共识机制与智能合约

区块链共识算法

- 分布式系统共识问题
- 基于工作量证明的共识算法
- 基于权益证明的共识算法
- 拜占庭容错类共识算法
- 私有链共识算法

分布式系统共识问题

- 在分布式系统中，共识指即使在部分节点故障、网络延时、网络分区甚至有恶意节点故意破坏的情况下，所有正常节点对客户的请求达成一致的处理，最终实现系统状态一致性的过程。
- 区块链共识算法是指为了完成区块链系统中节点的共识，建立在一系列协议或规则基础上，能够实现区块构造、选择出块节点、区块验证和区块上链等工作，从而达到所有节点所存储的分布式账本数据一致性的算法。
- 共识算法是区块链系统的核心技术，决定着区块链系统最终所能达到的性能以及可以应用的领域。

分布式系统的共识问题

- 分布式系统存在如下的问题：
  - 节点之间的网络通讯是不可靠的，包括任意延迟和内容故障；
  - 节点的处理可能是错误的，甚至节点自身随时可能宕机；
  - 同步调用会让系统变得不具备可扩展性

- 理想的分布式系统一致性应该满足：
  - 可终止性（Termination）：一致的结果在有限时间内能完成；
  - 一致性（Agreement）：不同节点最终完成决策的结果应该相同；
  - 合法性（Validity）：决策的结果必须是其它进程提出的提案。

FLP不可能原理

- 一致性是指整个分布式系统对外呈现的状态。
- 共识算法是实现系统中所有节点达成一致的过程的算法。

- 1985年，由Fischer, Lynch 和 Patterson三人提出FLP不可能原理。
  - 在网络可靠，存在节点失效（即便只有一个）的最小化异步模型系统中，不存在一个可以解决一致性问题的确定性算法。

CAP原理

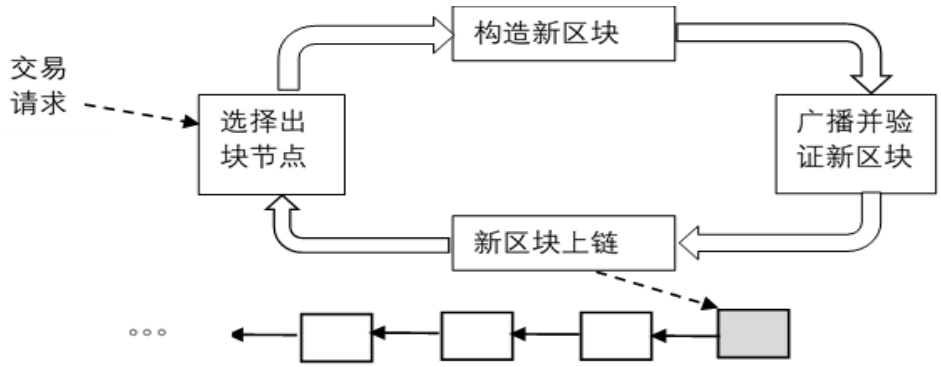
- 分布式领域CAP原理
  - 一致性（Consistency）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。
  - 可用性（Availability）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。
  - 分区容忍性（Partition tolerance）：因为节点之间的通信不可靠，网络可能发生分区。
- 定理：任何分布式系统只可同时满足二点，没法三者兼顾。
- 不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。

区块链共识算法的设计思想

- 在有拜占庭节点的情况下，达成共识，总得来说有两种思路：
  - 提高恶意节点的作恶成本
  - 诚实节点通过投票制止作恶节点

区块链系统共识过程模型

- 区块链系统共识过程模型
  - 区块链系统的共识过程可以用一个四阶段模型表示，即选择出块节点、构造新区块、广播并验证新区块、新区块上链。
  - 区块链共识算法的核心问题如何选择出块节点，在一些算法中又称为选主策略。



区块链共识算法分类

- 区块链系统共识算法分类
  - 是否拜占庭容错
  - 区块链系统部署类型
  - 是否强一致性
  - 选主策略

基于工作量证明的共识算法

- 比特币的信任基础不是依赖于中心化货币发行机构，耳机基于工作量证明（Proof Of Work， PoW ）共识算法。
- 核心思想是通过计算能力竞争的方式来保证数据一致性从而达成共识。
- 在比特币系统中，各节点基于各自的计算机算力的相互竞争来解决一个求解困难但验证容易的问题，最快解决该难题的节点获得区块记账权。
- PoW共识形成过程俗称“挖矿”，而参与竞争记账权的节点被称为“矿工”。

基于工作量证明的共识算法

- PoW共识算法的特点
  - 优点
    - 算法简单，容易实现
    - 节点间无需交换额外的信息即可达成共识
    - 破坏系统需要投入极大的成本，要有压倒大多数人的算力（51%攻击）。
  - 缺点
    - 浪费能源
    - 区块的确认时间难以缩短（10分钟左右），现在每秒交易量上限是7笔，因此性能比较低。

PoS 共识算法

- POS（Proof Of Stake）权益证明共识算法。
  - 主要应用：点点币(PeerCoin)、黑币(BlackCoin)等

- 核心思想：基于工作量证明的共识算法的主要问题在于巨大的能耗和较低的交易速度，为了解决这个问题，使用权益证明来代替基于哈希计算的工作量证明，持有的权益越大，挖到区块的概率越大。

- 权益体现为节点对特定代币的所有权，最常见权益方式币龄定义是每笔交易的代币金额乘以这笔交易的代币在账上留存的时间（以天为单位）。
  - 例如A具有1个币，持有80天，则节点A具有80币龄。

PoS 共识算法

- 挖矿方程
- Hash(Kernel)≤Target*币龄
  - Kernel类似于POW中的区块头，Target类似于POW中的随机数。可以看到币龄（即权益）越大，挖矿成功的概率越大

- PoS是针对PoW缺陷产生的替代技术，用户必须具有系统中的一些权益，每次竞争都会消耗掉用户的权益。

PoS 共识算法

- PoW中，挖矿成功的概率和矿工算力成正比，而PoS中，区块生成的概率和节点的权益(例如币龄)成正比，因此减少了消耗的算力；
- 每个区块交易都会消耗币龄，攻击者要想实现攻击必须累积很高的币龄，成本比PoW中的算力更高，攻击持续的难度也增加。
- 对于区块分叉，PoS算法以总消耗币龄最大为标准分确定主链。

PoS 共识算法

- PoS共识算法具有以下优点：
  - 节省能源
  - 提升性能
  - 降低中心化趋势
  - 安全性

PoS 共识算法

- PoS算法的缺点
  - 最初的货币还是得借助PoW算法来产生；
  - 拥有权益的参与者未必希望参与记账，这会导致囤币行为；
  - 用户可能选择离线累积币龄，这会导致区块链网络中在线节点数量的减少；
    - Blackcoin系统中实现的PoS共识算法中，不再使用币龄作为权益，而是使用权重，权重只有在线才能累积，从而鼓励节点保持在线，提高系统的安全性。

PoS 共识算法

- PoS算法的性能
  - PoS算法中因为挖矿计算的难度相比PoW算法大大降低，因此出块时间也改善很多。
  - 在Blackcoin中可以达到64秒一块的速度，相应的交易吞吐率可以达到数百TPS。

DPoS 共识算法

- Delegated Proof-of-Stake Consensus委托权益证明共识算法
  - 利益相关方（Stakeholders）投票选举见证人（Witnesses）来生成区块。
  - 每个账户允许为每个见证人投一张票，见证人必须至少有50%的投票才能当选， 当利益相关者提出他们所希望的见证人数量时，他们也必须投不低于该数量的投票。
  - 见证者生成区块成功可以获得一定收益，生成失败则没有收入，同时还有可能在未来被投票出局失去见证人身份。

DPoS 共识算法

- DPoS共识算法的共识过程分为两个阶段：
  - 见证人选举阶段：拥有权益的股东节点，投票选出N个见证人节点，每个见证人所获得的票数必须超过50%，因为50%被认为可以提供足够的去中心化程度。
  - 见证人出块：每一次见证人会有2秒的固定时间来生成新的区块，如果不能在给定时间内完成，就换一个见证人造块。见证人按时出块成功后，会将新区块发给其他见证人进行验证。

DPoS 共识算法

- 代表（Delegates）
  - 利益相关方选举代表进行网络参数设置，包括交易费用，区块大小，见证人服务费用和区块生产的间隔时间等。
  - 大多数代表批准了提议的变更后，利益相关方被授予2周的审查期，在此期间他们可以为代表投票是否同意或者取消提议的变更。
- 总结：
  - DPoS有点像是议会制度或人民代表大会制度。如果代表不能履行他们的职责（当轮到他们时，没能生成区块），他们会被除名，网络会选出新的超级节点来取代他们。

DPoS 共识算法

- DPoS算法的性能
  - BitShares系统中DPoS共识算法使得加密货币交易速度能够达到100000TPS的性能。

- 因为见证人都是选举产生的，可靠性很高，并且限定时间片来产生新块，因此很难发生两条竞争的区块链分叉共存的现象。

DPoS 共识算法

- DPoS主要应用
  - 比特股(bitshares)

- 优点
  - 大幅缩小参与验证和记账节点的数量，可以达到秒级的共识验证。

- 缺点
  - 由少数节点代替多数节点进行共识，其实是牺牲了区块链去中心化的特性，以此来换取共识效率的提升。（通过股东选举，本质上还是去中心化的。）

PBFT共识算法

- PBFT实用拜占庭容错共识算法基于一种状态机副本复制的思想，通过多轮消息传递来实现系统中诚实节点达成一致的共识，将算法时间复杂度降到了多项式级，从而可以在实际的区块链系统中实现。
- PBFT共识算法主要应用于联盟链中，可以容忍恶意节点不超过全网节点数量1/3的场景。
- 基于PBFT的区块链系统中的节点被分成主节点和备份节点两种。
- 每次生成新的区块并添加到区块链的工作作为一个完整的流程，称为视图，在每个视图中，只有一个节点会被选举为主节点，主节点负责生成新区块

PBFT共识算法

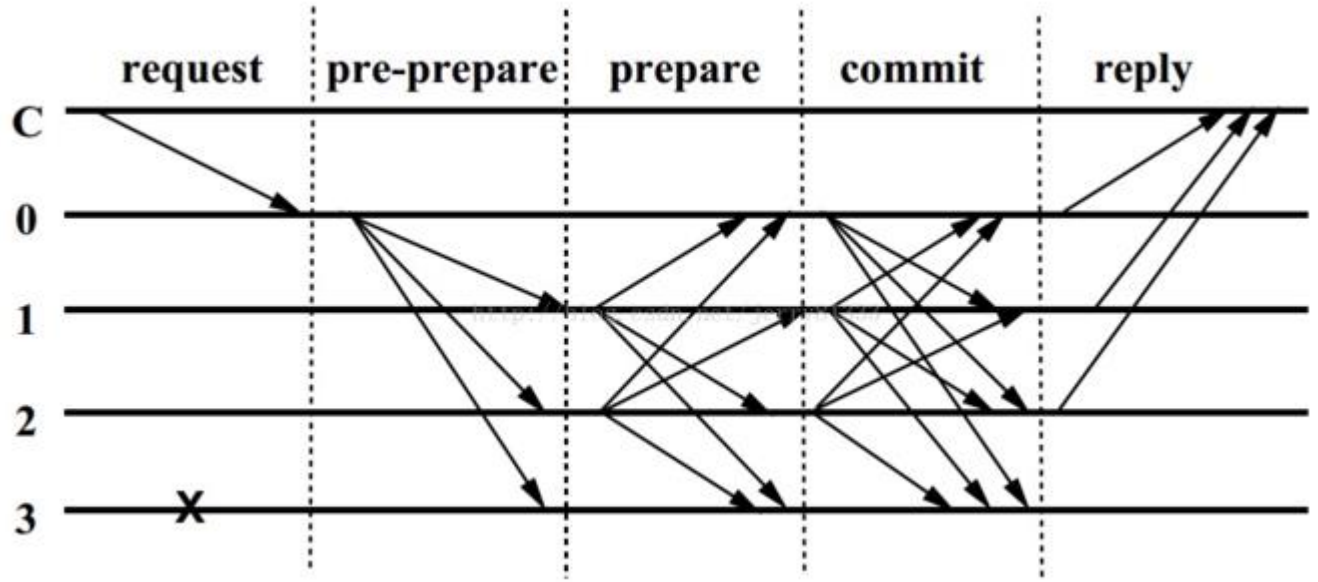
- 实用拜占庭容错（Practical Byzantine Fault Tolerance）共识算法
  - 主要应用于联盟链中，例如Hyperledger Fabric 0.6；
  - 可以容忍恶意节点不超过全网节点数量1/3的场景；
  - 所有参与记账的节点中选举一个主节点，其他节点是副本节点；
  - 每个生成新区块的过程称为一个视图，具有一个视图编号v；
  - 主节点编号p确定规则：$p = v \bmod n$，其中n是节点总数；
  - 必要时启动视图切换；
  - 通过多轮消息传递来实现系统中诚实节点达成一致的共识。

PBFT共识算法

- PBFT新区块产生过程
  - 请求（Request）阶段：客户端c向主节点p发送交易请求；
  - 预准备（Pre-prepare）阶段：主节点收到客户端的请求消息验证后，为请求分配编号，然后发送PRE-PREPARE消息给所有副本节点；
  - 准备（Prepare）阶段：副本节点收到主节点的PRE-PREPARE消息并验证后，其他节点包括主节点发送PREPARE消息；
  - 确认（Commit）阶段：所有参与记账的节点如果收到了2f+1个验证通过的PREPARE消息，则向其他所有节点发送COMMIT消息；
  - 响应（Reply）阶段：如果每个节点收到了2f+1个验证通过的COMMIT消息，说明当前网络中的大部分节点已经达成共识，可以运行客户端的请求操作o，并返回REPLY消息给客户端，而客户端如果如果收到f+1个相同的REPLY消息，说明客户端发起的请求已经达成全网共识。

PBFT共识算法

- 一个示例PBFT算法工作过程。



PBFT共识算法

- 如果参与共识的节点总数为n，则PBFT共识算法在准备阶段和确认阶段需要传输的消息数量都是$O(n^2)$数量级。
- 共识的时延大约在2~5秒钟，基本达到商用实时处理的要求。
- 当有1/3或以上参与记账的节点无法工作后，系统将无法提供服务。
- 这是一种确定性共识算法。

经典分布式共识算法

- 主要用于无拜占庭节点的分布式系统中
  - 分布式数据库
  - 私有链
- 典型代表算法
  - Paxos类：
  - Raft
- 主要思想：
  - 状态机复制

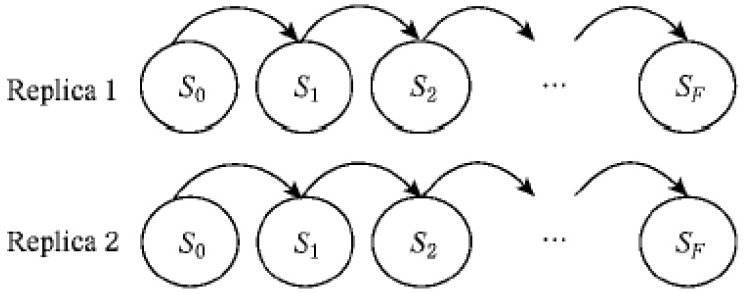


Fig. 1 The principle of replicated state machine

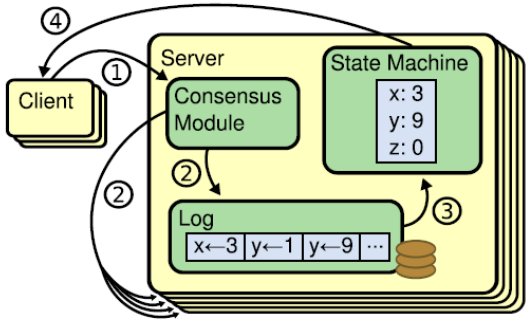


Figure 1: Replicated state machine architecture. The consensus algorithm manages a replicated log containing state machine commands from clients. The state machines process identical sequences of commands from the logs, so they produce the same outputs.

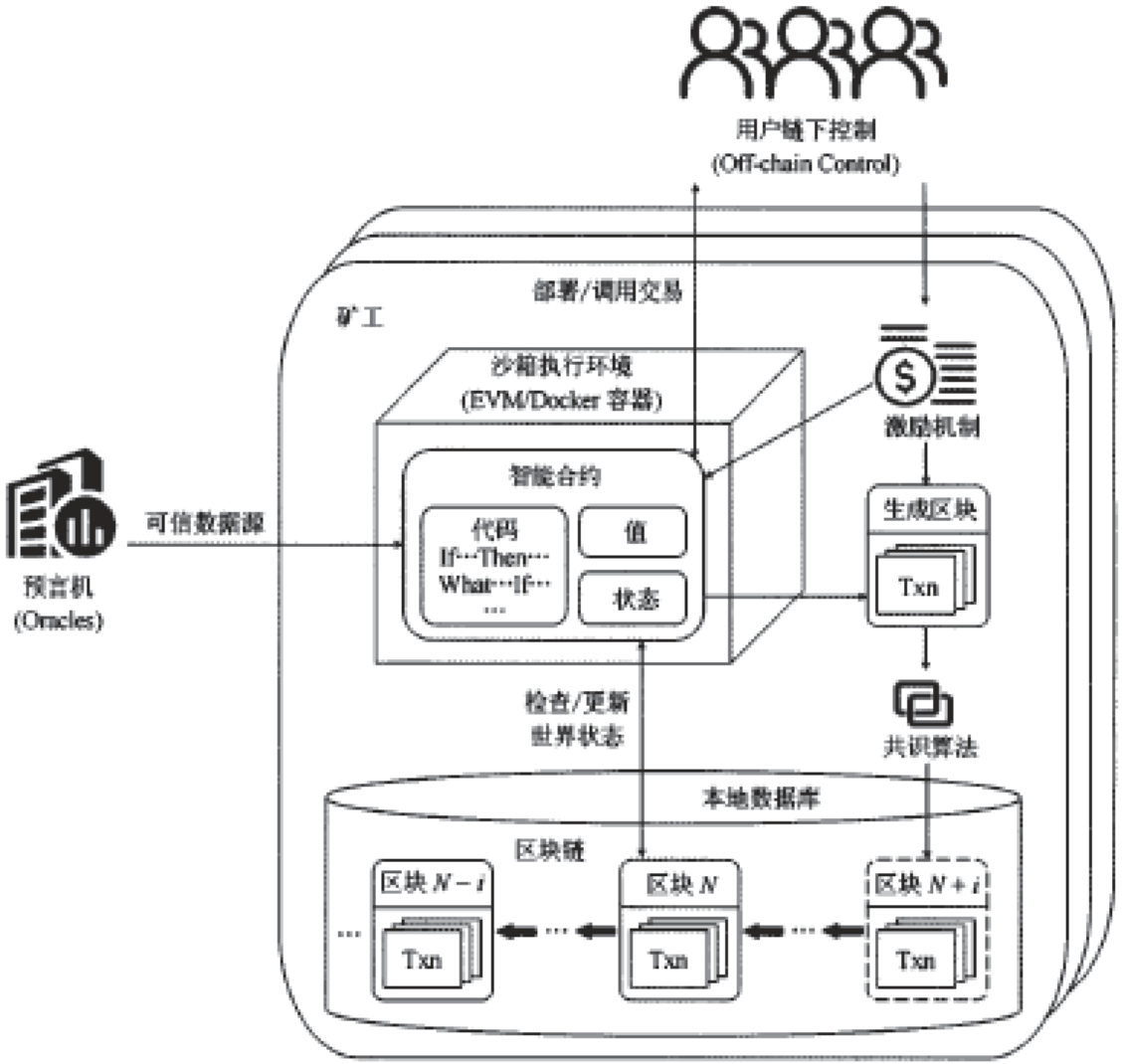
智能合约

- 智能合约的简单例子
  - 众筹：根据合约决定筹款、自动退款（失败时）、进度投票、成功后的自动分红等。
  - 投票：创建提案、投票操作、自动计票、宣布结果等
  - 网络购物：卖方发货再付款

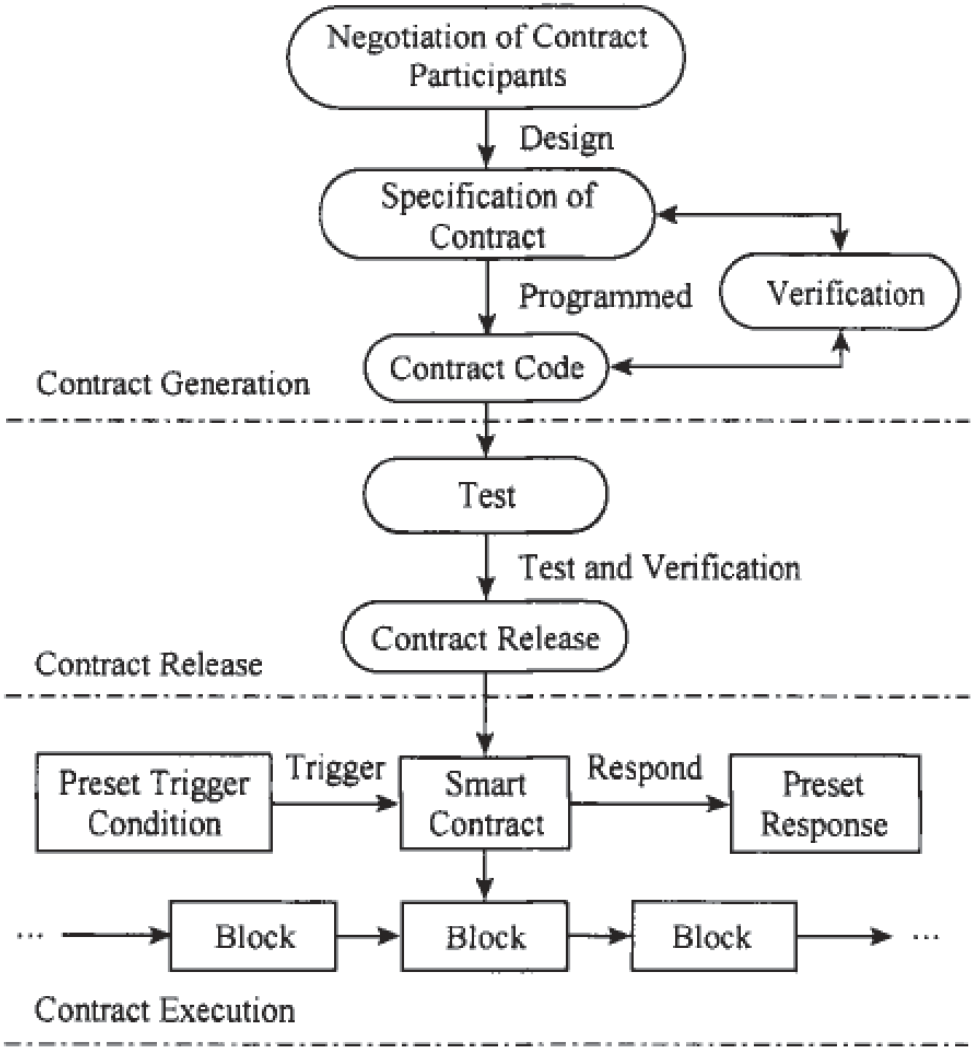
智能合约

- 智能合约是一组情景应对型的程序化规则和逻辑，是通过部署在区块链上的去中心化、可信共享的脚本代码实现的。
- 通常情况下，智能合约经各方签署后，以程序代码的形式附着在区块链数据上，经P2P网络传播和节点验证后记入区块链的特定区块中。
- 智能合约封装了预定义的若干状态及转换规则、触发合约执行的情景、特定情景下的应对行动等。
- 区块链可实时监控智能合约的状态，并通过核查外部数据源、确认满足特定触发条件后激活并执行合约。

智能合约运行机制

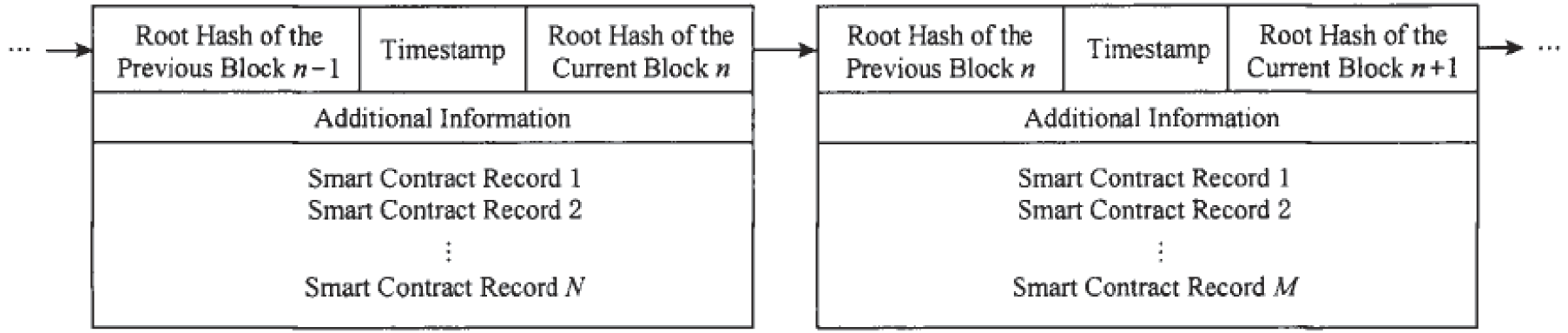

用户链下控制
(Off-chain Control)
部署/调用交易
矿工
沙箱执行环境
(EVM/Docker 容器)
激励机制
智能合约
代码
If…Then…
What…If…
…
值
状态
生成区块
Txn
可信数据源
预言机
(Oracles)
检查/更新
世界状态
共识算法
本地数据库
区块链
区块 N − i
区块 N
区块 N + i
Txn
Txn
Txn

智能合约的生命周期


Negotiation of Contract Participants
Design
Specification of Contract
Verification
Programmed
Contract Code
Contract Generation
Test
Test and Verification
Contract Release
Contract Release
Preset Trigger Condition
Trigger
Smart Contract
Respond
Preset Response
Block
Block
Block
Contract Execution

智能合约区块

- 智能合约区块的概念



智能合约

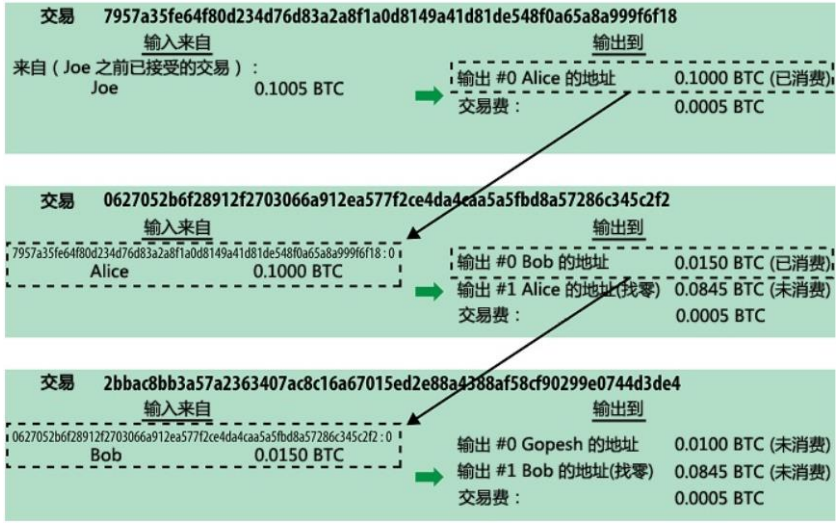
- 智能合约已在许多区块链系统上成功实现,比较著名的系统有以太坊和超级账本.
  - 以太坊((Ethereum)是一个开源的支持智能合约功能的公共区块链平台，在2013 年受比特币发展的启发而提出的。其由于支持用户开发智能合约的功能,被称为第二代区块链系统。
  - 超级账本(Hyperledger Fabric)是一个为部署商业应用许可区块链而设计的系统，具有良好的灵活性和通用性，支持种类繁多的非确定性智能合约(链码)和可插拔的服务。

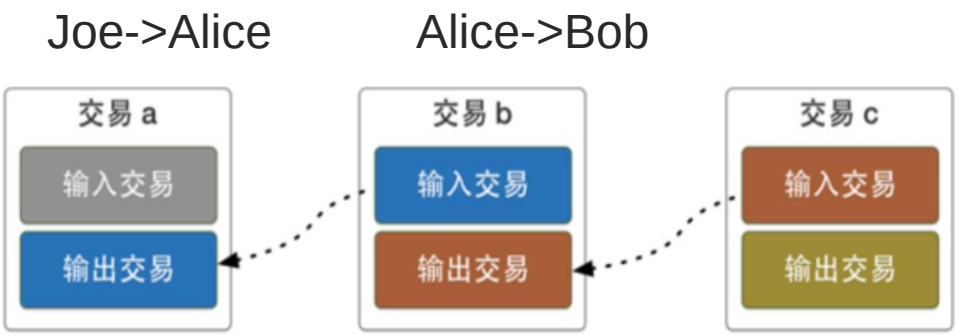
智能合约编程语言

- 从编程语言表现或者运行环境考虑,智能合约可以分为
  - 脚本型
    - 比特币系统可以允许通过编写基于堆栈的操作码(Opcode)来实现简单的交易逻辑,比如改变比特币花费的前提条件,这个系统称为比特币脚本系统。
  - 图灵完备型
  - 可验证合约型
    - 正在开发中的Kadena 项目提供一个可验证的智能合约系统,以保证商业应用逻辑的可靠性和高效性

比特币脚本语言

- 比特币交易验证过程
  - 比特币系统的交易依赖于脚本。支付款项时，是把支付的数额与接收者的公钥脚本（上一个交易的输出脚本）绑定到一起。日后，接收者可以用自己的签名脚本（当前交易的输入脚本）来确认使用权。每一笔交易的实现所依赖的只有脚本。
  - 比特币的脚本系统借助堆栈进行运算。验证交易时，脚本系统依次读取每个指令，并对堆栈进行操作。所有指令结束后，检查堆栈中的残留数据，如果均为TRUE，则交易有效，否则交易无效。





比特币脚本语言

比特币脚本语言实例

Hash:

> 9c50cee8d50e273100987bb12ec46208cb04a1d5b68c9bea84fd4a04854b5eb1

输入交易:

> 前导输入的Hash:
>
> > 437b95ae15f87c7a8ab4f51db5d3c877b972ef92f26fbc6d3c4663d1bc750149
>
> 输入脚本 scriptSig:
>
> > 3045022100efe12e2584bbd346bccfe67fd50a54191e4f45f945e3853658284358d9c06
> > 2ad02200121e00b6297c0874650d00b786971f5b4601e32b3f81afa9f9f8108e93c7522
> > 01038b29d4fbbd12619d45c84c83cb4330337ab1b1a3737250f29cec679d7551148a

输出交易:

> 转账值:
>
> > 0.05010000 btc
>
> 输出脚本 scriptPubKey:
>
> > OP_DUP OP_HASH160 be10f0a78f5ac63e8746f7f2e62a5663eed05788
> > OP_EQUALVERIFY OP_CHECKSIG

Hash:

> 9c50cee8d50e273100987bb12ec46208cb04a1d5b68c9bea84fd4a04854b5eb1

输入交易:

> 前导输入的Hash:
>
> > 437b95ae15f87c7a8ab4f51db5d3c877b972ef92f26fbc6d3c4663d1bc750149
>
> 输入脚本 scriptSig:
>
> > 3045022100efe12e2584bbd346bccfe67fd50a54191e4f45f945e3853658284358d9c06
> > 2ad02200121e00b6297c0874650d00b786971f5b4601e32b3f81afa9f9f8108e93c7522
> > 01038b29d4fbbd12619d45c84c83cb4330337ab1b1a3737250f29cec679d7551148a

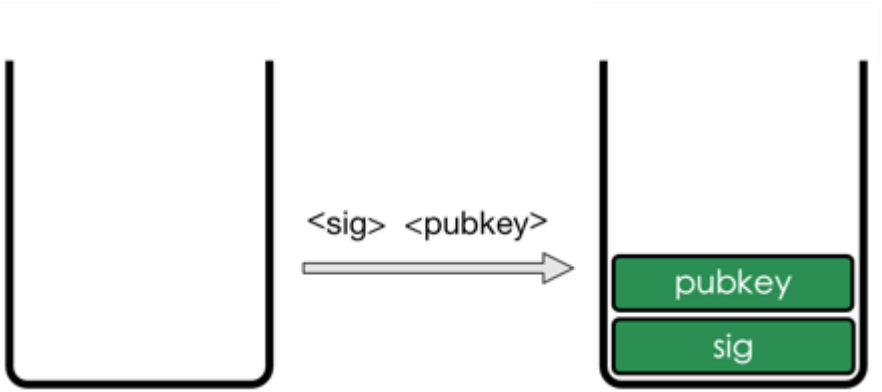
输出交易:

> 转账值:
>
> > 0.05010000 btc
>
> 输出脚本 scriptPubKey:
>
> > OP_DUP OP_HASH160 be10f0a78f5ac63e8746f7f2e62a5663eed05788
> > OP_EQUALVERIFY OP_CHECKSIG

```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash?> OP_EQUALVERIFY OP_CHECKSIG
```

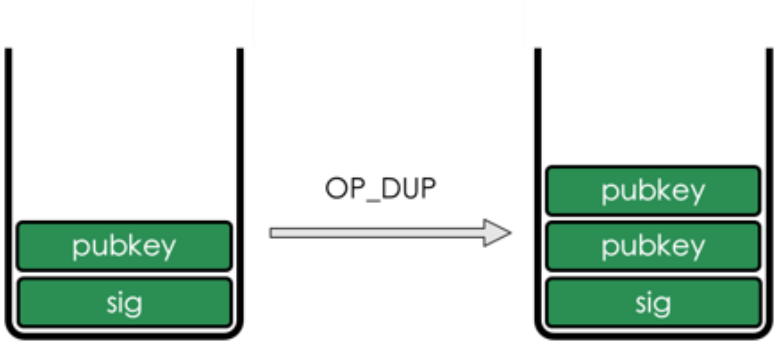
比特币脚本语言

- 脚本验证过程
  - 首先执行的是输入脚本（解锁脚本）。因为脚本是从左向右执行的，那么先入栈的是付款人私钥交易签名，随后是付款人的公钥。

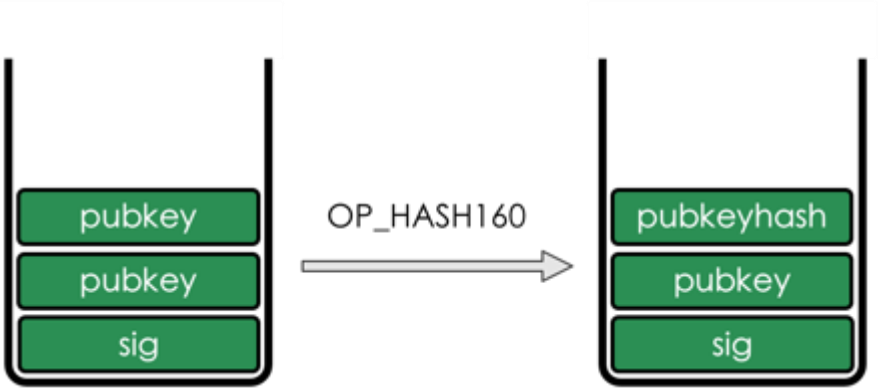


比特币脚本语言

- 脚本验证过程
  - 接着，执行的是输出脚本（锁定脚本）。从左向右执行，第一个指令是OP_DUP——复制栈顶元素，即复制一份付款人的公钥。

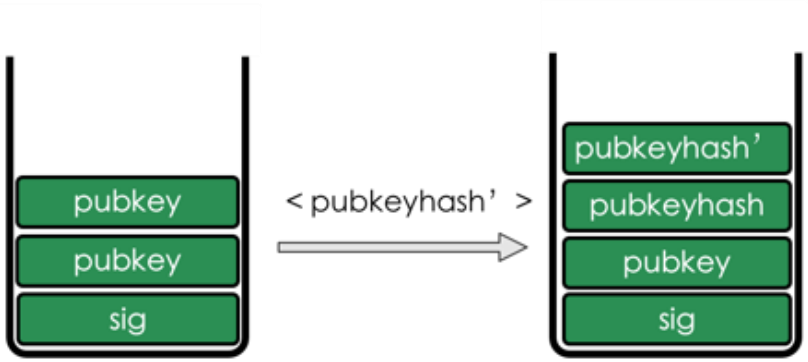


  - OP_HASH160——使用hash160算法对栈顶数据（公钥）做哈希运算，得到pubkeyhash

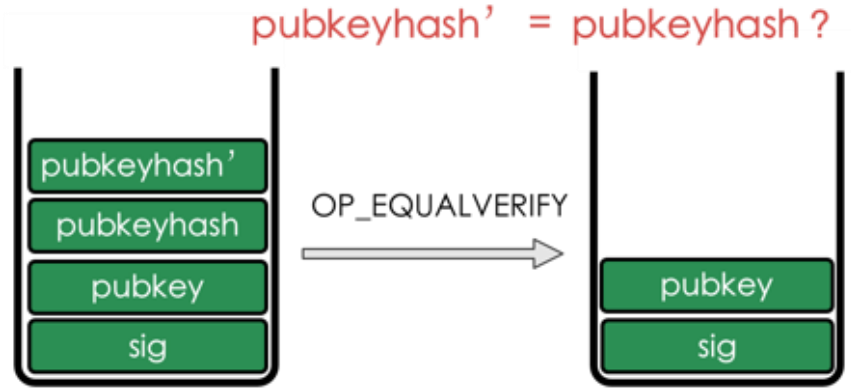


比特币脚本语言

- 脚本验证过程
  - 将输出脚本中的付款人的公钥哈希pubkeyhash‘入栈。

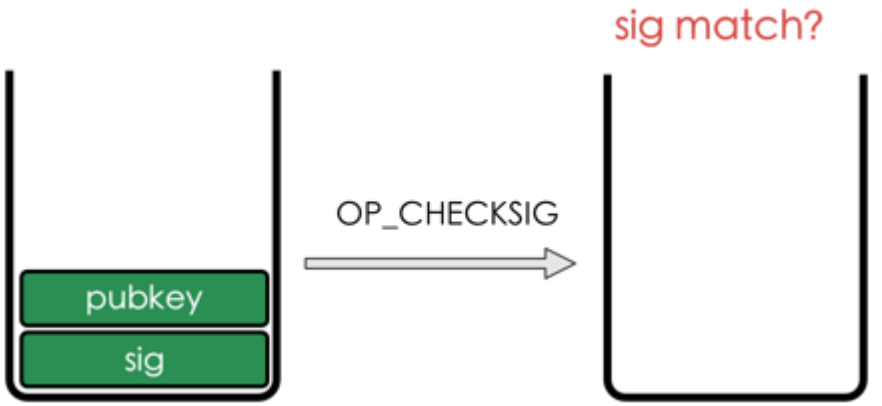


  - OP_EQUALVERIFY——检查栈顶前两元素是否相等，如果相等继续执行，否则中断执行，返回失败



比特币脚本语言

- 脚本验证过程
  - OP_CHECKSIG——使用栈顶前两元素执行签名校验操作（验证付款人公钥和数字签名是否匹配），如果相等，返回成功，否则返回失败。



比特币脚本语言

- 比特币脚本语言用于编写比特币交易中未花费交易输出(UTXO)的锁定脚本(Locking Script)和解锁脚本(Unlocking Script)。
- 锁定脚本确定了花费输出所需要的条件,而解锁脚本用来满足UTXO 上锁定脚本所确定的条件，解锁并支付。
- 当一条交易被执行时,每个UTXO 的解锁脚本和锁定脚本同时执行，根据执行结果(true/false)来判定该笔交易是否满足支付条件。
- 脚本是非图灵完备的语言,包含的操作码不具备循环和复杂的流控制功能,仅可执行有限的次数,避免了因编写疏忽等原因导致的无限循环或其他类型的逻辑炸弹.

图灵完备型智能合约编程语言

- 图灵完备型
  - 所谓图灵完备,是指能用该编程语言模拟任何图灵机，能够用图灵机能做到的所有事情，可以解决所有的可计算问题。
  - 以太坊系统提供以太坊虚拟机(Ethereum Virtual Machine,EVM)，合约代码在EVM 内部运行。以太坊用户使用特定语言编写智能合约代码，并编译成EVM字节码运行。
  - 超级账本提供另一种图灵完备智能合约，它在Docker容器环境中运行语言无关的智能合约，即智能合约代码可以使用任何编程语言进行编写，之后被编译器编译并打包进Docker 镜像，以容器作为运行环境。

图灵完备型智能合约编程语言

- 以太坊图灵完备型语言
  - 以太坊提供了智能合约专用开发语言,其他系统或平台大多采用通用编程语言。
  - Solidity （语法上类似于Java-Script）, 官方推荐
  - Serpent （语法上类似于Python ）
  - Mutan （语法上类似于C 语言）已停止维护
  - LLL 已经废弃,官方代码库也已经无法访问

智能合约的实现技术

- 区块链中智能合约的实现技术
  - 按照智能合约运行的环境进行划分,具体可分为3 类:
    - 嵌入式运行（例如比特币）
    - 基于虚拟机运行（例如以太坊）
    - 容器式运行（例如超级账本）

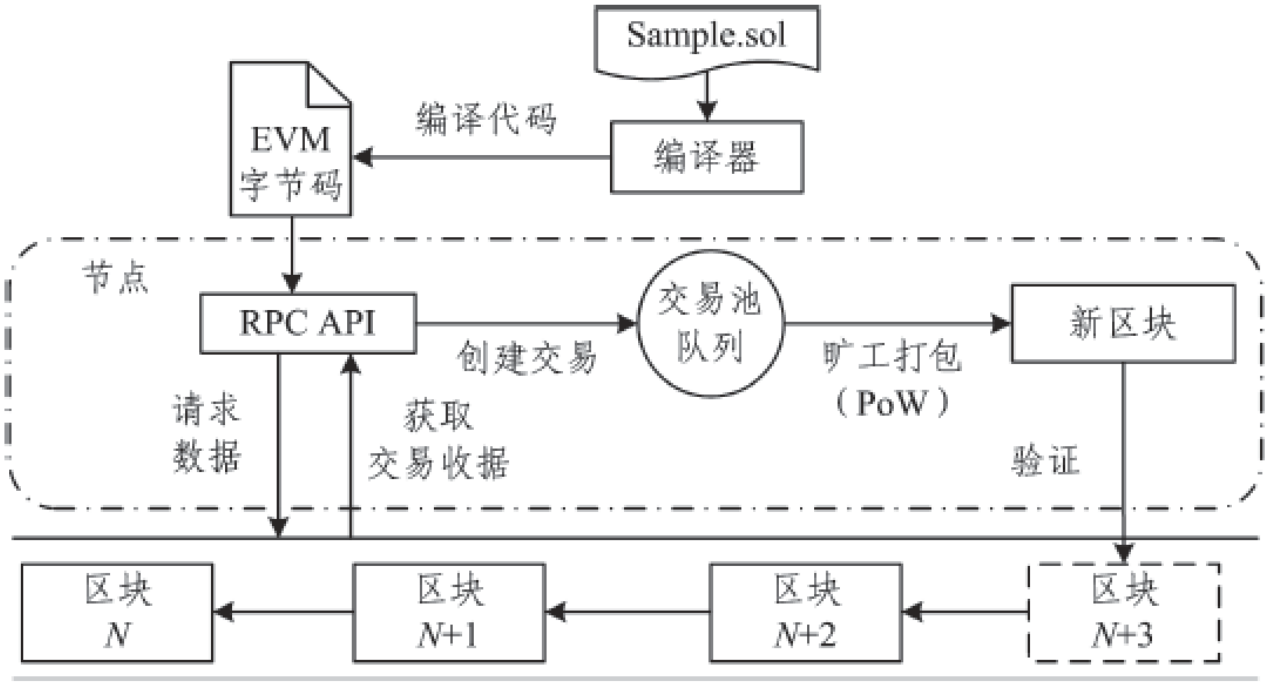
智能合约实践

- 以太坊通过建立终极的抽象的基础层-内置有图灵完备编程语言的区块链-使得任何人都能够创建合约和去中心化应用，并在其中设立他们自由定义的所有权规则、交易方式和状态转换函数。

- 以太坊有Go、C++、Python、JavaScript等不同的版本。

以太坊智能合约基本原理

- 以太坊虚拟机EVM是一个完全独立的沙盒，合约代码在EVM 内部运行并且对外隔离。
- 智能合约部署的流程



以太坊智能合约基本原理

- 以太坊中的两种不同类型的账户：
  - 外部账户：被私钥控制且没有任何代码与之关联。
  - 合约账户：被合约代码控制且有代码与之关联。

以太坊智能合约基本原理

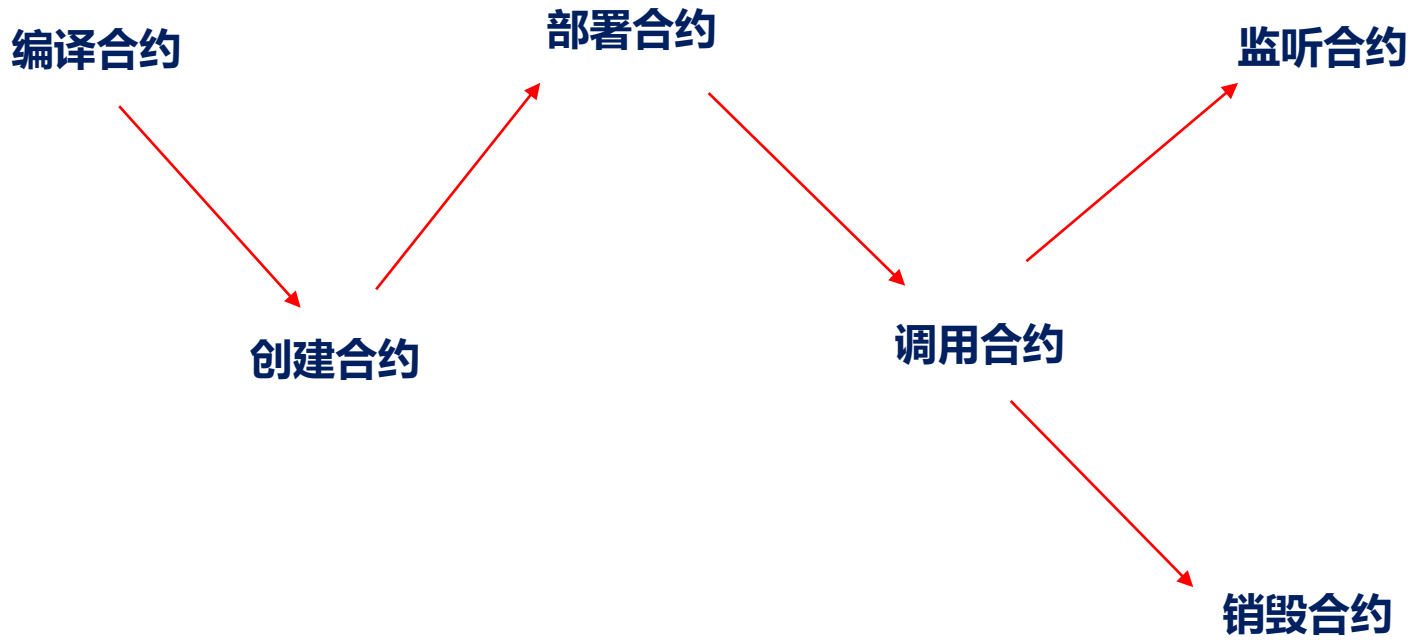
- 以太坊支持的交易类型
  - 从一个账户向另一个账户发送以太币
    - 一个外部账户向另一个外部账户发送以太币
    - 一个外部账户向另一个合约账户发送以太币
    - 一个合约账户向另一个合约账户发送以太币
    - 一个合约账户向另一个外部账户发送以太币
  - 智能合约部署
    - 外部账户在EVM上部署合约是通过交易的方式实现的。
- 通过以太坊浏览器可以查看交易信息
  - The Ethereum Blockchain Explorer
  - https://cn.etherscan.com/

以太坊智能合约基本原理

- 账户状态有四个组成部分：
  - nonce
    - 外部账户，nonce代表从此账户地址发送的交易序号。
    - 合约账户，nonce代表此账户创建的合约序号
  - balance： 此地址账户余额，以Wei为单位。1Ether=10^{18}Wei
  - storageRoot：此账户存储内容的Hash值的Merkle树的根节点Hash值，默认是空值。
  - codeHash：
    - 合约账户，EVM合约代码的hash值。
    - 外部账户，codeHash域是一个空字符串的Hash值。

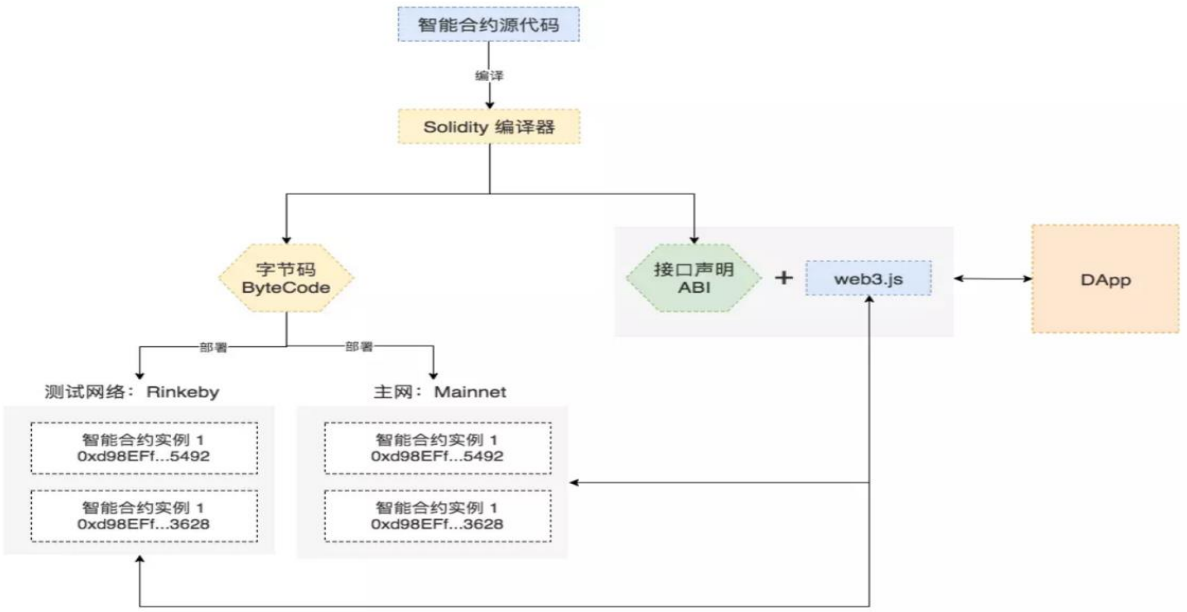
智能合约实践

- 以太坊智能合约应用步骤



以太坊智能合约基本原理

- 以太坊上部署和运行智能合约的过程
  1. 启动一个以太坊节点（如geth）。
  2. 使用智能合约语言编写智能合约（如Solidity）。
     - 先使用相关编译器将智能合约源代码编译为以太坊EVM字节码，编译时会生成智能合约的二进制接口规范（Application Binary Interface/ABI）。
     - 通过交易的方式将字节码部署到以太坊网络，成功部署都会为该智能合约产生一个合约账户。
     - DApp通过web3.js + ABI等方式去调用智能合约中的函数，这也是以太坊交易。

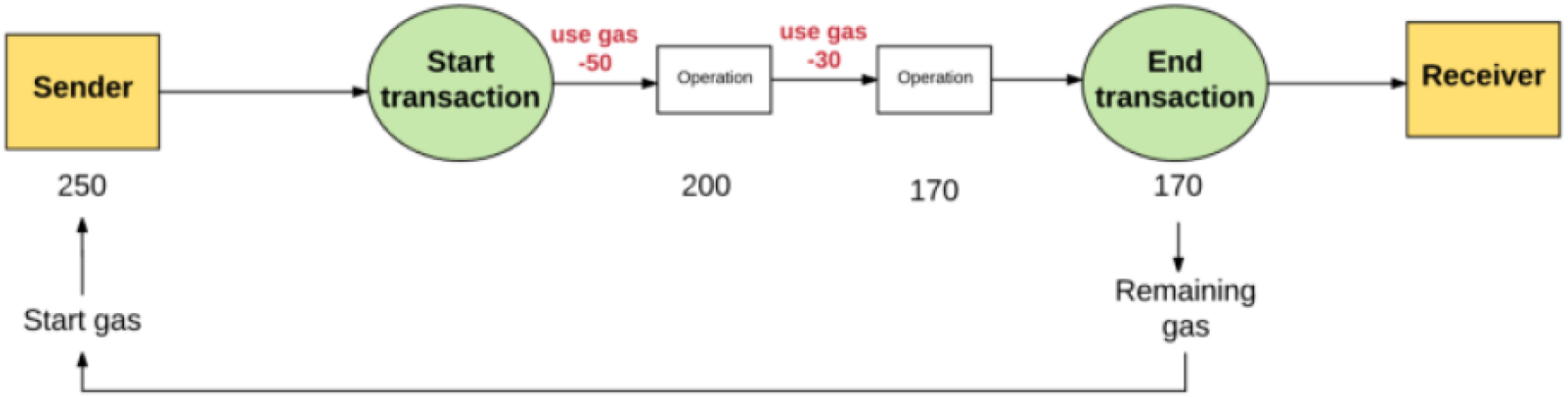


以太坊智能合约基本原理

- 以太坊的gas
  - 为了防止区块链网络资源滥用或由图灵完备引起的无限循环故障，任何可编程的计算都受计费限制；
  - 该计费机制以gas 为单位，创建智能合约、调用消息、存储空间使用以及虚拟机上的运行操作都对应一定的gas计费标准；
  - gas 计费的引入为智能合约的运行提供了机制上的安全保障，一旦gas 超过计费限制gasLimit，整个交易将会被回滚，以保证数据的完整性和安全性。
  - 合约执行会在所有节点中被多次重复，这个事实得使得合约执行的消耗变得昂贵，所以这也促使大家将能在链下进行的运算都不放到区块链上进行。

以太坊智能合约基本原理

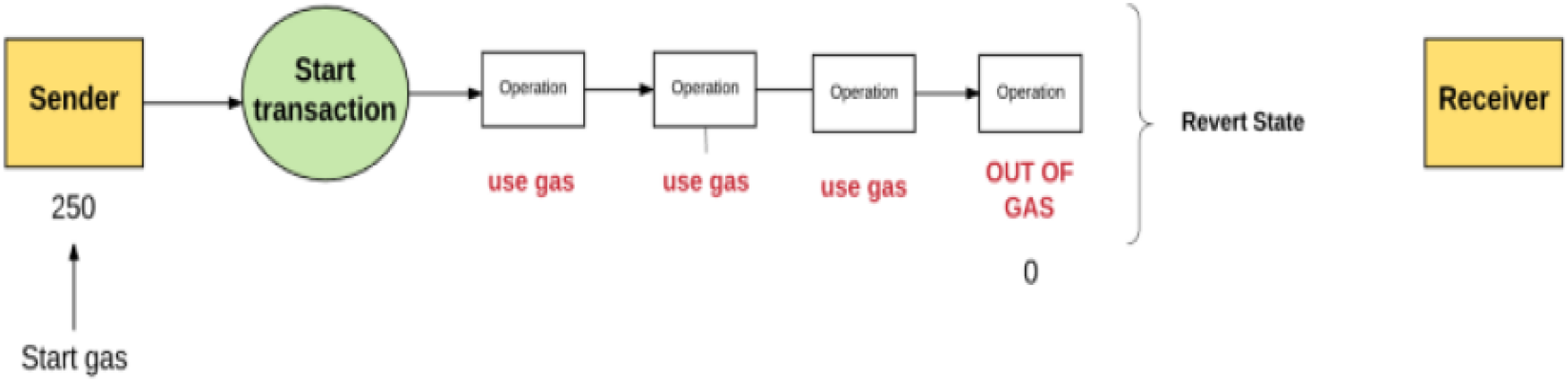
- gas limit代表用户愿意花费在gas上费用的最大值。如果在他们的账户余额足够,在交易结束时任何未使用的gas都会被返回给发送者。



所有消耗的gas最终被作为奖励转入矿工的账户。

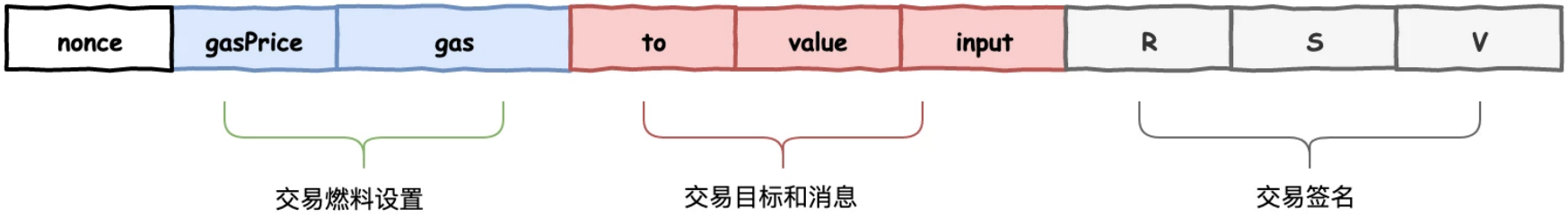
以太坊智能合约基本原理

- 没有提供足够的gas来执行交易，交易处理在耗尽所有gas后会被终止以及所有已改变的状态将会被恢复。



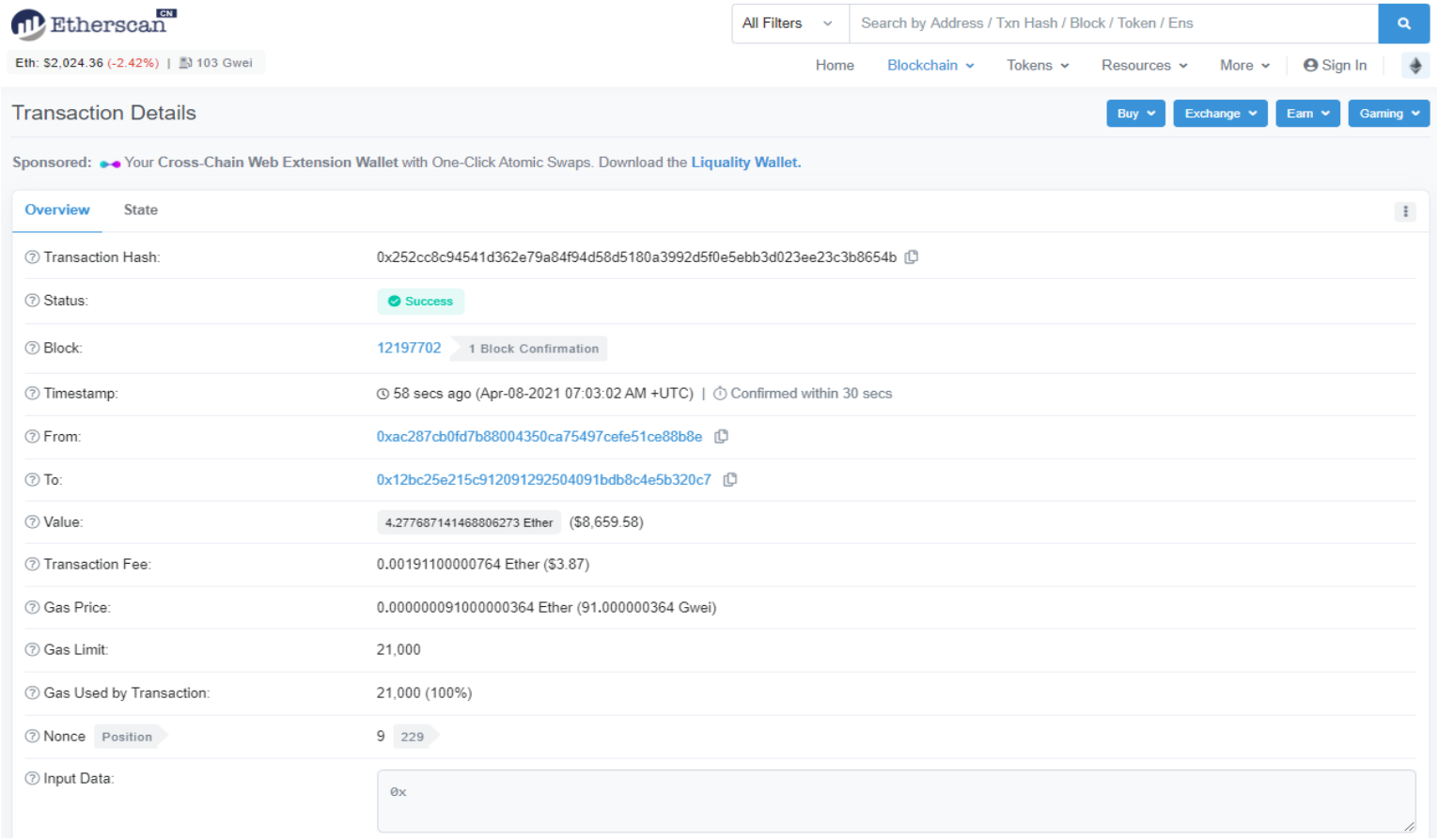
以太坊智能合约基本原理

- 以太坊交易数据结构
  - nonce：交易的编号，由发送方在交易前产生；
  - Gas：发送方支付的gas数量；
  - GasPrice：gas的价格；
  - from：交易发起方账户地址，可以是外部账户或合约账户；
  - to：接受账户地址，可以是外部账户或合约账户，如果是部署合约的交易，则此字段为空；
  - value：账户之间转移的以太币数量；
  - input：合约编译后被部署在EVM上的字节码，如果是普通交易则为空；
  - R、S、V：数字签名和交易的签名。



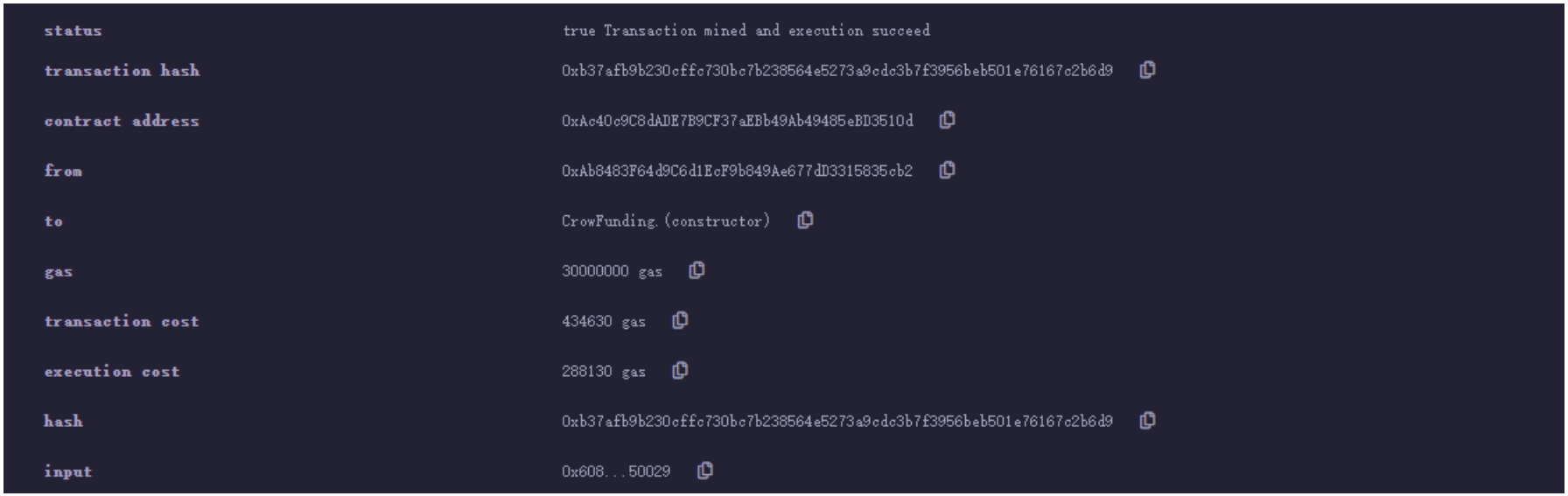
以太坊智能合约基本原理

- 普通交易



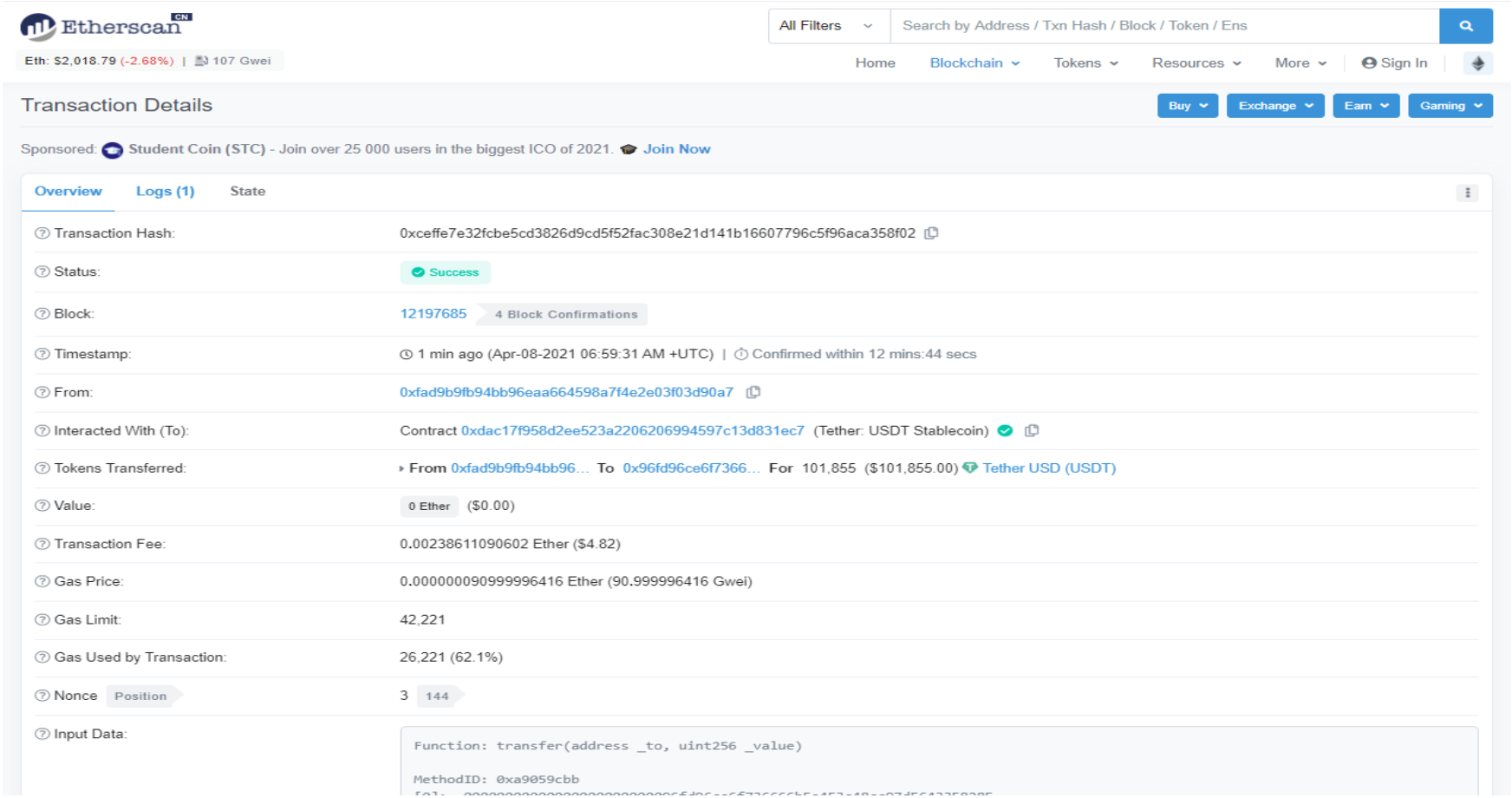
以太坊智能合约基本原理

- 部署智能合约交易
  - 合约部署实际上是发起了1笔交易，交易的接收者为空，以太坊将接收者为空的交易默认为是合约实例创建请求，这类交易中会携带当前部署合约的EVM字节码（ByteCode），部署成功的话会返回新建的合约账户。



以太坊智能合约基本原理

- 合约调用交易



以太坊DAPP

- 去中心化的应用程序(Decentralized App)：基于智能合约的应用。
- 典型构成：
  - 一个html界面
  - web3运行库
  - 一段JS代码
  - 部署在区块链上的一段智能合约
- DApp必须运行在一台能与以太坊节点交互的服务器上，或者任意一个以太坊节点上。
- DApp通过提交交易到区块链网络与对应的智能合约进行交互

以太坊DAPP

- 以太坊官网：
  - https://ethereum.org/zh/dapps/

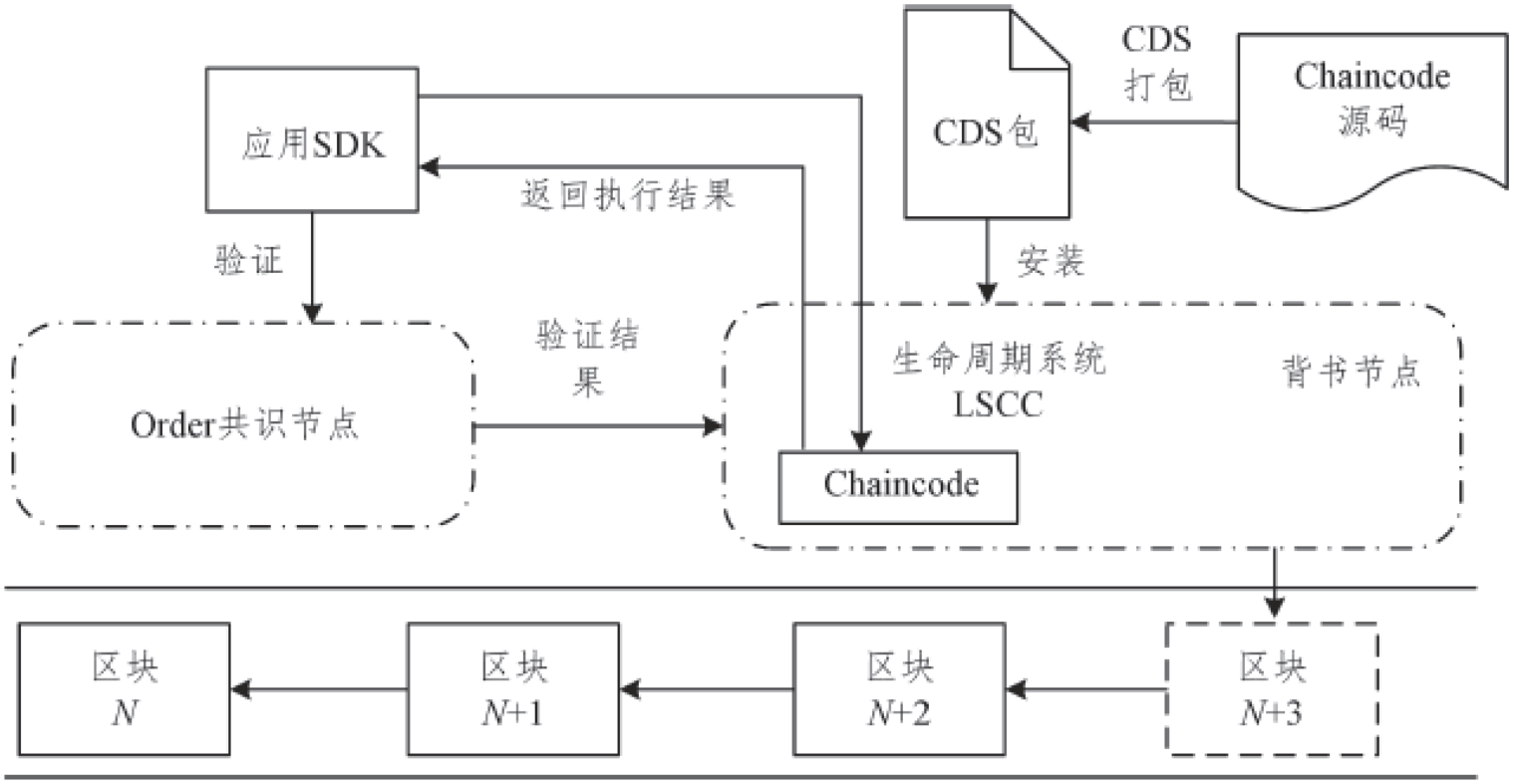
- 各区块链平台DAPP搜索
  - https://www.stateofthedapps.com/zh

超级账本智能合约

- 容器式运行
  - 超级账本的Chaincode 运行在一个受保护的Docker 容器中,是强制执行查询或修改超级账本数据状态的一段程序,可通过应用提交的交易对账本状态初始化并改变当前数据状态(Current State)，执行结果最终提交到网络中的每个账本节点。
- Chaincode 生命周期分为5 个阶段:
  - 打包(package)
  - 安装(install)
  - 实例化(instantiate)
  - 升级(upgrade)
  - 删除(delete).

超级账本智能合约

- Chaincode 的部署和执行流程



智能合约

- 以太坊和超级账本智能合约的对比

智能合约特性	以太坊平台	超级账本平台
执行环境	以太坊虚拟机(EVM)	Docker
编写语言	Solidity,Serpent,Mutan	Golang,Java
合约部署	作为交易广播到所有节点，通过矿工挖矿完成部署	直接在所有节点部署
合约升级	无法升级	v1.0 版本可以升级
合约间调用	可调用	许可后可调用
合约终止方式	计步/计价，引入 gas 消耗，对每一个执行指令都消耗 gas，消耗完毕强行终止	计时，以运行时间为标准来判定程序是否进入无限循环，超时后强行终止
库函数	少量对整数、字符串、JSON 等封装的库	Golang 库
加密货币	内置加密货币 Ether，可以利用合约交易加密货币或创建 Token	无内置加密货币，但可利用 Chaincode 创建 Token

智能合约

- 典型区块链应用与智能合约运行的环境

区块链系统	应用类型	智能合约运行环境	智能合约语言
Ethereum	通用应用	EVM	Solidity，Serpent，Mutan
Hyperledger	通用应用	Docker	Golang，Java
Bitcoin	加密货币	嵌入式运行	Golang，C++
Zcash	加密货币	嵌入式运行	C++
Quorum	通用应用	EVM	Golang
Parity	通用应用	EVM	Solidity，Serpent，Mutan
Litecoin	加密货币	嵌入式运行	Golang，C++
Corda	数字资产	JVM	Kotlin，Java
Sawtooth	通用应用	嵌入式运行	Python

智能合约

- 智能合约生态

平台	社区支持	技术文档	开发工具	合约的应用范围
嵌入式合约平台	除比特币外，其他社区支持较少	比特币文档丰富，Kadena只有英文文档	Btcdeb 调试工具 Pact Tool	一般为加密货币
虚拟机合约平台	社区活跃，支持者最多	中英文文档均丰富	Mist MetaMask Truffle Remix	通用合约，应用广泛
容器式合约平台	社区活跃，支持者多	中英文文档较丰富	Vscode go Docker	通用合约，偏隐私性