# 以太坊虚拟机（EVM）简介

2018.9

# 以太坊虚拟机（EVM）

- 以太坊虚拟机 EVM 是智能合约的运行环境
- 作为区块验证协议的一部分，参与网络的每个节点都会运行EVM。他们会检查正在验证的块中列出的交易，并运行由EVM中的交易触发的代码
- EVM不仅是沙盒封装的，而且是完全隔离的，也就是说在EVM 中运行的代码是无法访问网络、文件系统和其他进程的，甚至智能合约之间的访问也是受限的
- 合约以字节码的格式（EVM bytecode）存在于区块链上
- 合约通常以高级语言（solidity）编写，通过EVM编译器编译为字节码，最终通过客户端上载部署到区块链网络中

# EVM和账户

- 以太坊中有两类账户： **外部账户** 和 **合约账户**，它们共用EVM中同一个地址空间
- 无论帐户是否存储代码，这两类账户对 EVM 来说处理方式是完全一样的
- 每个账户在EVM中都有一个键值对形式的持久化存储。其中 key 和 value 的长度都是256位，称之为 **存储空间**（storage）

# EVM和交易

- 交易可以看作是从一个帐户发送到另一个帐户的消息，它可以包含二进制数据（payload）和以太币
- 如果目标账户含有代码，此代码会在EVM中执行，并以payload 作为入参，这就是合约的调用
- 如果目标账户是零账户（账户地址为 0 ），此交易就将创建一个 **新合约** ，这个用来创建合约的交易的 payload 会被转换为 EVM 字节码并执行，执行的输出作为合约代码永久存储

# EVM和gas

- 合约被交易触发调用时，指令会在全网的每个节点上执行：这需要消耗算力成本；每一个指令的执行都有特定的消耗，**gas** 就用来量化表示这个成本消耗
- 一经创建，每笔交易都按照一定数量的 gas 预付一笔费用，目的是限制执行交易所需要的工作量和为交易支付手续费
- EVM 执行交易时，gas 将按特定规则逐渐耗尽
- **gas price** 是交易发送者设置的一个值，作为发送者预付手续费的单价。如果交易执行后还有剩余， gas 会原路返还
- 无论执行到什么位置，一旦 gas 被耗尽（比如降为负值），将会触发一个 out-of-gas 异常。当前调用帧（call frame）所做的所有状态修改都将被回滚

# EVM数据存储

**Storage**

- 每个账户都有一块持久化的存储空间，称为 storage，这是一个将256位字映射到256位字的 key-value 存储区，可以理解为合约的数据库
- 永久储存在区块链中，由于会永久保存合约状态变量，所以读写的 gas 开销也最大

**Memory**（内存）

- 每一次消息调用，合约会临时获取一块干净的内存空间
- 生命周期仅为整个方法执行期间，函数调用后回收，因为仅保存临时变量，故读写 gas 开销较小

**Stack**（栈）

- EVM 不是基于寄存器的，而是基于栈的，因此所有的计算都在一个被称为栈（stack）的区域执行
- 存放部分局部值类型变量，几乎免费使用的内存，但有数量限制

# EVM指令集

- 所有的指令都是针对"256位的字（word）"这个基本的数据类型来进行操作
- 具备常用的算术、位、逻辑和比较操作，也可以做到有条件和无条件跳转
- 合约可以访问当前区块的相关属性，比如它的块高度和时间戳

# 消息调用（Message Calls）

- 合约可以通过消息调用的方式来调用其它合约或者发送以太币到非合约账户
- 合约可以决定在其内部的消息调用中，对于剩余的 gas，应发送和保留多少
- 如果在内部消息调用时发生了 out-of-gas 异常（或其他任何异常），这将由一个被压入栈顶的错误值所指明；此时只有与该内部消息调用一起发送的 gas 会被消耗掉

# 委托调用（Delegatecall）

- 一种特殊类型的消息调用
- 目标地址的代码将在发起调用的合约的上下文中执行，并且 msg.sender 和 msg.value 不变
- 可以由此实现“库”（library）：可复用的代码库可以放在一个合约的存储上，通过委托调用引入相应代码

# 合约的创建和自毁

- 通过一个特殊的消息调用 **create calls**，合约可以创建其他合约（不是简单的调用零地址）
- 合约代码从区块链上移除的唯一方式是合约在合约地址上的执行自毁操作 **selfdestruct**； 合约账户上剩余的以太币会发送给指定的目标，然后其存储和代码从状态中被移除



# Q&A


尚硅谷