区块链中的智能合约

邹亮
360区块链实验室高级研究员、虚拟机专家

SPEAKER

INTRODUCE

邹亮 Software Engineer

- 2015 年博士毕业，计算机软件理论方向
- 多年编译器工作经验
- 参加智能合约语言 Solidity 的编译器项目，并作出贡献
- 设计 360 区块链的公链共识和公链多语言智能合约支持方案

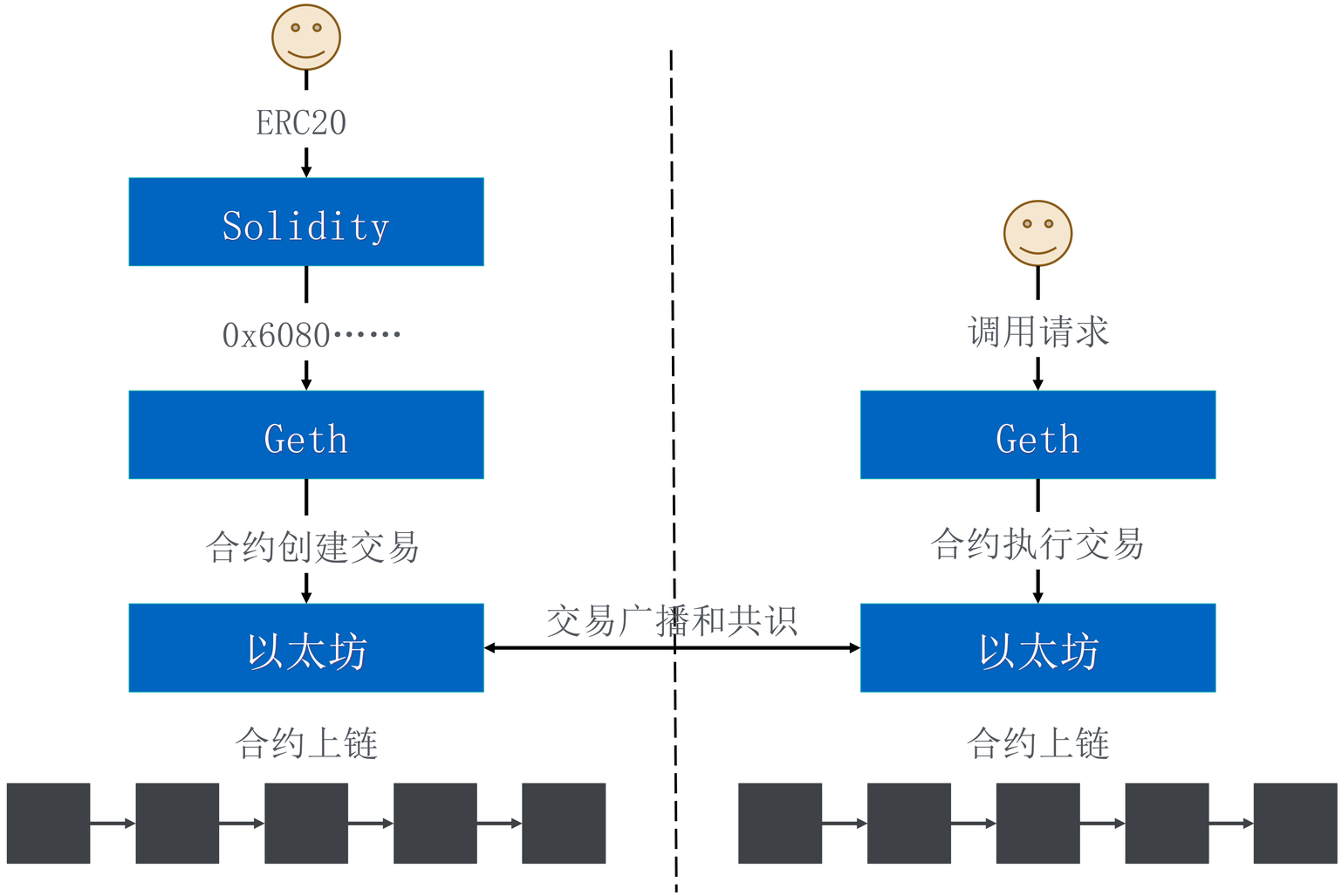
TABLE OF

CONTENTS 大纲

- EVM 架构原理
- 当前智能合约的问题
- Go 智能合约
- 常见问题及其防护

EVM 架构原理

以太坊合约的执行


ERC20
Solidity
0x6080……
Geth
合约创建交易
以太坊
合约上链
交易广播和共识
调用请求
Geth
合约执行交易
以太坊
合约上链

EVM 字节码

```
1 contract C {
2   int x;
3   constructor () public {
4     x = 88;
5   }
6   function f(int y)public payable returns (int) {
7     return y + 1;
8   }
9 }
```

合约代码

```
18 tag_1:
19     /* "x.sol":24:63  constructor () public {... */
20   pop
21     /* "x.sol":56:58  88 */
22   0x58
23     /* "x.sol":52:53  x */
24   0x0
25     /* "x.sol":52:58  x = 88 */
26   dup2
27   swap1
28   sstore
29   pop
30     /* "x.sol":0:65  contract C {... */
31   dataSize(sub_0)
32   dup1
33   dataOffset(sub_0)
34   0x0
35   codecopy
36   0x0
37   return
38 stop
```

合约上链

EVM 字节码

```
29 sub_0: assembly {
30         /* "y.sol":0:86  contract C {... */
31       mstore(0x40, 0x80)
32       jumpi(tag_1, lt(calldatasize, 0x4))
33       calldataload(0x0)
34       0x100000000000000000000000000000000000000000000000000000000
35       swap1
36       div
37       0xffffffff
38       and
39       dup1
40       0x1c008df9
41       eq
42       tag_2
43       jumpi
44     tag_1:
45       0x0
46       dup1
47       revert
48         /* "y.sol":15:84  function f(int x)pub
```

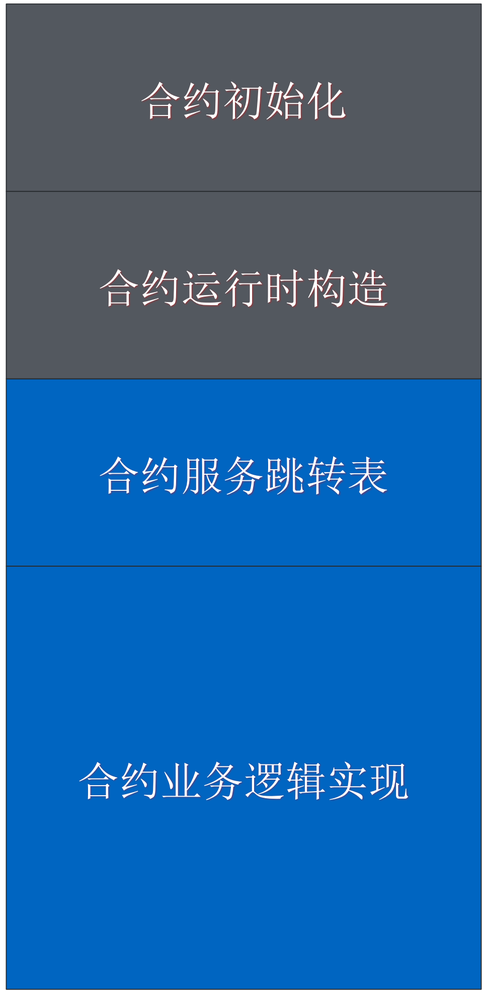
服务寻址

EVM 字节码

```
117      tag_5:
118          /* "x.sol":107:110  int */
119        0x0
120          /* "x.sol":129:130  1 */
121        0x1
122          /* "x.sol":125:126  y */
123        dup3
124          /* "x.sol":125:130  y + 1 */
125        add
126          /* "x.sol":118:130  return y + 1 */
127        swap1
128        pop
129          /* "x.sol":66:135  function f(int y)public pay
130        swap2
131        swap1
132        pop
133        jump  // out
```

业务逻辑

EVM 字节码整体结构



初始化 Storage 变量、构造子合约等

返回合约运行时

合约运行时的入口

合约具体业务逻辑实现

EVM 存储系统

Call Data

Stack

Memory

- ➢ 协议返回
- ✓ 内部参数
- ✓ 内部返回

Storage

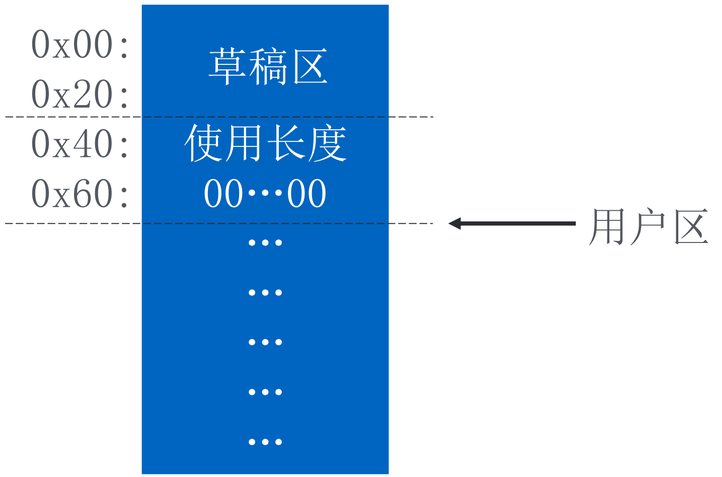
- ➢ 状态变量
- ✓ 局部变量

EVM 交易传参

- 参数会以一定方式编码成字节传输
- 调用返回跟参数使用同样的编码方式

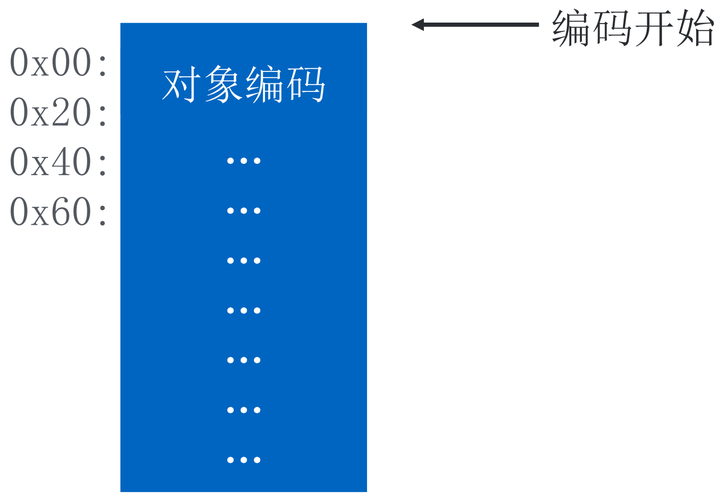
EVM Memory

- Solidity 没有内存复用，新变量直接由空地址开始存放
- Solidity 中所有内存对象都占用 32 个字节的整数倍
- Solidity 内存不支持变长类型



EVM/Solidity Storage

- 基本类型：依次排列，落在一个 256 位字内部
- 组合类型：起始位置 256 位对齐，成员按顺序编码

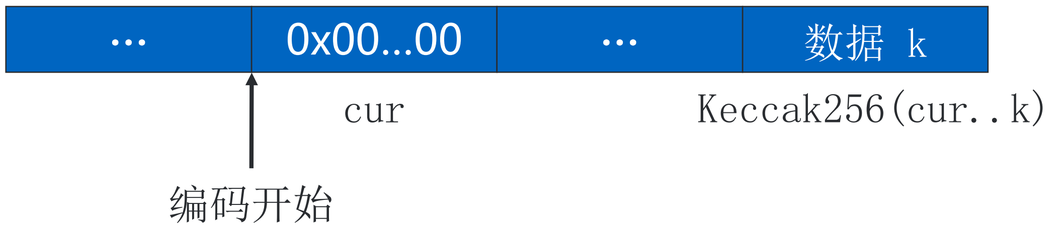


EVM/Solidity Storage

- 变长数组：长度放在当前位置，数据从 Keccak256(cur)开始存放

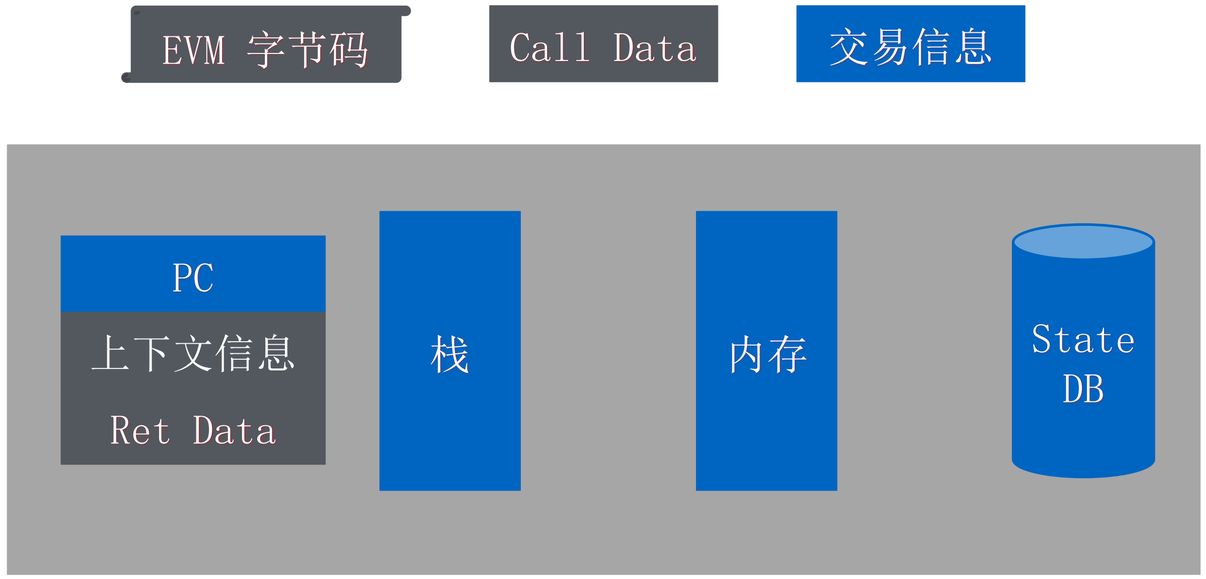
EVM/Solidity Storage

- 字典：当前 32 字节弃用，key k 对应的数据放在 Keccak256(cur..k)

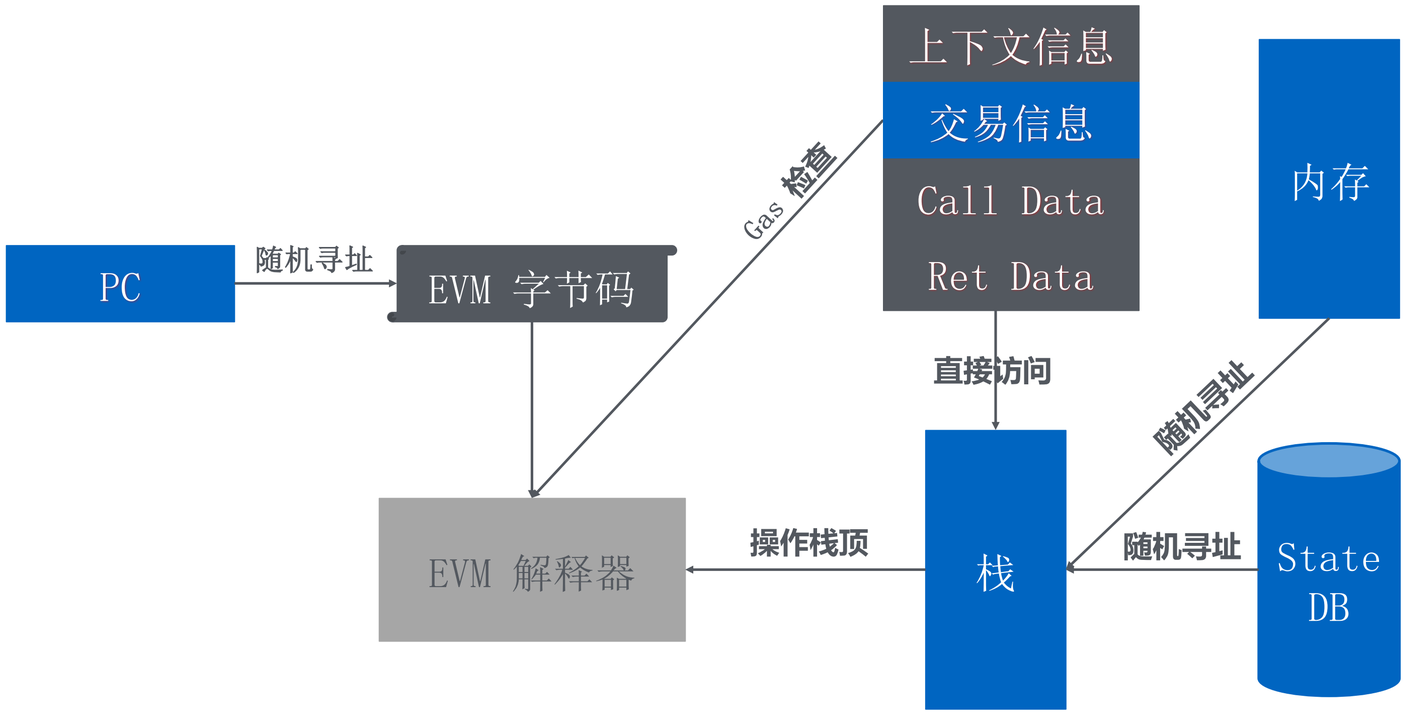


EVM 结构图

- 上下文信息：对应于 Solidity 中 block.xx、tx.xx
- 交易信息：对应于 Solidity 中 msg.xx



EVM 执行模型


上下文信息
交易信息
Call Data
Ret Data
内存
PC
随机寻址
EVM 字节码
Gas 检查
直接访问
随机寻址
EVM 解释器
操作栈顶
栈
随机寻址
State DB

局限性

EVM 的局限性（1）

- EVM 没有适合编译优化的中间表示（Solidity 提供并改进中）
- EVM 没有任何的编译优化相关的内容（分析、动静态优化）
- EVM 没有任何层次上的并行模型（SIMD、MIMD、VLIW、...）
- EVM 没有自动甚至手动存储管理（Solidity 也很简单）
- EVM 同时只能操作 17 个局部变量（受限于 Swap 指令）

EVM 的局限性（2）

- Solidity 的中间表示层还不太稳定
- Solidity 目前只有简单的编译优化
- Solidity 没有并行执行模型（EVM 不提供）
- Solidity 没有自动存储管理，手动管理也很粗糙

其他选择

- NEO：.NET ➔ NEO（没有并行）
- EVM 的 SIMD 指令（EIP 616，进展缓慢）
- eWASM：SIMD 使能了，多线程还不支持

Go 智能合约设计

智能合约需要什么

- 确定性：不确定性导致的不可重复验证
- 安全性：智能合约的隔离
- 终止性：Gas 控制

确定性带来的问题

- 在库层面做裁剪（常见方案是不支持所有已有库）
- 在预编译过程做裁剪（目前没有具体实践）
- 在语法树上做裁剪（目前没有具体实践）
- 加强编译器语义检查（目前没有具体实践）
- 语言标准中的未定义行为消除（EVM 没有，EOS 提及少但很重视）
- 裁剪执行指令集（常见方案，因此禁止了很多高性能方案）
- 不确定行为审计、监测、动态管理（联盟链有类似方案）

安全性主要考虑方向

- 只读信息防篡改：代码、链上下文、交易上下文（gas 除外）
- 对数据库中数据权限管理（余额、 Nonce、代码、Storage、Log）
- 各种传统攻击模式的防护
  1. 算数问题：除零错、整数溢出、浮点误差累积等
  2. 指针和资源问题：空指针、野指针、使用未初始化内存、栈堆数据段
  3. 并发错误：死锁、饥饿、数据竞争
  4. 逻辑问题：重放攻击、时序逻辑问题等
  5. 不合理的类型转换导致的溢出或逻辑问题
  6. 输入检查缺失、错误处理不完整
  7. 未定义行为导致的逻辑问题
  8. 不终止导致的栈溢出或其他资源超出系统限制等问题
  9. … …

私链跟公链不一样

- 私链可以提供严格的审查制度，可以考虑保留高性能特性
- 私链的智能合约不应该只做简单的转账交易

Go 语言的优势

- 完善的编译优化框架和算法（分析、动静态优化）
- 完善的并行模型（SIMD、MIMD、VLIW、...）
- 完善的存储管理
- 成熟的同步异步并行执行模型
- 成熟的自动存储管理，如引用计数和 GC 等

Go 智能合约

- 合约编译：约定合约构造函数、构建跳转表
- 合约上链：源代码上链
- 合约执行：调用执行 + 启动缓存
- 安全管理：合约池做进程隔离、API 做链数据库权限管理
- 合约管理：合约通过严格审计测试后再上线

Go 智能合约的应用

- 用于开发 360 磐石链共识的股权抵押和 VRF 自选举
- 股权抵押数据需要达成全网共识，需要使用智能合约
- VRF 函数不可能使用 Solidity 开发
- 在我们能确保它的确定性的情况下，我们选择了 go

THANKS