP1
090 REVERT
```

Extract first four bytes of calldata (function signature hash).

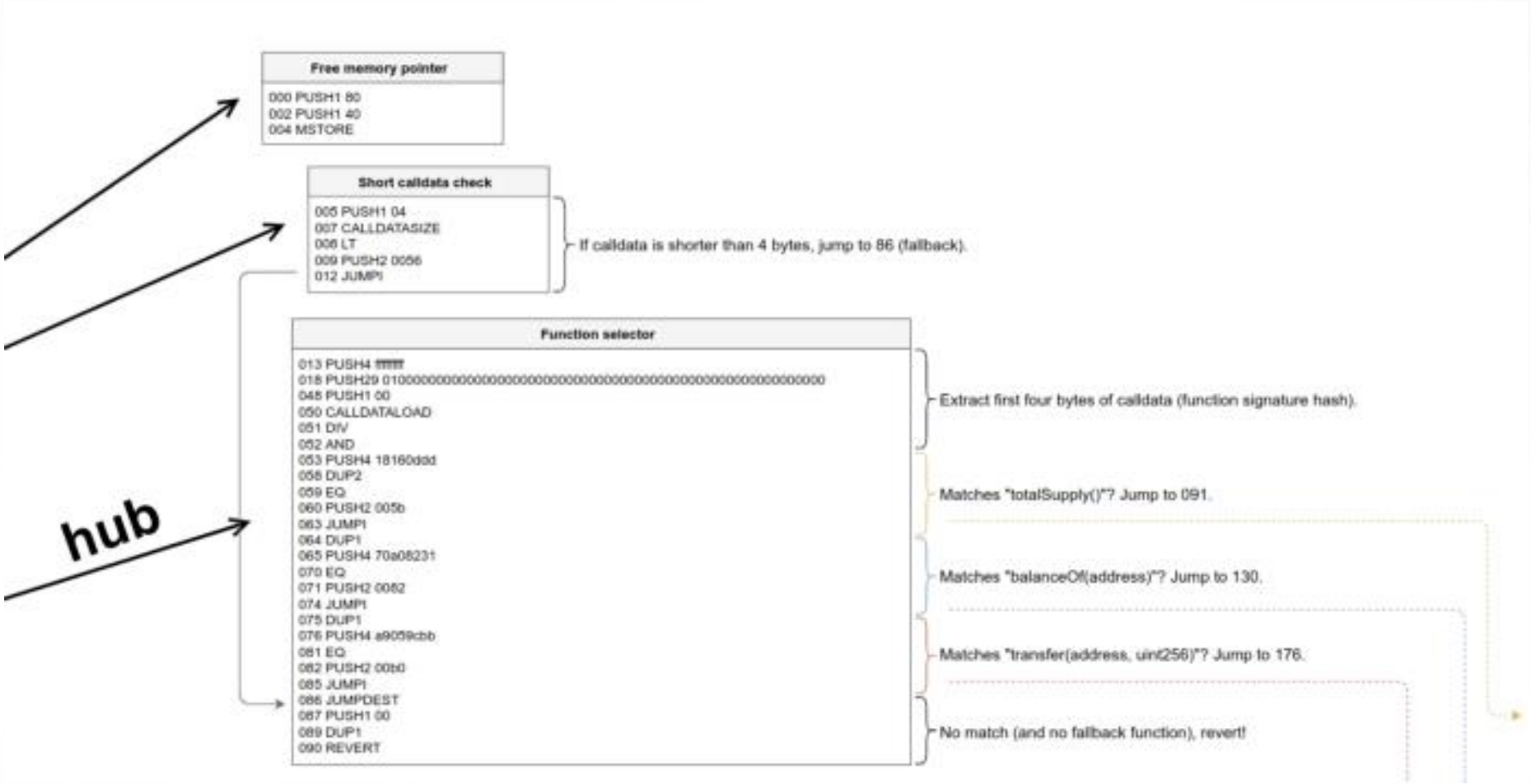
Matches "totalSupply()"? Jump to 091.

Matches "balanceOf(address)"? Jump to 130.

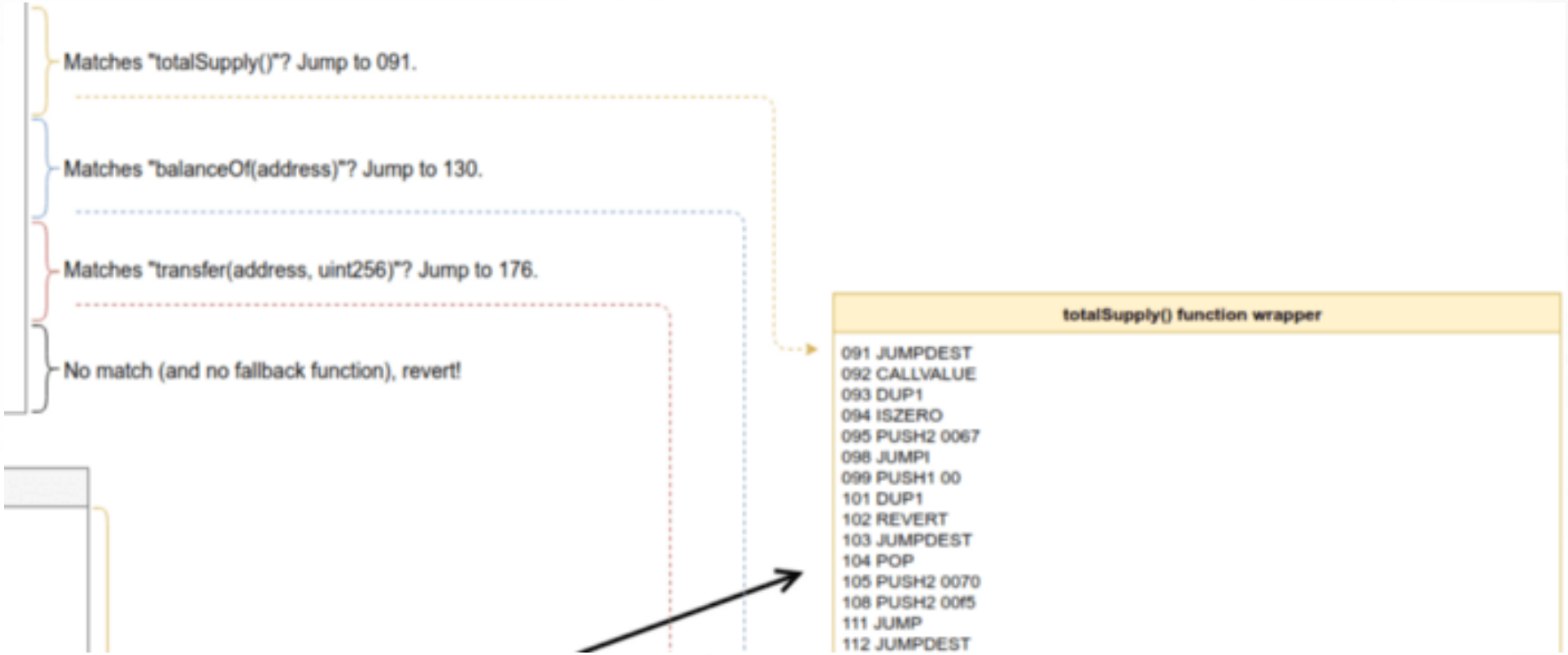
Matches "transfer(address, uint256)"? Jump to 176.

No match (and no fallback function), revert!

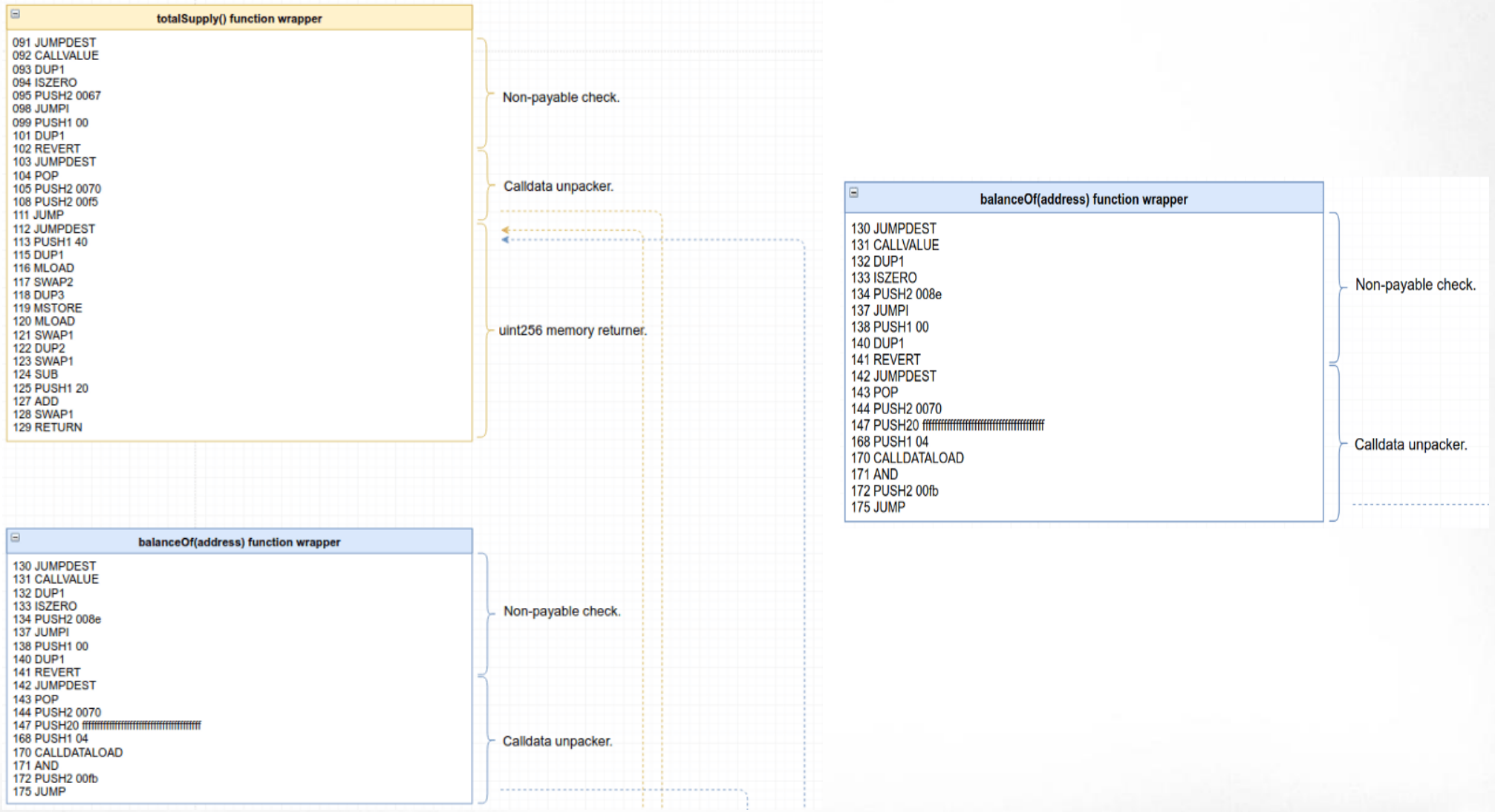
汇编程序 — 函数选择子


Free memory pointer
000 PUSH1 80
002 PUSH1 40
004 MSTORE
Short calldata check
005 PUSH1 04
007 CALLDATASIZE
008 LT
009 PUSH2 0056
012 JUMPI
If calldata is shorter than 4 bytes, jump to 86 (fallback).
Function selector
013 PUSH4 ffffffff
018 PUSH29 0100000000000000000000000000000000000000000000000000000000
048 PUSH1 00
050 CALLDATALOAD
051 DIV
052 AND
053 PUSH4 18160ddd
058 DUP2
059 EQ
060 PUSH2 005b
063 JUMPI
064 DUP1
065 PUSH4 70a08231
070 EQ
071 PUSH2 0082
074 JUMPI
075 DUP1
076 PUSH4 a9059cbb
081 EQ
082 PUSH2 00b0
085 JUMPI
086 JUMPDEST
087 PUSH1 00
089 DUP1
090 REVERT
Extract first four bytes of calldata (function signature hash).
Matches "totalSupply()"? Jump to 091.
Matches "balanceOf(address)"? Jump to 130.
Matches "transfer(address, uint256)"? Jump to 176.
No match (and no fallback function), revert!
hub

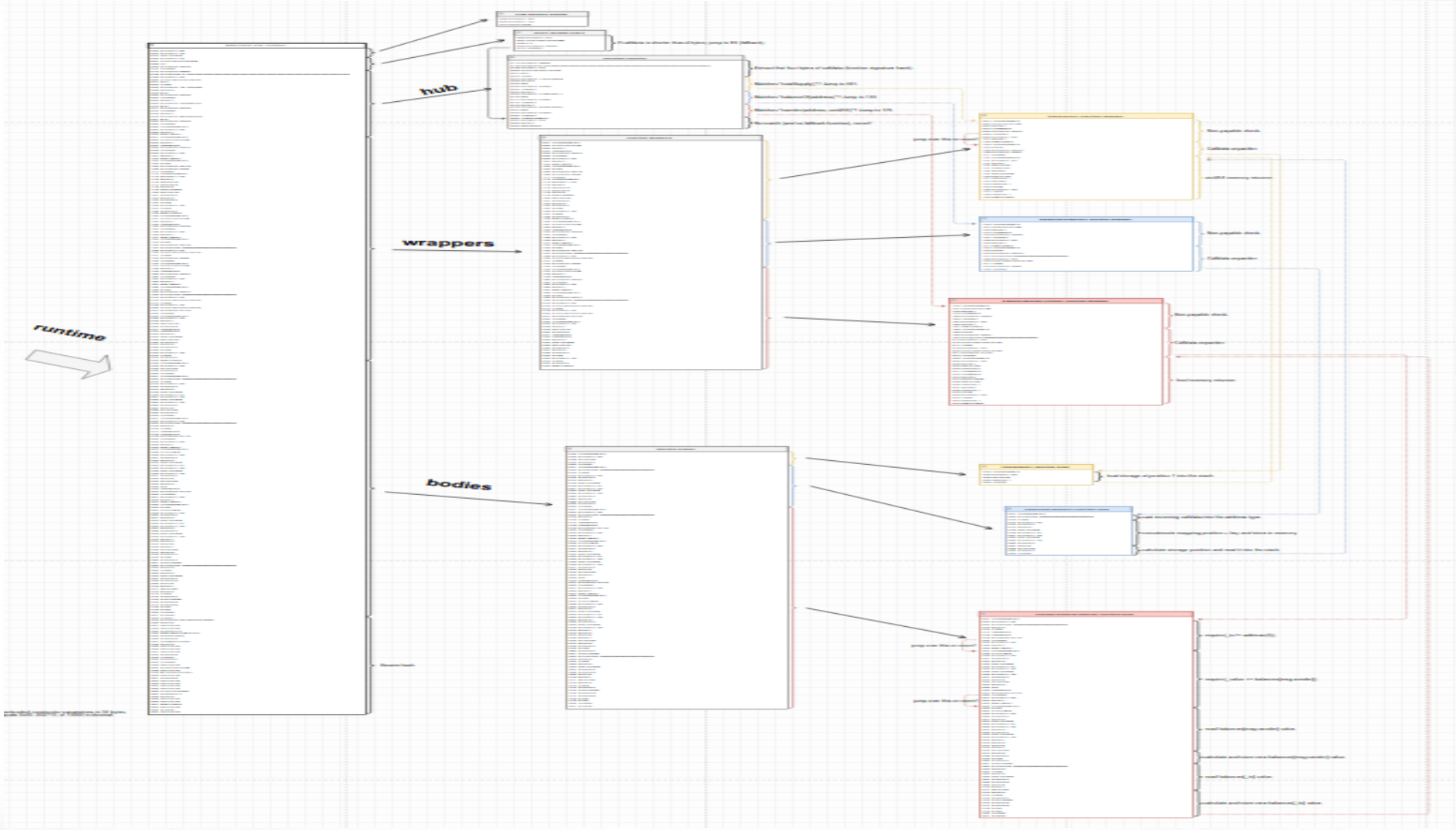
汇编程序 — 函数Wrapper



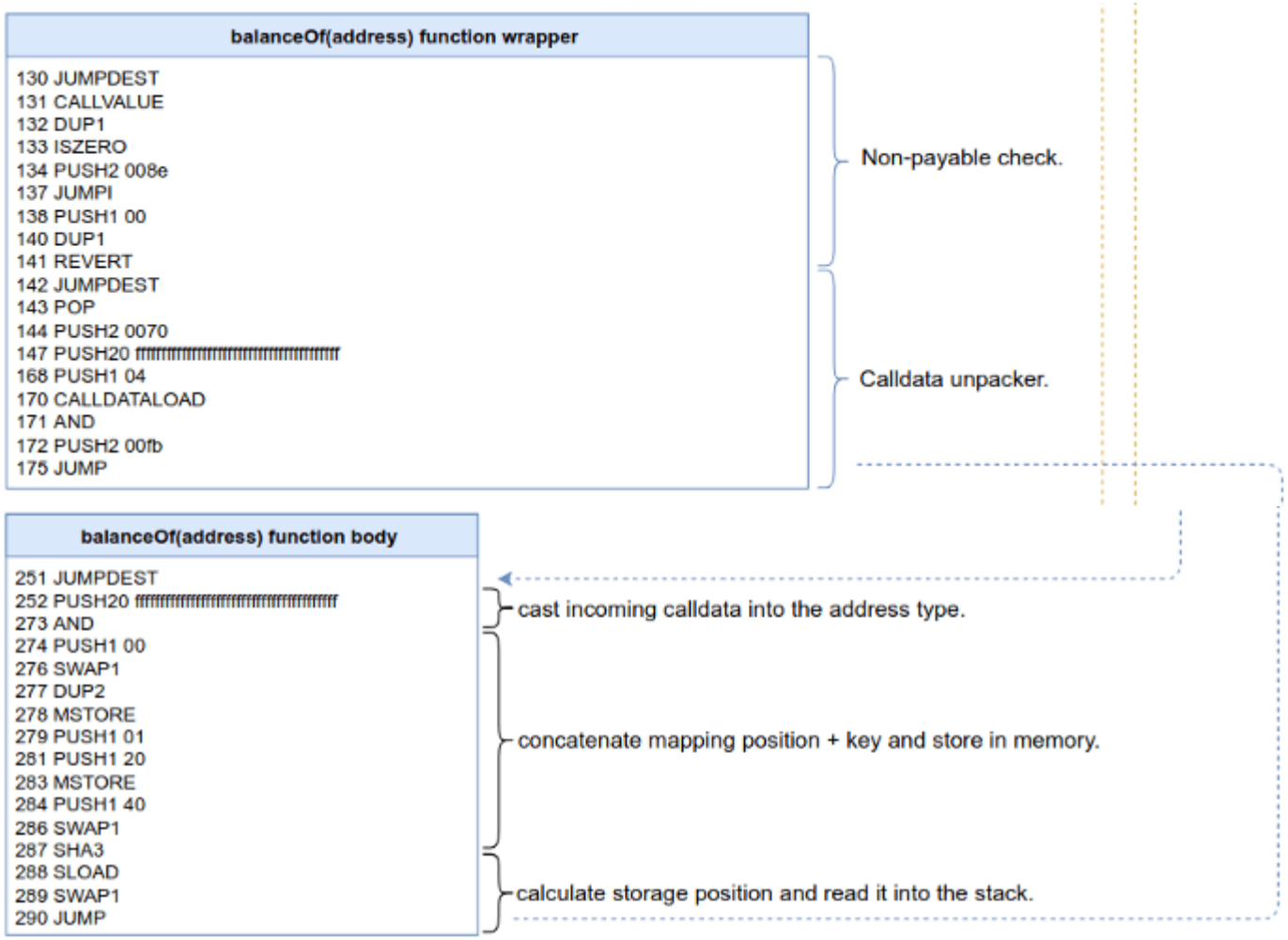
汇编程序 — 函数Wrapper



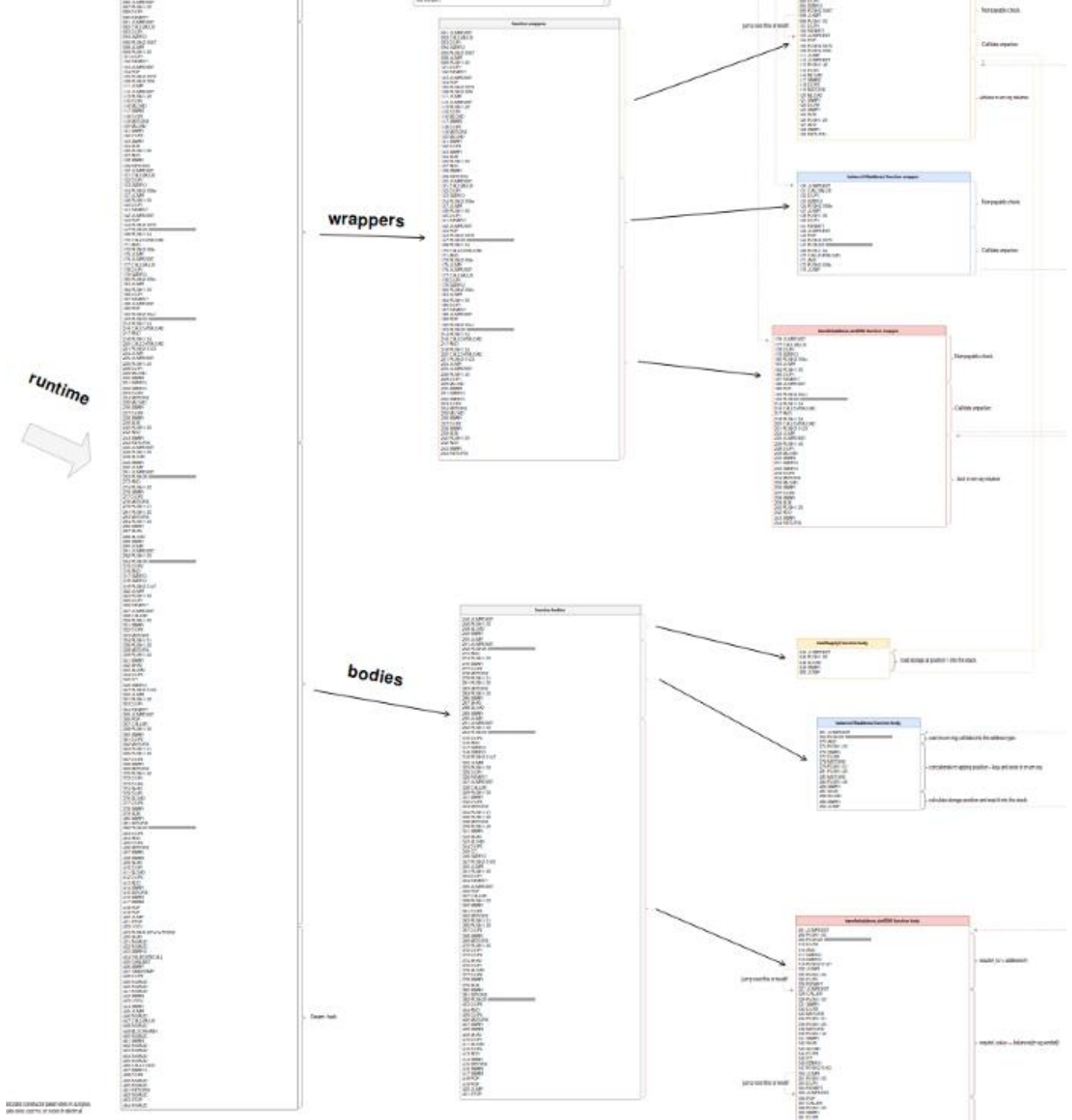
汇编程序 — 总体调用图



汇编程序 — 总体调用图


balanceOf(address) function wrapper
130 JUMPDEST
131 CALLVALUE
132 DUP1
133 ISZERO
134 PUSH2 008e
137 JUMPI
138 PUSH1 00
140 DUP1
141 REVERT
142 JUMPDEST
143 POP
144 PUSH2 0070
147 PUSH20 ffffffffffffffffffffffffffffffffffffffff
168 PUSH1 04
170 CALLDATALOAD
171 AND
172 PUSH2 00fb
175 JUMP
Non-payable check.
Calldata unpacker.
balanceOf(address) function body
251 JUMPDEST
252 PUSH20 ffffffffffffffffffffffffffffffffffffffff
273 AND
274 PUSH1 00
276 SWAP1
277 DUP2
278 MSTORE
279 PUSH1 01
281 PUSH1 20
283 MSTORE
284 PUSH1 40
286 SWAP1
287 SHA3
288 SLOAD
289 SWAP1
290 JUMP
cast incoming calldata into the address type.
concatenate mapping position + key and store in memory.
calculate storage position and read it into the stack.

合约总体调用图



既见君子，云胡不喜

电话 **13240946967**
邮箱 zy731@hotmail.com
微信 **gavinzheng731**
博客**https://my.oschina.net/gavinzheng731/**

谢谢聆听！

郑嘉文

Q&A