第一章 导论

1. 数字货币概述

   货币的本质：一种记账方式，资金的变化在支付体系中体现为数额的增减

   价值转移：一方减少的价值=另一方增加的价值　　信任危机的解决：技术手段代替第三方

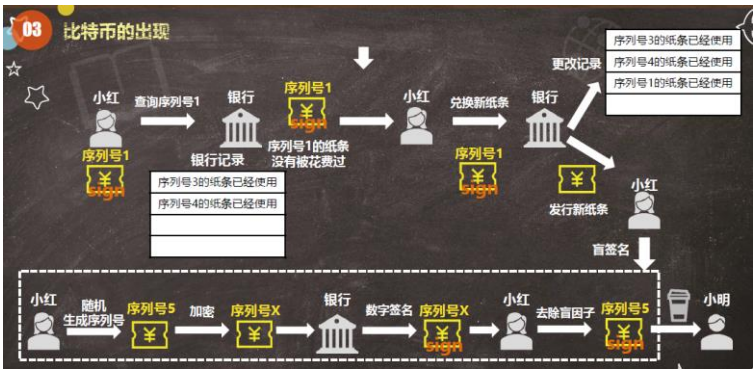
2. 盲签名

   消息发送方首先将消息盲化，让签名方对盲化的消息进行签名，然后消息发送方去除签名中的盲化因子，得到签名方对原消息的签名。

3. “双花”问题（“双重支付”问题）

   为数字货币所面临的最大问题：数据可以被复制，相当于伪造现金

   而盲签名可以解决加密货币面临的“双花”问题



4. 电子货币所需要解决的问题：

   稀缺性——确保货币值钱

   　　解决办法：电子黄金：发行机构储存实物黄金，根据黄金价格来发行电子货币

   　　哈希现金：花费时间求解数学题目，保证电子货币的稀缺性（工作量证明 Proof of Work）

   安全性

   　　解决办法：时间戳

   　　哈希指针，链式结构

   　　文件集合成块，按块连接，每个块的文件通过树状结构连接

5. 电子货币的发展

   B-Money 匿名的分布式加密货币系统，每个节点单独记录，独立维护账本。

   Bit Gold 工作量证明+时间戳，节点网络管理分类账，但是没有解决全网节点的账本同步问题。

   Bitcoin 与其他虚拟货币的区别：去中心化的特性。　　先代码，后理论

6. 区块链简介

   比特币系统

   　　实现点对点的比特币交易，保证交易数据不被篡改；

   　　交易的数据组成区块，每个区块通过链式结构形成交易总帐本；

   　　交易总帐本的内容可以在没有中心机构介入的情况下达成全网一致。

   分布式账本技术

   　　记账方式：集中式记账，分布式记账

   　　区块链技术：去中心化的分布式记账

   区块链

   　　一种多中心化基础架构与分布计算范式

   　　多种技术的集成：密码学+分布式系统+智能合约+……

7. 区块链的特点

   分布式对等结构

   数据公开透明

   数据不可篡改

   数据可追溯

8. 区块链的技术架构

9. 公有链、联盟链、私有链

公有链、联盟链和私有链的比较

	公有链 Public Blockchain	联盟链 Consortium Blockchain	私有链 Private Blockchain
参与者	任何个人或团体自由进出	联盟成员	仅限团体内部人员
共识机制	PoW/PoS/DPoS	PoS或PBFT	Raft
记账人	全网节点	联盟成员协商选定	内部自定义
激励机制	需要	可选	不需要
中心化程度	去中心化	多中心化	中心化
突出特点	信用的自建立	效率高，成本低	内部透明，安全性高
交易效率	慢	快	快
典型场景	数字货币	交易、结算等B2B场景	内部审计和数据库管理

公有链：优势：任何人都能查看数据；节点用户参与共识过程

劣势：依赖内建激励机制，容易受恶意用户攻击；交易确认效率低，交易费用高，资源消耗大，可扩展性差；脱离国家监管，实际应用受阻

联盟链：优势：交易验证和交易确认的效率高，良好的可扩展性

劣势：参与管理的节点数量远远少于接入节点数量，可能篡改数据

私有链：优势：可及时追踪错误来源；交易效率高，交易成本低；数据隐私性强

劣势：适用特殊的应用场景

第二章 区块链密码学基础

1. 密码

   通讯双方按已经设定好的规则对所要传输的数据信息进行特殊转换的一种重要保密手段。

2. 密码体制：五元组{M, C, K, E, D}

   明文空间M：全体明文的集合。

   密文空间C：全体密文的集合。

   密钥空间K：密钥用于控制加密和解密，在明文转换为密文或密文转换为明文的算法中作为输入参数。全体密钥的集合称为密钥空间。

   加密算法E：在密钥控制下对明文进行加密的算法。

   解密算法D：在密钥控制下对密文进行解密的算法。

3. 密码学分类

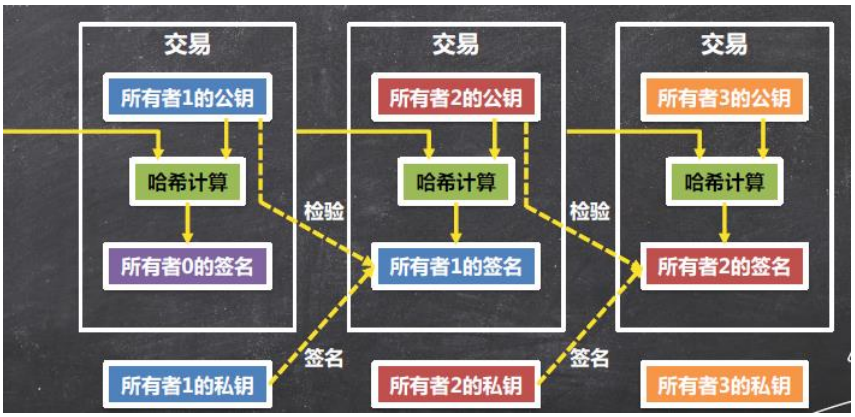
   对称密钥、公钥密码、混合密码（对称密码加密明文，公钥密码加密对称密码使用的密钥）

4. 交易回滚

   金融机构不承认本次操作交易行为产生的交易数据，将已发生的交易行为视为非法、交易数据视为无效，将交易发生后的状态恢复到交易前的状态。

5. 比特币基于密码学的交易模式

   密码哈希函数，数字签名；基于时间序列的链式结构



6. 哈希函数 & 密码哈希函数

   哈希函数：输入长度任意，输出哈希值长度固定

   密码哈希函数：

   特性：碰撞阻力：无法找到两个不同的输入值 x 和 y，使得 H(x) = H(y)

   隐秘性：给定 y = H(x)，给定 y，无法确定 x

   谜题友好性：对于任意n位的输出值y，假定k选自高阶最小熵分布，如果无法找到一个可行的方法，在比搜索 2^n 次少很多的时间内找到输入值x，使得$H(k||x) = y$ 成立，那么我们称哈希函数H为谜题友好。

   常见的密码哈希函数：MD4, MD5, SHA-1, SHA-2, SHA-3, RIPEMD-160, SM3

7. Merkle-Damgrad（MD）变换方法

   作用：将接受固定长度输入的哈希函数转换为可接收任意长度输入的哈希函数

   压缩函数：基础的、可用于固定输入长度、具有碰撞阻力的哈希函数称为压缩函数

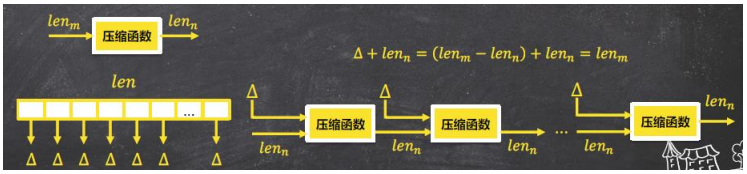
   变换步骤：

   压缩函数的输入固定长度为$len!$，输出长度为len''，$len'' < len!$，令 $\Delta = len! - len''$

   将任意长度的输入数据，按照 Δ 大小进行划分；通过补位把输入长度变成 Δ 的整数倍

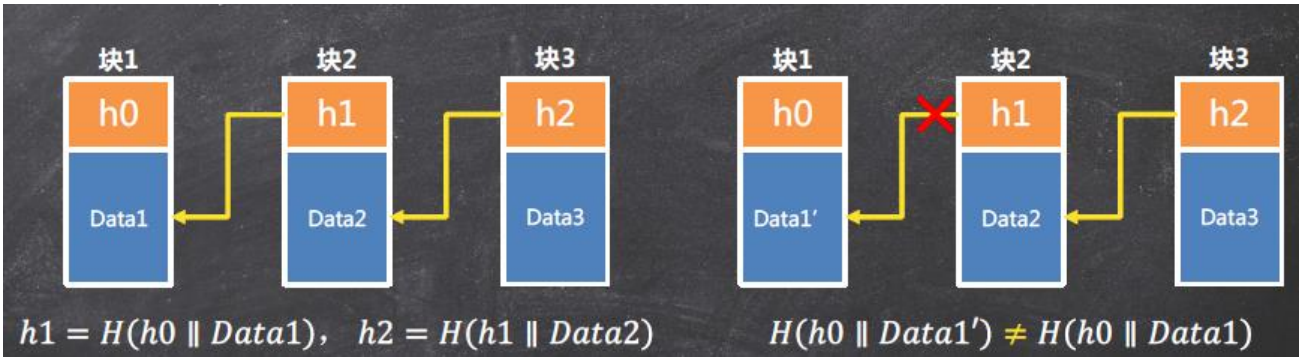
   将固定长度 Δ 的分块和前一个分块的输出len''一起代入压缩函数

   前一个分块的输出作为下一个压缩函数的输入之一，形成链式结构

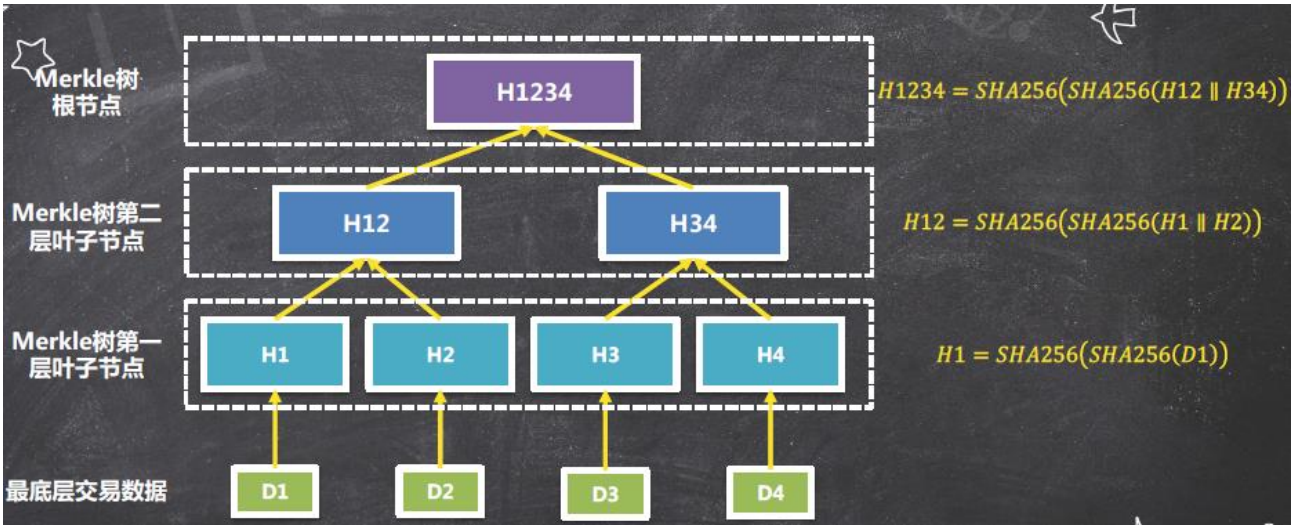


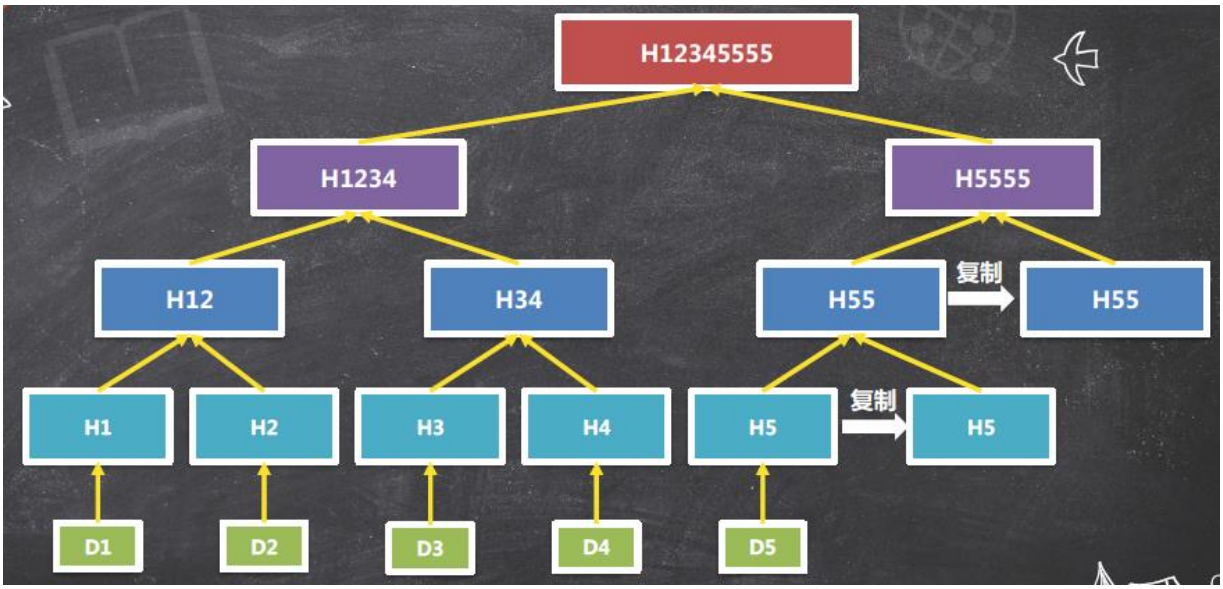
8. 公钥密码（略）、椭圆曲线密码（基于离散对数问题的公钥密码算法在相同安全级别下，密钥长度比 RSA 算法的密钥长度短）
9. 数字签名
   加密认证技术，签名数字化
   要求：他人无法仿造，签名与文件之间不能解耦合
   采用公钥密码设计理念
   密钥对：私钥——生成签名，公钥——验证签名
   - 流程：

   生成公私钥对: $(sk, pk) = createKeys(keyinfo)$
   签名：$sig = sign(sk, msg)$
   验证：$isValid = verify(pk, msg, sig)$
10. 椭圆曲线数字签名算法（Elliptic Curve Digital Signature Algorithm）
    比特币使用的数字签名算法，对 256 位的哈希值签名，输出 512 位的字符串
    基于椭圆曲线密码算法，通用标准的椭圆曲线方程为$y^2 = x^3 + ax + b$
11. 指针、哈希指针、链式结构
    指针：编程语言中的一个对象，直接指向机算机内存中存放数据的地址
    哈希指针：数据存储位置和数据哈希值。不是结构体，是一串数据的哈希值。可利用哈希指针构建不同的数据结构。
    链式结构：区块哈希指针计算：$h_1 = H(h_0 \| Data_1)$。防止数据篡改：密码哈希函数的碰撞阻力特性



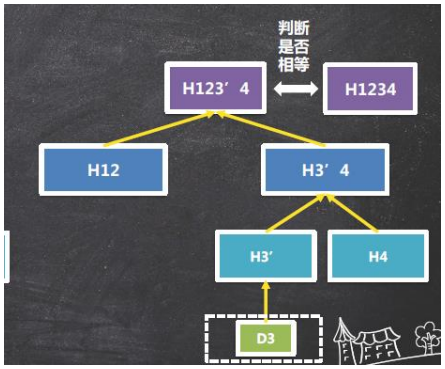
12. 区块数据
    区块头部+交易信息
13. 梅克尔树(Merkle Tree)
    利用哈希指针存储交易数据





梅克尔树作用：

（1） 高效验证交易：。假定哈希值长度为 32 字节，如果区块包含N个交易，为验证特定交易是否存在于区块里，节点只需要创建log_2N个 32 字节的哈希值，并将它们组成 Merkle 路径。



数据校验方向：从树底往上校验交易数据

（2） 防止数据篡改

（3） 提高区块链的运行效率和可拓展性

（4） 支持“简单支付验证”协议

14. 创世区块

比特币区块头包含的主要字段

字段	含义
Hash	本区块的哈希值，可以当作区块ID用于查询、防篡改使用
Previous Block	前一个区块的哈希值，即前序区块ID；创世区块中该字段为全0
Next Block(s)	后一个区块的哈希值，即后序区块ID
Merkle Root	梅克尔树根节点
Number of Transactions	交易数量，创世区块中只有一笔交易，即中本聪因为建立了第一个区块而获得50比特币
Timestamp	产生此区块的时间
Difficulty	挖矿难度，用于控制全网要投入多少算力来产生一个区块
Bits	挖矿难度指标，保证全网算力平均10分钟找到一个新的区块
Size	此区块的数据大小
Height	区块高度，从0开始，也可用于标识区块
Version	版本号
Nonce	使得挖矿运算结果满足难度要求的一个随机数，挖矿运算的目的就是找到这个数
Block Reward	区块奖励

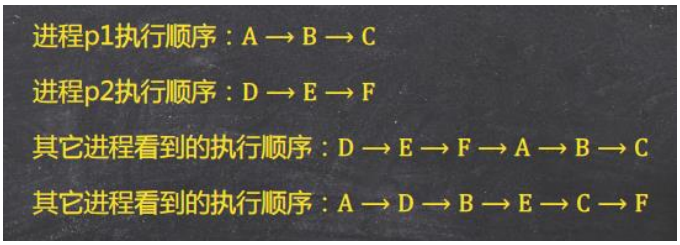
第三章 分布式系统核心技术

分布式系统

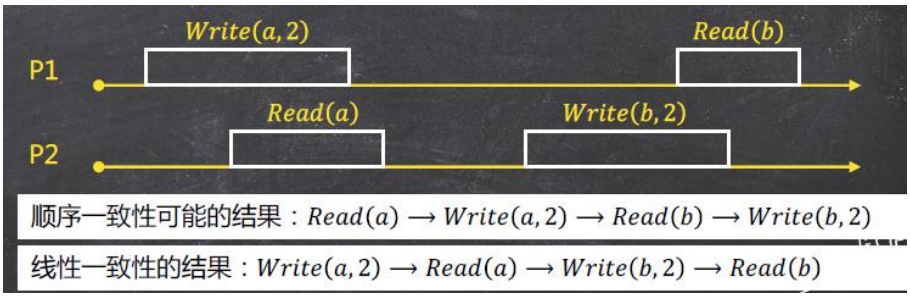
1. 分布式系统定义
   组件分布再计算机网络上
   组件之间仅仅通过消息传递来通信并协调行动
2. 分布式系统的特点
   分布性：分布式系统中的多台计算机在空间上随意分布，分布情况也会随意变动
   对等性：分布式系统中的计算机没有主从之分，既没有控制整个系统的主机。也没有被控制的从机，组成分布式系统的所有节点都是对等的
   并发性：在一个计算机系统中，程序运行过程中的并发性操作是非常常见的行为
   缺乏全局时钟：分布式系统是由一系列空间上随意分布的多个进程组成的，很难定义两个事件究竟谁先谁后
   故障总是发生：组成分布式系统的所有计算机都有可能发生任何形式的故障
   处理单点故障：在整个分布式系统中，如果某个角色或者功能只有某台单机在支撑，那么这个节点称为单点，其发生的故障称为单点故障。避免单点故障的关键在于把这个功能从单机实现变为集群实现。
3. 副本
   数据副本：指在不同的节点上持久化同一份数据，当某一个结点上存储的数据丢失时，可以从副本上读取到该数据
   服务副本：指多个节点提供同样的服务，每个节点都有能力接受来自外部的请求并进行相应的服务

一致性问题

4. 一致性问题定义
   一致性：对于多个服务节点，给定一系列操作，在约定协议的保障下，使得它们对处理结果达成“某种程度”的协同
   传统分布式系统中的一致性：对请求进行全局排序
   解决方案：将可能引发不一致操作的并行操作进行串行化
5. 一致性的要求
   可终止性：一致的结果在有限的时间内能完成
   约同性：不同节点最终完成决策的结果是相同的
   合法性：决策的结果必须是某个节点提出的提案
6. 带约束的一致性
   严格一致性：（很难实现）要求系统不发生任何故障且所有节点之间的通讯无需任何时间
   强一致性：
   （1）顺序一致性（因果一致性）：所有进程看到的全局执行顺序一致，并且每个进程看自己操作的顺序跟实际发生顺序一致。



   （2）线性一致性：要求系统看起来似乎只有一个数据副本，客户端的操作都是原子的，并且顺序执行；所有进程的操作似乎是实时同步的，而且跟实际发生顺序一致。



（3）弱一致性：在某些方面弱化的一致性

最终一致性：总会存在一个时刻，让系统达到一致的状态。

共识算法

7. 共识

   一致性 VS 共识

   一致性是多个副本对外呈现的状态

   共识是多个节点对某个事件达成一致看法的过程

8. 共识算法：分布式系统中大部分节点对于某个提案达成一致意见的过程

   确定性状态机模型：各个节点从相同的状态开始接收相同顺序的指令，则可以保证相同的结果状态。关键在于对多个事件的顺序进行共识，即排序。

9. 非拜占庭错误/故障错误：出现故障但是不会伪造信息

   拜占庭错误：伪造信息、恶意相应、拜占庭节点

10. 常见算法

    崩溃容错算法：性能较好，处理较快，能容忍不超过一半的故障节点

    拜占庭容错算法：

    确定性算法：一旦达成共识就不可逆转，即共识是最终结果

    概率类算法：共识结果是临时的，随着时间推移或者某种强化，共识结果被推翻的概率越来越小，最终成为事实上的结果。代表：PoW(Proof of Work)。性能较差，可容忍不超过 1/3 的故障节点。

11. FLP 不可能原理

    在网络可靠但允许节点失效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共识算法

    同步（可以很容易地判断消息是否丢失）

    系统中的各个节点的时钟误差存在上限

    消息传递必须在一定时间内完成，否则认为失败

    各个节点完成处理消息的时间是一定的

    异步（无法判断节点是否故障还是传输故障）

    系统中各个节点可能存在较大的时钟差异

    消息传输时间是任意长的

    各节点对消息进行处理的时间也可能是任意长的

12. CAP 原理

    一致性(Consistency)：任何事务应该都是原子的，所有副本上的状态都是事务成功提交后的结果，并保持强一致。

    可用性(Availability)：系统（非失败节点）能在有限时间内完成对操作请求的应答。

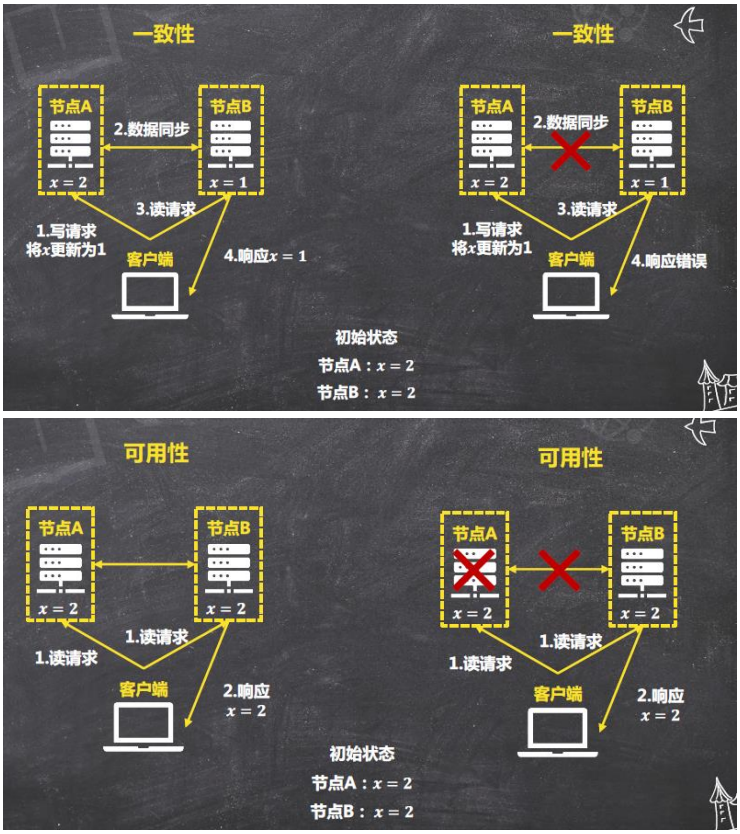
    分区容忍性(Partition)：系统中的网络可能发生分区故障（成为多个子网，甚至出现节点上线和下线），即节点之间的通信无法保障。而网络故障不应该影响系统正常服务。

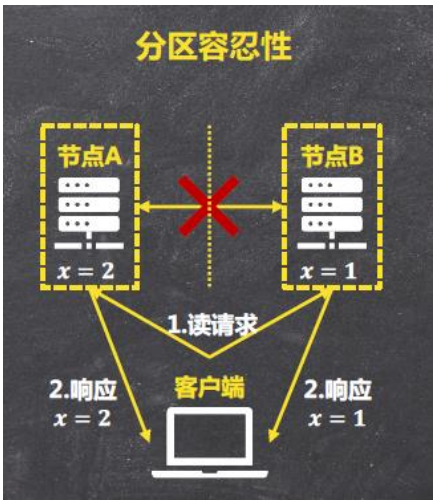
    应用场景：

    弱化一致性

    弱化可用性

    弱化分区容忍性





13. ACID 原则与多阶段提交

定义：

Atomicity（原子性）：每次事务是原子的，事务包含的所有操作要么全部成功，要么全部不执行。一旦有操作失败，则需要回退状态到执行事务之前。

Consistency（一致性）：数据库的状态在事务执行前后的状态是一致的和完整的，无中间状态。即只能处于成功事务提交后的状态。

Isolation（隔离性）：各种事务可以并发执行，但彼此之间互相不影响。按照标准 SQL 规范，从弱到强可以分为未授权读取、授权读取、可重复读取和串行化四种隔离等级。

Durability（持久性）：状态的改变是持久的，不会失效。一旦某个事务提交，则它造成的状态变更就是永久性的。

两阶段提交：将事务的提交分解为预提交和正式提交两个阶段，规避冲突的风险

预提交：协调者发起提交某个事务的申请，各参与执行者需要尝试进行提交并反馈是否能完成

正式提交：协调者如果得到所有执行者的成功答复，则发出正式提交请求。如果成功完成，则算法执行成功。

优点：简单容易实现

缺点：

同步阻塞问题：当参与者占有公共资源时，其他节点访问公共资源处于阻塞状态。

单点故障问题：较坏情况下可能一直无法完成提交

数据不一致问题：协调者和执行者在第二个阶段出现故障|

三阶段提交：对预提交阶段做进一步拆分：尝试预提交和预提交

尝试预提交：协调者询问执行者是否能进行某个事务的提交。执行者需要返回答复，但无需执行提交。这就避免出现部分执行者被无效阻塞住的情况。

预提交：协调者检查收集到的答复，如果全部为真，则发起提交事务请求。各参与执行者需要尝试进行提交并反馈是否能完成。

正式提交：协调者如果得到所有执行者的成功答复，则发出正式提交请求。如果成功完成，则算法执行成功。

Paxos 算法与 Raft 算法

14. Paxos 算法

Paxos 问题：分布式系统中存在故障，但不存在恶意节点的场景下的共识达成问题

三种逻辑角色

提案者（Proposer）：提出一个提案，等待大家批准（Chosen）为结案（Value）。通常由客户端担任该角色。

接受者（Acceptor）：负责对提案进行投票，接受（Accept）提案。通常由服务端担任该角色。

学习者（Learner）：获取批准结果，并帮忙传播，不参与投票过程。可为客户端或服务端。

约束要求

安全性（saftety）：保证决议结果是对的，无歧义的，不会出现错误情况。

有是被提案者提出的提案才可能被最终批准；

在一次执行中，只批准一个最终决议。被多数接受的结果成为决议。

存活性（liveness）：保证决议过程能在有限时间内完成。

决议总会产生，并且学习者能获得被批准的决议。

单个提案者+多个接受者

提案者只要收到了来自多数接受者的投票，即可认为通过。

一旦提案者故障，则整个系统无法工作。

多个提案者+单个接受者

接受者收到多个提案，选第一个提案作为决议，发送给其它提案者即可。

容易发生单点故障，包括接受者故障或首个提案者节点故障。

多个提案者+多个接受者

同一时间段（如一个提案周期）内只有一个提案者：退化到单提案者的情形。

允许同一时间片段内可以出现多个提案者：给提案编序号。

两阶段提交：

准备阶段（Prepare）

提交阶段（Commit）

15. Raft 算法

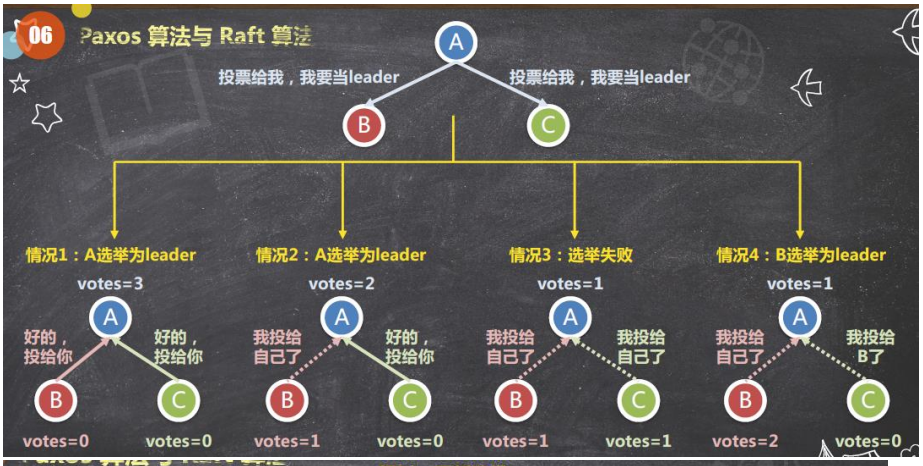
用于管理日志一致性的协议

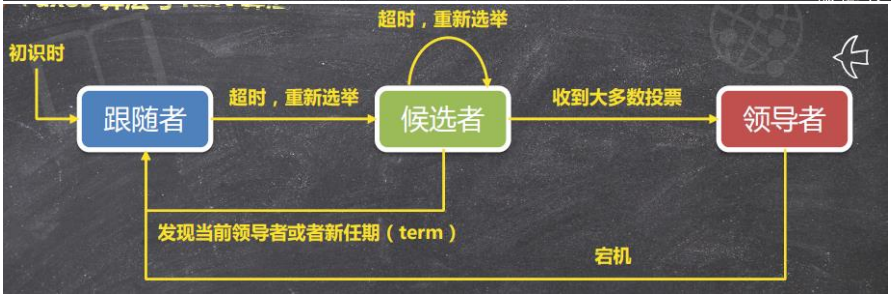
三种逻辑角色：领导者（leader）、候选者（candidate）、跟随者（follower）

基本思想：通过先选出领导节点来简化流程和提高效率

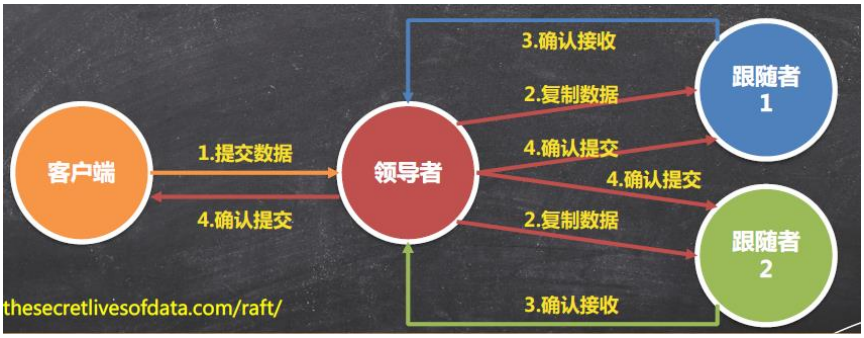
共识过程：

领导选举:：开始所有节点都是跟随者，在随机超时发生后未收到来自领导者或候选者消息，则转变角色为候选者（中间状态），提出选举请求。最近选举阶段（Term）中得票超过一半者被选为领导者；如果未选出，随机超时后进入新的阶段重试。领导者负责从客户端接收请求，并分发到其他节点。





同步日志：领导者会决定系统中最新的日志记录，并强制所有的跟随者来刷新到这个记录，数据的同步是单向的，确保所有节点看到的视图一致。



重新选举：领导者定期向所有跟随者发送心跳信息，跟随着如果发现心跳信息未收到，则可以认为领导者已经下线，尝试发起新的选举。

任期：任期用连续的数字进行表示。每一个任期的开始都是一次选举。如果一个候选者赢得了选举，则在该任期的剩余时间担任领导者。在某些情况下可能选不出领导者，则会开始另一个任期，并且立刻开始下一次选举。Raft 算法保证在给定的一个任期最多只有一个领导者。

拜占庭问题与算法

16. 两将军问题

    两个将军要通过信使来达成进攻还是撤退的约定，但信使可能迷路或被敌军阻拦（消息丢失或伪造），如何达成一致?

    根据 FLP 不可能原理，此问题无通用解。

17. 拜占庭问题

    假如节点总数为N，故障节点数为F，则当$N \geqslant 3F + 1$ 时，问题才有解

18. 实用拜占庭容错算法

    可在恶意节点不超过总数的 1/3 的情况下同时保证安全性和存活性

    采用 RSA 签名算法、消息验证编码和无碰撞哈希函数等密码学技术确保消息传递过程中无法被篡改和破坏

    概念:

    客户端（client）：向主节点发起请求的客户端节点。

主节点（primary）：在收到客户端请求后分配序号，排好序后广播。
备份节点（replica，也称副本节点）：接收广播消息，验证请求合法性，投票，触发视图切换（view change）协议来推举新的主节点。
视图（view）：通过轮换或随机算法选出某个节点为主节点，此后只要主节点不切换，则称为一个视图（View），它描述了一个多副本系统的当前状态。节点在同一个视图上对数据达成共识。
流程：

客户端向主节点发送请求。
主节点通过广播将请求发送给其他副本。（广播过程包括：预准备、准备、提交）
所有副本都执行请求并将结果发回客户端。
客户端需要等待$f + 1$个不同副本节点发回相同的结果，作为整个操作的最终结果。

可靠性指标

19. 可靠性指标
    可靠性指描述系统可以提供服务能力的重要指标（完美的可靠性不存在）。
    概率意义上粗略反应了系统提供服务地可靠性。
20. 两个核心时间
    平均故障间隔时间：即系统可以无故障运行的预期时间
    平均修复时间：发生故障后，系统可以恢复到正常运行的预期时间
21. 提高可靠性的方法
    增强系统中单个组件的可靠性
    消灭单点

第四章 比特币

简介

1. 发明人：中本聪
2. 特点：去中心化、匿名性、通胀预防

密钥对、地址和钱包

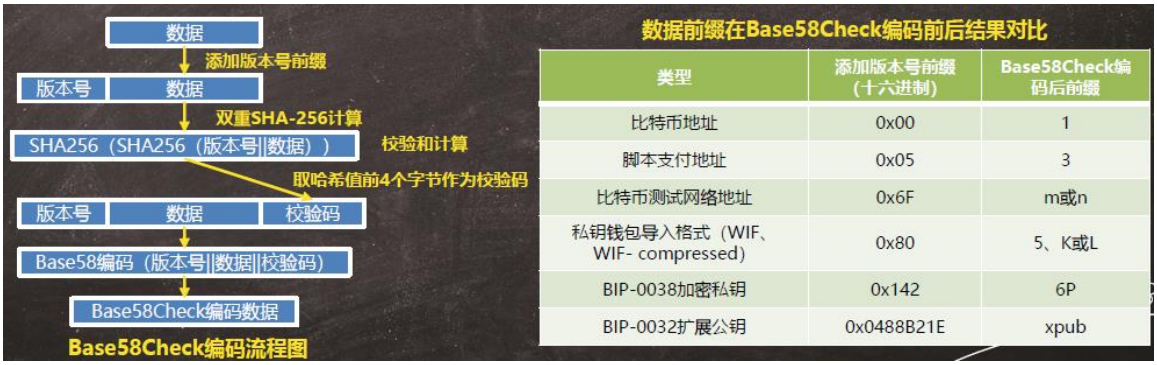
3. 比特币密钥对

私钥：

普通用户一般使用比特币钱包客户端生成，用随机生成的 256 位二进制数作为私钥——私钥空间大小为 2^{256}

表示方法为 Hex, WIF, WIF-compressed

表示方式	前缀	概述	同一私钥的不同编码
Hex	无	64位十六进制数	1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edc c32c8ffc6a526aedd
WIF	5	Base58Check编码，包含版本前缀0x80和32位二进制校验码好的Base58编码	5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2Jpbnkeyhfs YB1Jcn
WIF-compressed	K或L	方法与WIF相同，在编码之前添加后缀0x01	KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6Ywgd GWZgawvrtJ



数据前缀在Base58Check编码前后结果对比

类型	添加版本号前缀（十六进制）	Base58Check编码后前缀
比特币地址	0x00	1
脚本支付地址	0x05	3
比特币测试网络地址	0x6F	m或n
私钥钱包导入格式（WIF、WIF- compressed）	0x80	5、K或L
BIP-0038加密私钥	0x142	6P
BIP-0032扩展公钥	0x0488B21E	xpub

公钥：

非压缩格式公钥：

公钥是椭圆曲线上的一个点（$\boldsymbol{x}$，$\boldsymbol{y}$），$\boldsymbol{x}$和$\boldsymbol{y}$分别是一个 256 位的二进制数；可用十六进制表示。

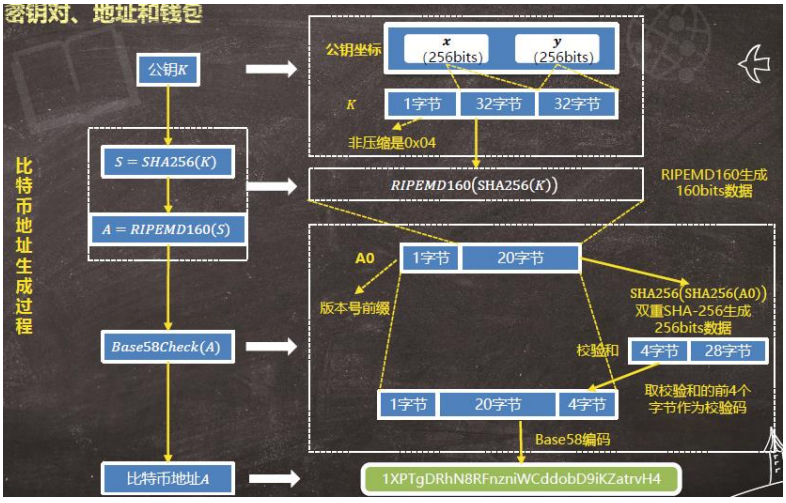
非压缩格式公钥K = 04 ‖ $\boldsymbol{x}$ ‖ $\boldsymbol{y}$

压缩格式公钥：

y 坐标可以由 x 坐标计算出，只需保存 x 坐标（$y^2 \bmod p = (x^3 + 7) \bmod p$）

地址：

由一串从公钥转换而来，由数字和字母随机组成的字符串。由公钥通过密码哈希函数 SHA-256 和 RIPEMD160 计算并编码而来



4. 比特币钱包（用于保护用户私钥的工具，将私钥存放在钱包中）

分类：

（根据钱包存放位置分类）

移动钱包：在移动终端上运行的钱包

在线钱包：通过网页浏览器浏览第三方服务器来操作和使用钱包

硬件钱包：可以产生私钥的小型硬件设备

脑钱包：把私钥记在大脑中

纸钱包：把私钥打印在纸上

冷钱包：从未在比特币网络中储存过，不与网络连接，又称离线钱包。

热钱包：钱包作为节点接入比特币网络，必须保持与网络的连接，又称为在线钱包

（根据私钥生成方式分类）

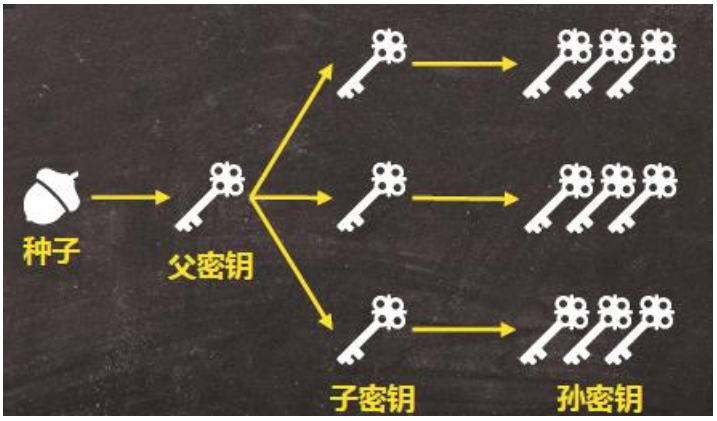
随机钱包：随机生成的私钥的集合（大量私钥难以管理、备份及导入导出钱包）

种子钱包：随机生成的数字结合其他信息，可以衍生出很多私钥

层次化种子钱包（HD 钱包）

存储树形结构的密钥集。一个种子生成一个父密钥，再由父密钥派生出若干子密钥，而每个子密钥又派生出若干孙密钥，依此类推。

同一层级的密钥之间是没有任何关联的，不能由一个密钥推算出另一个密钥。



比特币交易

5. 定义：

广义：某用户授权将自己地址里的一定数量的比特币资产转移到另一个用户地址中，告诉全网并得到全网认可。

狭义：将比特币资产从交易输入转移到交易输出

交易输入：指明了这笔比特币资产的来源，通常是上一笔关联交易的输出

交易输出：知名了这笔比特币资产的去向，这笔比特币资产的新拥有者可以通过另一笔授权将此次获得的比特币资产转移给其他人

6. 流程：

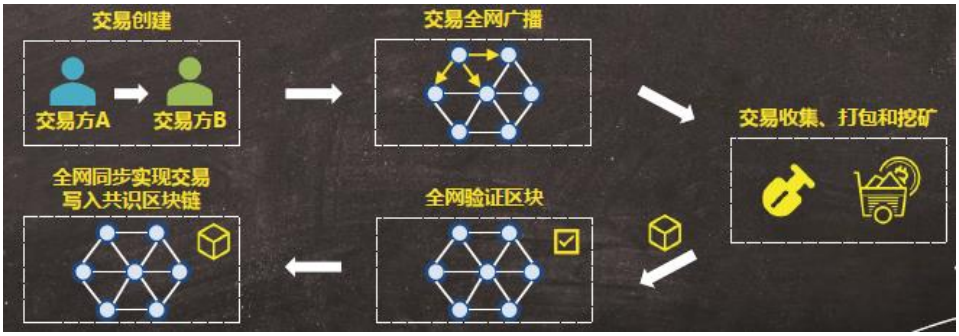
交易创建

交易全网广播

交易收集、打包和挖矿

全网验证交易

全网同步实现交易写入共识区块链



7. 币基交易
   比特币系统给挖矿成功的节点输出比特币
   币基交易是每个新产生区块的第一笔交易，交易输入中没有上一笔关联交易信息
   交易中的比特币来源于系统发放的区块奖励和交易费，交易输出指定了矿工接收比特币的地址
   比特币系统中除了币基交易之外，其他的交易都可被称为普通交易。

8. 交易创建
   交易可以由任何人通过比特币客户端在线或者离线生成。
   Key:确定有足够的比特币资金所有权，构成交易并签名。
   每一笔独立的交易都包括交易输入和交易输出以及该笔交易的哈希值。
9. 交易全局广播
   广播交易必须在线。
   网络中的节点只要接收到新的有效交易，都会转发给附近的节点。
   节点在继续转发之前独立验证交易，确保交易的有效性。
10. 交易收集、打包和挖矿
    矿工节点收到交易后，既要验证交易的有效性，还要将有效的交易加入自身维护的交易池。
    矿工节点从交易池中按照特定的优先级选择若干交易，并打包为一个临时区块，然后通过挖矿来竞争区块记账权。
11. 全网验证区块并确认交易
    验证区块头部哈希值是否符合工作量证明要求。
    验证区块包含的每笔交易的有效性。
    通过验证的区块被节点采纳，节点把区块的挂接在自己存储的区块链上。
12. 全网同步实现交易写入共识区块链
    交易需要经过 6 个区块的确认才能认定该区块是永久挂链的。
    确认的区块越多，交易的可信度就越高。
13. 比特币的交易结构
    由交易输入和交易输出构成交易的输出和输出与用户的真实身份没有关联
    交易输入：指明了比特币资金的来源，并包含本次交易的签名
    交易输出：指明了比特币交易的金额和资金去向，并被新拥有者的私钥锁定

    UTXO：未花费的交易输出，是比特币交易中不可拆分的基本结构单元。除了币基交易，每一笔交易输入中必然引用前序交易的 UTXO。
    UTXO 只要没有被后面的交易引用，就一直是未被花费状态，存储在 UTXO 列表中。
    核验交易时，首先根据交易输入追溯到上一个或上几个 UTXO 是否合法存在，同时查验当前交易的付款方是不是对应引用的 UTXO 的收款方，从而确保交易输入有效。
    交易写入区块链后，交易的 UTXO 也被记入区块链。
    所有未花费的输出构成当前整个比特币网络的 UTXO 列表。
    确认交易的付款方是否有足够的比特币支付当前交易时，节点只需核验对应的 UTXO 即可，不需要从账本建立之初的交易开始核查。

    交易输入主要包括交易哈希值、输出索引、解锁脚本大小、解锁脚本以及序列号
    解锁脚本：
        输出地址对应的公钥以及用该公钥所对应的私钥对交易的签名

比特币交易输入结构

字段	含义
交易哈希值（Transaction Hash）	指向上一笔包含待花费的UTXO的交易哈希指针
输出索引（Output Index）	指向上一笔待花费的UTXO的索引值
解锁脚本大小（Unlocking Script Size）	解锁脚本长度（以字节为单位）
解锁脚本（Unlocking Script）	满足UTXO锁定脚本中解锁条件的脚本
序列号（Sequence Number）	交易替换功能，目前暂未使用，设置为0xFFFFFFFF

交易输出主要包括金额和锁定脚本。

锁定脚本：

把交易输出锁定在收款人的比特币地址上，也就是将比特币的所有权转移给新的所有者

锁定脚本约束收款人只能通过该地址中公钥对应的私钥签名才能花费被锁定的比特币资产。

比特币交易输出结构

字段	含义
金额	比特币价值（以“聪”为单位）
锁定脚本大小（Locking Script Size）	锁定脚本的长度（以字节为单位）
锁定脚本（Locking Script）	定义交易输出的锁定脚本

币基交易只有单一输入和单一输出

不花费上一个关联交易输出的比特币，没有哈希指针指向上一个关联交易。

交易输出金额包括新发行的比特币和当前区块所有交易的交易费。

交易输入里有一个特殊的字段，称为币基参数。其主要作用是通过矿工写入不同的随机数来调整区块头部数据，继而进行挖矿。

币基交易输入结构

字段	含义
交易哈希值（Transaction Hash）	所有位固定为0，表明不是一个交易哈希指针
输出索引（Output Index）	所有位固定为1，即0xFFFFFFFF
币基数据大小（Coinbase Data Size）	币基数据的长度，2～100字节
币基数据（Coinbase Data）	即币基参数，可以是任意内容
序列号（Sequence Number）	设置为0xFFFFFFFF

14. 比特币的交易脚本

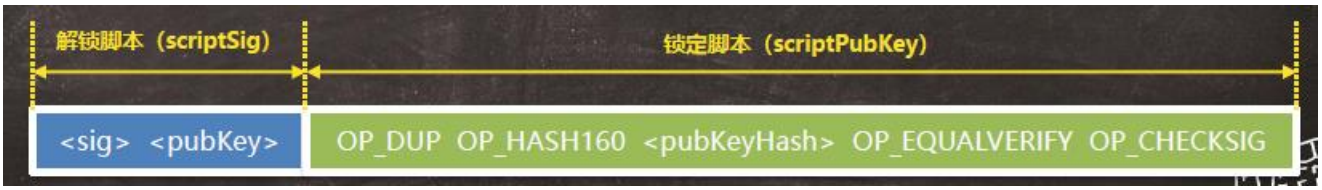
解锁脚本和锁定脚本统称为比特币脚本，比特币交易通过执行脚本来实现

锁定脚本（位于交易输出）——又称为脚本公钥

把 UTXO 锁定在收款人的比特币地址上，只有满足条件才能花费 UTXO

解锁脚本（位于交易输入）——通常包含一个数字签名，又称为脚本签名

只要满足上一笔关联交易输出 UTXO 的锁定脚本的受限条件，引用的 UTXO 就能被再次花费



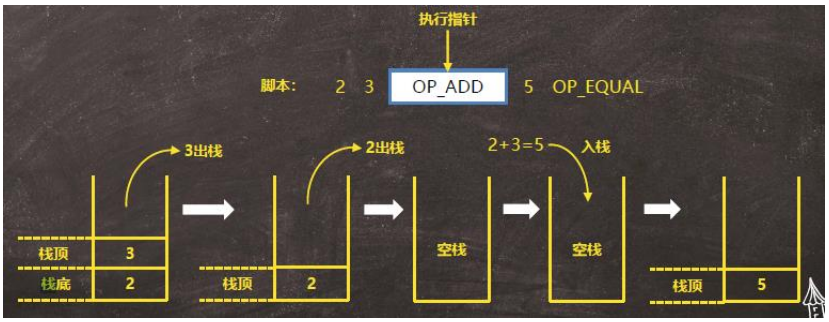
一笔交易广播后，节点同时运行锁定脚本和解锁脚本以验证交易的有效性。自动化执行校验，不需要人工参与。

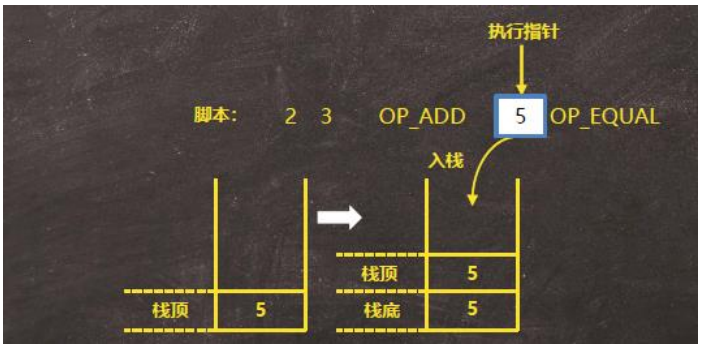
15. 脚本语言——基于栈的执行语言

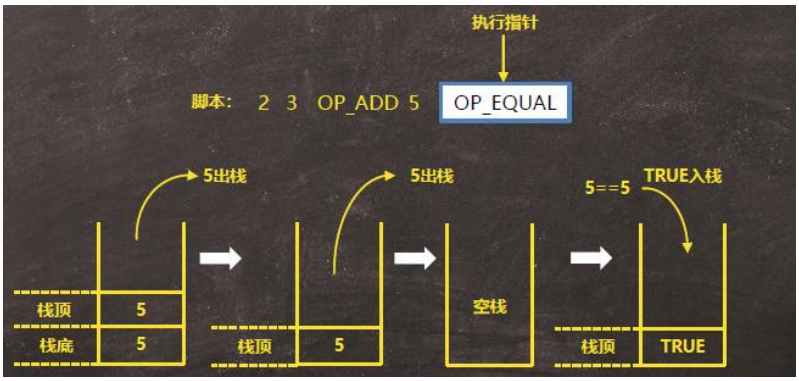
栈的基本操作——入栈出栈，数据先进后出

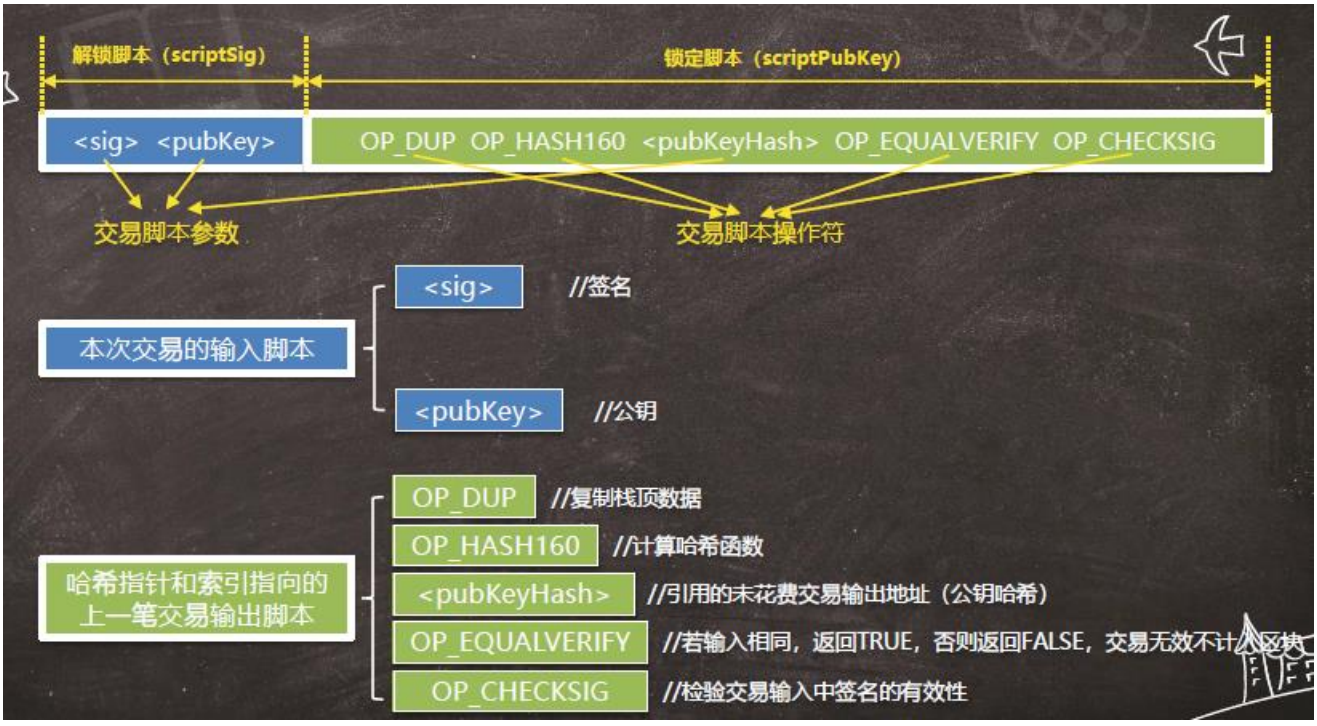
比特币脚本由参数和操作符构成，通过从左到右依次处理每个条目（包括参数和操作符）来执行脚本

一个例子：









16. 比特币标准交易脚本
    P2PKH(为主要脚本), P2PK, MS, P2SH, OP_RETURN

比特币网络

17. 点对点网络(P2P Network)

网络中的所有节点都处于对等的地位，没有主从之分；节点通过 P2P 协议实现点对点通讯并通过特定协议实现相互之间资源的共享。

分类：

集中式 P2P（存在索引服务器）、纯分布式 P2P（节点完全对等）、混合式 P2P（节点分为超级节点和普通节点）、结构化 P2P（节点按照一定的规则进行连接）

特点：

去中心化、扩展性、健壮性、私密性、高性能

18. 节点类型和作用

节点功能：

网络路由：所有节点都应该具有的功能，接收和转发交易以及区块数据

区块链数据库：用于分布式存储数据，独立验证收到的交易是否有效

钱包：保存用户控制的所有私钥，同时能查看用户比特币资产、发起交易、查看历史交易记录

挖矿：矿工节点收集交易，挖矿成功获得交易的记账权，进而可把交易最终写入区块链数据库，完成对交易的确认

节点类型：

比特币核心节点（标准客户端）(完全节点，骨干)

参与全网的路由功能，发现和维持与其他节点之间的连接；验证并传递交易和区块数据；通过挖矿获取记账权，将交易记入区块链数据库；核心节点还具备钱包功能，提供比特币交易工具。

完全区块链节点(完全节点，骨干)

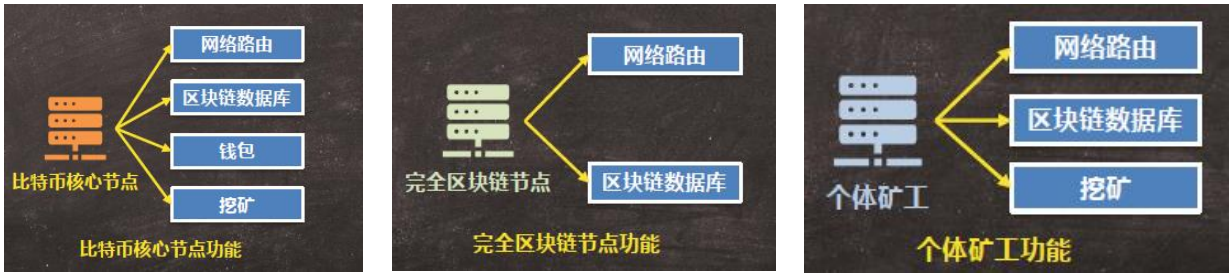
存储有最新、最完整的区块链数据，能独立地完成对所有交易数据的校验，不需要借助外部其他节点的数据或信息；可获取其他节点发来的交易数据以及具有挖矿功能的节点发出的区块数据，交易验证后继续广播，区块验证后写入本地区块链数据库。主要用于处理如查询资产之类的业务操作

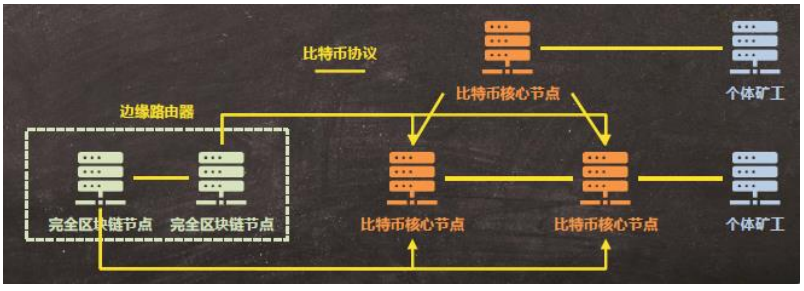
可组成“边缘路由器”。开发人员可基于完全区块链节点构建顶层的应用服务，从而实现资产查询、交易支付处理、交易机构搭建、区块链信息调阅等功能。

个体矿工(完全节点，骨干)

主要任务是“挖矿”。交易生成并在全网广播，每个个体矿工都会打包交易，同时增加一笔自己收款的币基交易在里面。打包完成后构成一个区块，矿工通过挖矿竞争记账权，第一个成功挖出符合工作量证明要求的区块的个体矿工会把新区块添加到自己本地的区块链副本顶端，同时把这个新区块通过网络发送给其他节点，其他节点接收后验证区块的真实性和准确性，如果验证通过则添加该区块到自己的区块链副本顶端

个体矿工需维护一个完整的区块链副本。个体矿工接收到新区块会立刻验证区块头部及其包含的交易，验证通过则更新本地区块链副本，并放弃当前的挖矿工作，重新打包交易进入下一轮挖矿竞赛。个体矿工一般通过比特币协议与比特币核心节点相连。





轻量级(SPV)钱包

具有钱包和网络路由功能。该类节点没有完整区块链副本，需要给具有区块链副本的节点发送查询消息，获取交易所在区块的相应梅克尔路径，使用“简单支付验证”（Simplified Payment Verification, SPV）方法来验证交易是否成功，从而确认向自己付款或自己支付的交易是否成功。适合运行在存储和计算资源有限的设备上。

矿池协议服务器

在集中式矿池网络中存在一个矿池协议服务器，矿工与其连接。矿池矿工和矿池协议服务器之间的通信采用 Stratum 协议或其他矿池挖矿协议，矿池协议服务器使用比特币协议与其他比特币节点通信。矿池协议服务器分为两类，一类是 Stratum 服务器，另一类是 Pool 服务器

Stratum 矿池：Stratum 矿工通过 Stratum 协议连接到 Stratum 服务器，构成 Stratum 矿池。Stratum 矿池是一种托管矿池，Stratum 矿工加入其中共同挖矿，一旦有一个矿工挖到矿，比特币奖励就会发到矿池地址，当奖励达到一定规模，矿池按照每个矿工的算力贡献来分配比特币收益。

Pool 矿池：将 Pool 服务器的功能去中心化，实现了一个类似比特币区块链但难度较低的份额链（Share Chain）。Pool 矿工以每 30 秒产生一个份额区块的速度在份额链上挖矿并获得份额，每个区块记录 Pool 矿工的算力贡献，并继承之前份额区块上的算力贡献。当某个份额区块达到比特币网络难度目标时，这个区块就被传播到比特币网络上，然后根据每个矿工对份额区块的算力贡献来分配区块奖励。

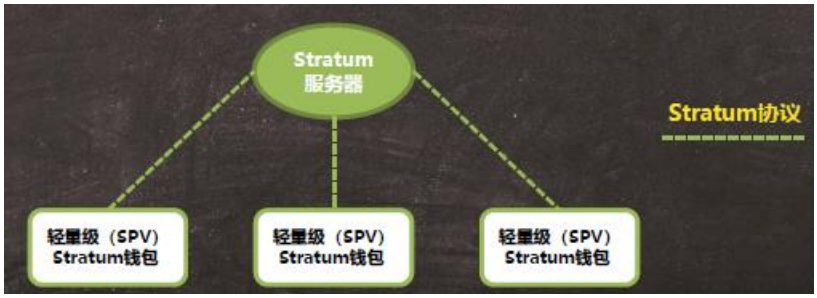
矿池矿工

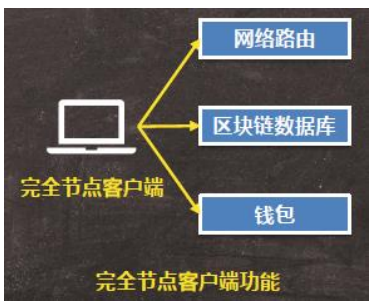
矿池矿工具备挖矿功能，但不具备网络路由功能，也没有完整区块链数据，所以该类节点既不能独立加入比特币网络，也不能验证交易，只能通过加入矿池，依赖矿池协议服务器来发挥作用，所以该节点需要运行 Stratum 协议或其他矿池挖矿协议与相关服务器建立联系。

矿池矿工主要分为两大类，即 Stratum 矿工和运行其他矿池挖矿协议的矿工（主要指 Pool 矿工）。矿池矿工与矿池协议服务器通过相关的矿池协议相连。个体矿工只有自身挖到矿后才能获得比特币收益，但加入矿池的每个矿工只需要一起协作挖矿，就能共享比特币收益，从而获得更稳定的挖矿收益。

轻量级(SPV)Stratum 钱包

钱包功能，并运行 Stratum 协议。不具备网络路由功能，所以不能直接与比特币网络相连，只能通过 Stratum 服务器实现接入，从而间接实现与比特币网络的连接，进而实现钱包功能。轻量级（SPV）Stratum 钱包和轻量级（SPV）钱包，又称为 SPV 节点或轻量节点（Lightweight Node）。





完全节点客户端

具备网络路由、区块链数据库和钱包功能。该节点不从事挖矿，主要功能是管理用户的钱包，可以独立验证交易。它作为 Stratum 服务器接入比特币网络的接入端，同时也为 SPV 节点提供区块链数据查询服务。

19. 网络发现与同步

网络同步

通过握手协议交换“握手”信息

地址传播与发现

节点的重启

节点的保活

区块的同步

完全节点类型的新节点成功加入比特币网络后，从比特币网络中下载从创世区块到当前最新区块的所有区块，在本地构建一个完整的区块链副本

比特币的共识机制

20. 共识

分布式系统中大部分节点对某个提案达成一致意见的过程

拜占庭节点：伪造信息、恶意响应

共识成本：

存在一定的信任前提时，达成共识相对容易，共识性能也较高

不存在信任前提时，需要付出相对较大的成本，而且性能往往较差

比特币解决拜占庭问题的方法：

限制一段时间内整个网络中出现提案的个数（通过工作量证明来增加提案成本）

去除最终的确认约束，约定好始终沿着已知最长的链进行扩展

21. 去中心化共识

比特币没有中心化机构，每个参与者都自发遵循共同的规则来维护区块链账本

22. 工作量证明(Proof of Work, PoW)

根据矿工占全网计算能力的比例来决定矿工解题成功的概率

矿工将包含临时随机数的区块头部数据代入计算来尝试解题

解题成功的概率与矿工的计算能力相关——具有谜题友好性的密码哈希函数

矿工通过不断地调整 nonce 值来改变密码哈希函数地输入

目标区域：小于或等于一个目标值（目标难度）

系统可以通过设置目标区域来调整谜题的难度

谜题的目标难度理论上随着当前网络上挖出区块的平均时间而动态变化，控制后续区块的产生时间维持在 10 分钟左右。

计算：

比特币系统的最大目标值：0x00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff ——难度为 1

难度值计算公式：$$Difficulty = \frac{difficulty\_1\_target}{current\_target}$$

$difficulty\_1\_target$：难度为 1 的目标值　　$current\_target$：当前目标值，决定了区块生成难度

- 比特币出块时间估算：全网难度 $\times 2^{32}$ / 全网算力 = 出块时间（秒）

目标值调整：根据全网算力调整

$$target2 = target1 \times \frac{time1}{time2}$$

- $target1$：当前目标值
- $target2$：要调整到的目标值
- $time1$：最近2016个区块产生的时间
- $time2$：2016*10分钟，预订的出块时间

出块时间设定为10分钟左右

$$time = \frac{10}{ratio}$$

- $ratio$：单个矿工节点的算力占全网计算能力的比例
- $time$：矿工节点发现下一个区块的平均时间

23. 去中心化的共识过程

Step 1:拥有完整区块链副本的节点对每个交易进行独立的验证，并将有效交易广播给全网。

交易的有效性？是否双重支付？交易是否被本节点接收过？脚本是否符合规范？

交易池：节点维护的一个临时的未确认交易列表，也称为内存池。

·已创建并发布在网络上、通过交易验证但尚未被写入区块链的交易，都会被接收的节点放入各自的交易池中

·孤儿交易池：存放造势找不到所引用的前序交易的交易

·当新交易被纳入交易池时，节点首先检查孤儿交易池，判断里面是否有引用该交易的输出的孤儿交易，一旦发现有孤儿交易引用该交易的输出，则立刻将该孤儿交易从孤儿交易池中移至交易池，同时搜索孤儿交易池，检查是否有孤儿交易引用这个不再是孤儿的交易的输出，以此类推，直至补全整条交易链。

Step 2:竞争区块记账权，将交易纳入新区快并全网广播

交易纳入新区块：各个矿工节点按照一定的优先级规则，从交易池中选出尚未被记入区块链的有效交易构建一个空区块（即“候选区块”）。只有当矿工节点成功计算出符合要求的区块哈希值从而获得记账权，并且被全网其他节点认可，该区块才变为有效区块。

构造币基交易：由每个矿工自行构造，区块中的第一个交易，用来给矿工发奖励，只有获得记账权的矿工所构造的币基交易才有效。挖矿奖励包括区块奖励和新区块里所有的交易费。比特币系统通过奖励的方式将新的比特币投放到流通领域。

构造区块头部：版本号，前序区块哈希值，梅克尔树根，时间戳，找到区块的难度（点数），临时随机数

全网广播被挖出来的区块：挖矿成功的矿工节点立即将构建好的区块发送给与之对等的邻居节点，这些节点接收并验证该区块，验证通过后继续发送给与它们对等的邻居节点，这样新区块呈波浪状广播至全网。

Step 3:节点独自校验新区块，并更新本地区块链副本

验证区块：

区块数据结构是否符号语法要求

区块头部的哈希值小于或等于目标难度，以此来确认是否满足足够的工作量证明

区块的时间戳大于之前 11 个区块的平均时间戳，且最晚不超过当前验证时刻之后的两个小时

区块大小在限制范围内

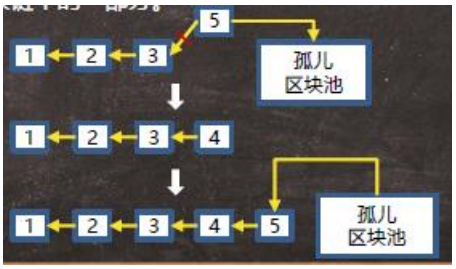
第一个交易（且只有一个）是币基交易

系欸 DNA 使用步骤 1 的交易检测规则来验证所有的交易，来确保它的有效性

更新区块链副本

节点在本地区块链副本中根据区块中的“前序区块哈希值 pre_H”字段找到前序区块

如果在本地区块链副本中未找到新区块的前序区块，那么把这个新区块当作“孤儿”区块，暂时存放在孤儿区块池中，一旦该孤儿区块的前序区块到达本节点并被存储在链条上，那么孤儿区块就从孤儿区块池中被释放出来，并与其前序区块相连，正式成为区块链中的一部分。

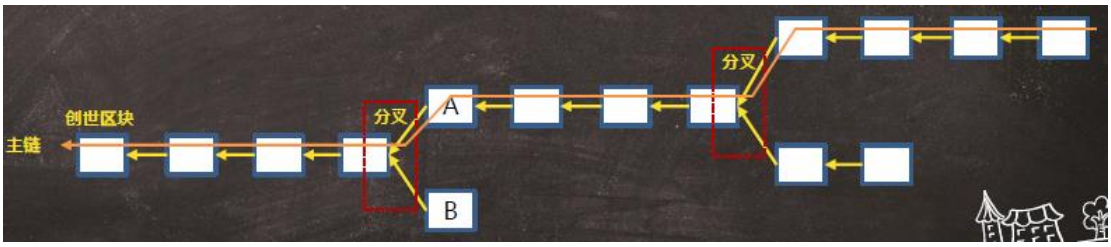


Step 4:每个节点独立选择最长的区块链

区块链副本可能存在若干组分支。

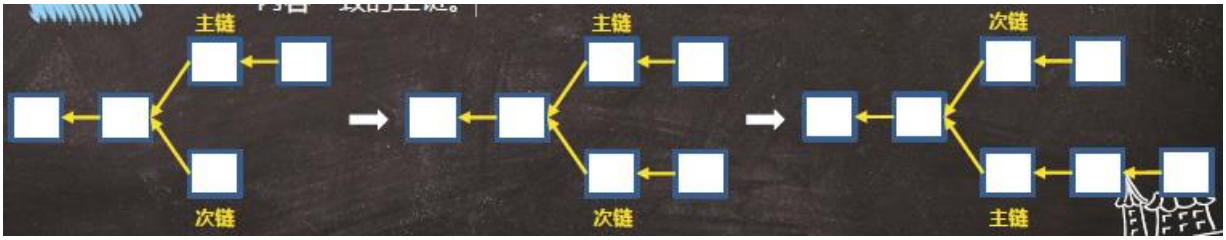
分支有主分支和次分支之分，主分支上包含的区块最多，进而形成了区块链副本的主链，次分支形成了次链。

大多数情况下主链都是拥有最多工作量证明的链条，通常是最长的链条。



比特币系统允许保留分支次链。有些情况下，节点收到的新区块是基于次链的，验证通过后就将其挂接到次链上，这时节点会比较次链和主链的区块数，如果次链的区块数超过了主链，那么该节点将选择次链作为新的主链，而原来的主链变成次链。

通过选择含有最多区块的链条，全网的所有节点最终达成共识，即本地都保存内容一致的主链。



24. 区块链分叉

比特币系统是去中心化的分布式网络，区块到达各个节点的时间会存在差异，同一时刻不同的节点可能拥有不一样的区块链副本。

比特币系统规定每个节点总是选择最长的链条，最终达到全网的统一。

分叉是各个节点的区块链副本暂时不一致的现象。

当更多的区块加入某个分支链时，各个节点会将其视为主链而进行同步更新，最终得到收敛从而解决分叉问题。

25. 共识算法的有效性

资金难窃取：

每一笔交易记录了该笔交易引用的前序交易输出、转账的金额、转账地址和前序交易输出所剩金额

窃取比特币资金——伪造他人签名

数字签名保证，只要攻击者没有获取到目标用户的私钥，攻击者就无法伪造目标用户的签名

双重支付问题（双花问题）：

数字货币可以被复制，同一笔钱能被花两次

节点在接收到广播的交易以后，既要验证签名是否属实，也要验证这笔交易是否涉及双重支付。

当 6 个区块确认以后，发起双重支付的另一条分支链很难赶超主链

26. 比特币的激励机制

比特币的发行

系统给成功挖矿的节点给予比特币奖励，最初的区块奖励金额是 50BTC，每产生 21 万个区块减半一次。

比特币的经济价值

稀缺性（发行总量固定）

持久性（具有较好的价值存储手段）、

可携带性（用户可通过自由携带的钱包里的私钥来管理和控制比特币）

可互换性（比特币不会因其过去的使用权或所有权，而影响其当前场景的使用）

可分割性（比特币的最小单位是 1 聪（Satoshi，以中本聪的名字命名））

易识别性（比特币难以伪造）

比特币的供应：随着矿工不停挖矿而产生

矿工通过币基交易获取奖励（挖矿奖励+区块中包含的所有交易的交易费）

目前矿工的主要收益来源是新挖出区块的区块奖励。随着时间的推移，区块奖励额度下降，交易费将逐渐成为矿工的主要奖励来源。

比特币的 51%攻击：比特币网络中存在一个掌握绝大部分挖矿计算能力的攻击者

无法盗用其他用户的比特币、无法压制某些交易、无法改变区块的奖励、可实现双重支付

27. 简单支付协议(Simplified Payment Verification)

完全节点拥有完整的区块链副本，需要较大的存储空间
SPV 节点只下载区块头部和关心的交易数据，不保存全网所有的历史交易数据
没有完整交易数据的节点无法构建和维护更新全网所有未花费交易输出 UTXO，SPV 提供的方法能让其他节点提供自己关心的交易在区块链上的梅克尔路径

交易验证：
Step 1:定位要验证的交易所在的区块
SPV 节点询问存有完整区块链副本的节点，获取所关心交易在链上对应的梅克尔路径和相应的区块头部，然后根据梅克尔路径计算出梅克尔树根哈希值，从而验证交易隶属于该梅克尔树。接着将交易与区块相连，并利用区块头部信息将区块定位于本地所存的区块链（确切地说是区块头部链），以此验证交易被记录进区块链。
Step 2:确认交易是否在区块链上被 6 次确认
SPV 节点需要其他节点配合验证交易，可能被诱导连接到虚假网络中。

SPV 节点获取区块头部：给对等节点发送 getheaders 请求消息，对等节点收到请求后返回 headers 消息，消息中包含区块头部信息。
SPV 节点获取交易：给对等节点发送 getdata 消息，对等节点反馈包含交易的 tx 消息。
SPV 节点对特定交易的获取请求会暴露其自身的钱包地址，第三方可跟踪钱包发生的所有交易请求，然后根据 SPV 钱包地址等数据进行关联，从而导致用户隐私泄露。

使用"布隆过滤器"（bloom filters）保护隐私：
布隆过滤器允许 SPV 节点通过调节搜索精度来调节私密性。SPV 节点将布隆过滤器提供给对等节点，由对等节点过滤出它感兴趣的交易，不会泄露 SPV 节点的地址。
布隆过滤器构成：包含一个N位的二进制可变长度数组和M个哈希函数。
工作机制：（略）

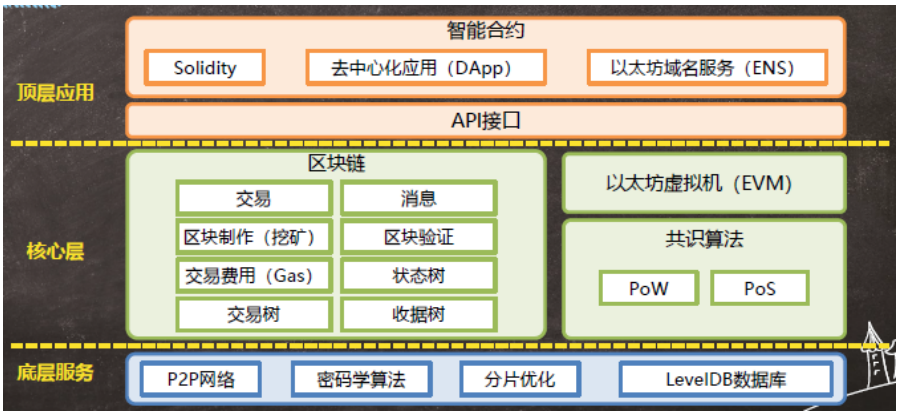
第五章 以太坊

以太坊概述

1. 以太坊与比特币的区别
   以太坊有更快的出块速度以及更先进的奖励机制。目前出块间隔设计为 12 秒。
   以太坊支持智能合约，用户可以自己定义数字资产和流通的逻辑，通过以太坊虚拟机几乎可以执行任何计算。
   以太坊的社区更加活跃，技术生态更加完善。
2. 以太坊核心技术
   智能合约
   以太坊利用图灵完备的虚拟机实现对任意复杂代码逻辑（即智能合约）的解析
   开发者使用 Solidity 或 Serpent 等语法创建可在以太坊虚拟机上运行的应用
   每个以太坊节点都运行虚拟机并执行相同的指令
   Proof of Stake(PoS，股权证明)
   在共识中，验证人（validator）轮流提议新块并对下一个块投票，每个验证人的投票权重取决于其持币量的大小（即股权）。验证人为区块链网络提供出块服务，网络也会给验证人返回奖励
   以太坊实现的 PoS 机制命名为 Casper
   Casper FFG：一种混合 PoW/PoS 的共识机制
   Casper CBC：协议在开始阶段时部分确定的，其余部分协议以证明能够满足所需/必需属性的方式得到
3. 去中心化应用
   运行在去中心网络上的应用软件
   狭义：运行在区块链上的一组智能合约组成了 DApp
   广义：具有开源、去中心化、激励机制和共识机制等特性的应用
   开源：代码的修改需要由大多数用户达成共识决定
   去中心化：DApp 运行的所有操作必须被记录在一条公开的、去中心化的区块链之上
   激励机制：奖励投入算力、内存空间等资源以维持 DApp 运行的用户
   共识协议：网络中各用户节点在运行 DApp 的过程中，通过密码学算法展示某种特定的价值证明，向其他用户节点证明其运行的正确性，从而使网络中的所有节点对某一运行过程达成共识。

以太坊架构和组成

4. 以太坊整体架构
   底层服务：包含 P2P 网络服务、LevelDB 数据库、密码学算法以及分片（Sharding）优化等基础服务。
   核心层：包含区块链、共识算法和以太坊虚拟机等核心元件。
   顶层应用：包括 API 接口、智能合约以及去中心化应用等。



5. 以太坊中的叔块
   临时分叉：两个独立的矿工先后发现了两个不同的满足要求的区块。
   孤块（orphan block）：如果一个块不是最长链的一部分，那么它被称为“孤块”。
   叔块：该区块的父块是当前区块的父块的父块。
   最重的链（heaviest）：以太坊会支付报酬给发现叔块的矿工，还会给打包叔块的后继矿工奖励，引用叔块使主

链更重。

叔块奖励= (叔块高度+ 8 – 引用叔块的区块高度) * 普通区块奖励/ 8

以太坊规定，位于区块的前 K 层（2<= k <= 7）的孤块，才能称为区块的叔块。

6. 以太坊中的四棵树

   状态树、交易树、收据树和存储树，均采用 MPT 数据结构

   可以帮助以太坊客户端完成一些简易的查询，如查询某个账户的余额、某笔交易是否被包含在区块中等

   区块、交易等数据存储在 LevelDB 数据库中。LevelDB 数据库是一个键值对（key-value）数据库，key 一般与哈希值相关，value 则是存储内容的 RLP（Recursive Length Prefix）编码

7. 数据编码与压缩

   RLP（Recursive Length Prefix）是一种编码算法，用于编码任意的具有嵌套结构的二进制数据

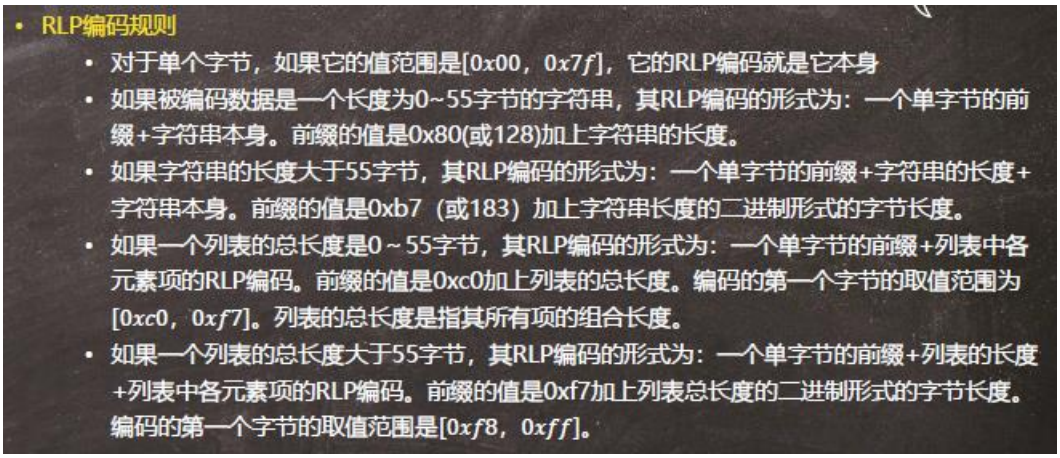
   以太坊中的区块、交易等数据结构会先经过 RLP 编码处理，然后再存储到数据库中

   RLP 编码只处理两种类型的数据：字符串和列表

   字符串一般指的是一串二进制数据

   列表是一个嵌套递归结构，列表元素可以是字符串或列表

   [“red”，[“white”，“red”]，“yellow”，“green”]



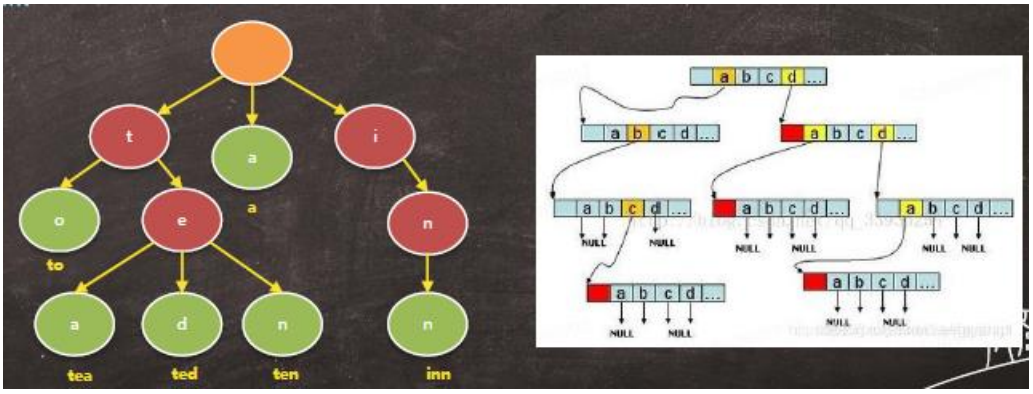
8. 字典树(Trie)

   又称为字典树或者前缀树（prefix tree），主要用于统计和排序大量的字符串

   根节点不包含字符，除根节点外每一个节点都只包含一个字符

   从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串

   每个节点的所有子节点包含的字符都不相同



缺点：高度不可控、安全系数不高

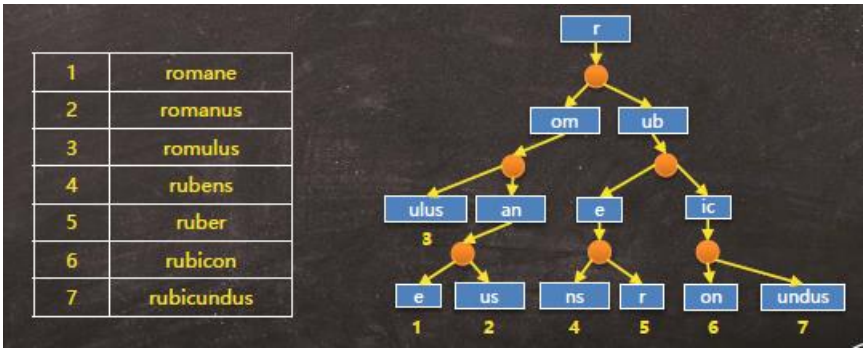
9. 基数树（Radix Tree）

   又称压缩前缀树（compact prefix tree），是一种空间优化后的字典树

   如果一个节点只有唯一的子节点，那么这个子节点就会与父节点合并存储

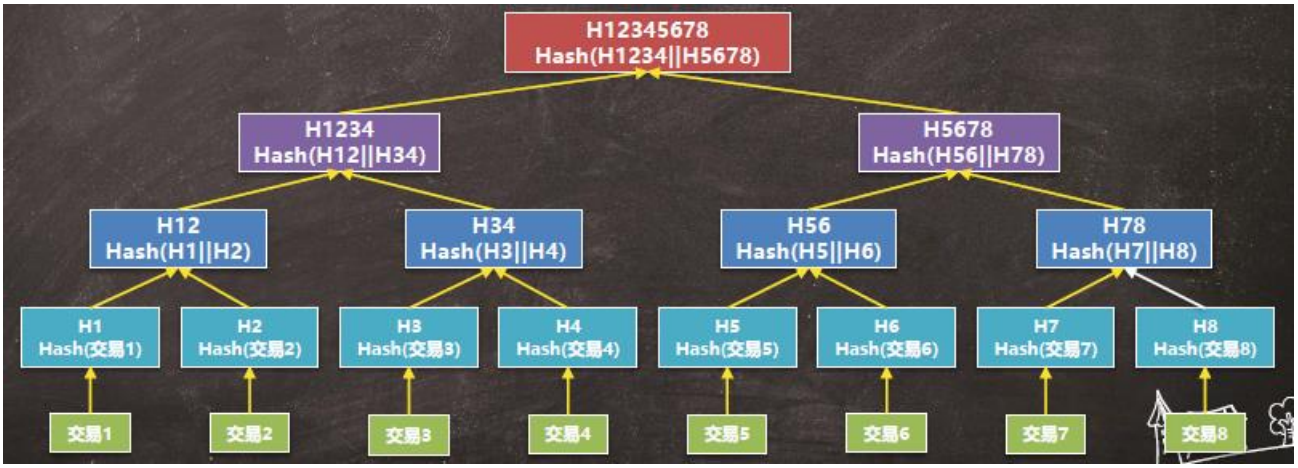
10. Patricia 树

    基数树的一种。如果一个基数树的“基数”（radix）为 2 或 2 的整数次幂，则被称为“Patricia 树”



11. Merkle 树

    用每个节点的 hash 值来建立对应的关系。可以实现数据校验，防止篡改



12. Merkle Patricia Tree（MPT）

    以太坊采用账户模型：账户存在多个属性（余额、代码、存储信息），属性（状态）需要经常更新。

    以太坊对数据结构的要求

    要求 1：在执行插入、修改或者删除操作后能快速计算新的树根，而无需重新计算整个树，并且可以快速检验节点的正确性。

    要求 2：即使攻击者故意构造非常深的树，它的深度也是有限的。否则，攻击者可以通过构建足够深的树使得每次树更新变得极慢，从而执行拒绝服务攻击。

    要求 3：对节点的访问效率要求较高。

    MPT 的作用

    存储任意长度的键值对（key-value）数据；

    提供了一种快速计算所维护数据集哈希标识的机制；

    提供了快速状态回滚的机制；

    提供了一种称为 Merkle 证明的证明方法，进行轻节点的扩展，实现简单支付验证；

    MPT 节点分类

    空节点（NULL）

    叶子节点（leaf）：没有子节点。表示为[encodedPath, value]

    分支节点（branch）：可以有多个子节点。表示为[v0 … v15, vt]

    扩展节点（extension）：只能有一个子节点。表示为[encodedPath，其它节点的 hash]

    Key 定义

    Origin Key：数据的原始 key，为字节数组。

Secure Key：为原始 key 计算哈希$Keccak256(Origin\ Key)$的结果，长度固定为 32 字节，用于防止深度攻击。

Hex Key：将 Origin Key 或 Secure Key 进行半字节（nibble）拆解后的 key，为 MPT 树真正存储的 key。每个 nibble 的长度为 4 bit，可以表示数字 0~15。在以上条件的限制下，MPT 树 key 的长度固定为 64 字符。

HP Key：hex prefix encoding，Hex 前缀编码。当使用 nibble 寻找路径时，为区分奇偶长度，叶子节点和拓展节点的 encodedPath 使用一个前缀作为标签，这个标签也用于区分节点类型。

节点哈希：当子节点内容的 RLP 编码结果小于 32 字节时，则父节点直接存储子节点内容，否则，父节点存储编码结果的哈希值。——父节点哈希值的计算依赖于子节点

13. 交易树：

每个区块都有一棵独立的交易树，对应区块头里的交易根。交易树的 key（路径）为 rlp(transactionIndex)，value 为交易序列化后的值。其中 transactionIndex 表示交易在该区块中的索引。

收据树：

每个交易对应一个交易收据，交易收据记录了交易执行结果，包括执行状态、Gas 消耗、事件日志等。每个区块也有自己的收据树，对应区块头里的收据根。与交易树类似，key 为 rlp(transactionIndex)，value 为交易收据序列化后的值。

状态树：

状态树是全局的、不断更新的。它的 key 为$keccak256(ethereumAddress)$，value 为$rlp(ethereumAccount)$。其中$ethereumAddress$表示以太坊账户地址，$ethereumAccount$表示以太坊账户，包含四个字段$[nonce, balance, storageRoot, codeHash]$。

存储树：

存储树存储了合约的所有状态数据，每个合约有单独的存储树。它的 key 为$keccak256(position)$，value 为 position 对应值（32 字节）的 rlp 编码。其中 position 为状态变量在合约中存储槽的位置，用 32 字节表示。

14. LevelDB 数据库

一种高效的键值对（key-value）数据库，键值均为二进制

以太坊中的 LevelDB 数据库：

BlockDB：保存块的主体内容，包括块头和交易

StateDB：保存账户的状态数据

ExtrasDB：保存收据信息和其它辅助信息

用户接口：$put\ k, v$，$get\ k, v$，$delete\ k, v$

键（key）和值（value）都是任意长度的字节数组，一条记录（即一个键值对）默认按照 key 的字典顺序存储

支持遍历，包括前向和反向

支持原子写操作（ atomic write ）

支持过滤策略（bloomfilter）

支持数据自动压缩

底层提供了抽象接口，允许用户定制

15. 以太坊账户

外部账户

存储以太币

由私钥控制，是用户实际控制的账户

每个外部账户有一对公私钥，公钥由私钥生成，地址由公钥生成（采用 SHA3）

不能包含以太坊虚拟机（EVM）代码

合约账户

包含合约代码的账户，相关联的代码通过交易或者其它合约发送的调用来激活

由合约代码控制

合约账户地址由合约创建时合约创建者的地址，以及该地址发出的交易共同计算得出

合约被执行时，只能操作合约账户拥有的特定存储

以太交易和共识

16. 以太币

以太币来源：矿前+区块奖励+叔区块奖励+叔区块引用奖励

矿工挖出新区块获得的奖励：静态奖励+动态奖励

静态奖励：每挖出一个新块，可获得 5 个以太币作为奖励。

动态奖励：矿工挖出的区块中包含的所有交易费用归矿工所有。如果该区块包含叔区块，矿工还可以从每个叔区块引用中获得约 0.15 个以太币的奖励。每个区块最多引用 2 个叔区块，被引用过的叔区块不能重复利用。

货币单位：最小单位为 wei

17. 交易费用

以太坊上的每一笔交易都需要支付一定的费用，用于支付交易执行所需要的计算开销

Gas: 用来衡量一笔交易所消耗的计算资源的基本单位。以太坊节点执行一笔交易所需的计算步骤越多、越复杂，该交易消耗的 Gas 就越多。

Gas Price：一单位 Gas 所需的手续费（以太币）

例：一个转账交易消耗 21000Gas，假设 Gas Price 为 1Gwei/Gas，那么这笔交易的手续费为 0.000021Ether。

Gas Limit

单个交易：交易发送者愿意为这笔交易执行所支付的最大 Gas 数量，需要发送者在发送交易时设置，可以保护用户免受错误代码影响以致消耗过多的交易费

单个区块：区块所允许包含的最大 Gas 总量

用户愿意为一笔交易支付的最高金额$Gas\ Limit \times Gas\ Price$

区块的 Gas Limit 设置得越大，矿工可以获取越多的交易费，但是需要更多的带宽

18. 交易内容

from：交易发送者的地址，必填

to：交易接收者的地址，如果为空则意味这是一个创建智能合约的交易

value：发送者要转移给接收者的以太币数量

data：数据字段，如果存在则是表明该交易是一个创建或者调用智能合约交易

Gas Limit：表示这个交易允许消耗的最大 Gas 数量

Gas Price：表示发送者愿意支付给矿工的 Gas 价格

nonce：用来区别同一用户发出的不同交易的标记

hash：由以上信息生成的哈希值，作为交易的 ID

r、s、v：交易签名的三个部分，由发送者的私钥对交易 hash 进行签名生成

19. 交易类型

转帐交易：从一个账户向另一个账户发送以太币，需指定交易的发送者、接收者、转移的以太币数量

```
web3.eth.sendTransaction({
    from: " 0xb60e8dd61c5d32be8058bb8eb970870f07233155" ,
    to: " 0xd46e8dd67c5d32be8058bb8eb970870f07244567 " ,
    value: 10000000000000000
});
```

创建智能合约的交易：通过发送交易将智能合约部署到区块链上。“to”字段为空字符串；在“data”字段中指定初始化合约的二进制代码，之后合约被调用时，该代码的执行结果将作为合约代码。

```
web3.eth.sendTransaction({
    from: " 0xb60e8dd61c5d32be8058bb8eb970870f07233155 " ,
    data: " contract binary code "
});
```

执行智能合约的交易：通过交易执行已经部署在区块链上的智能合约。通过“to”字段指定要调用的智能合约的地址；通过“data”字段指定要调用的方法以及向该方法传递的参数。

```
web3.eth.sendTransaction({
    from: " 0xb60e8dd61c5d32be8058bb8eb970870f07233155 " ,
    to: " 0xb4259e5d9bc67a0f2ce3ed372ffc51be46c33c4d " ,
    data: " hash of the invoked method signature and encoded parameters "
});
```

交易查询

```
web3.eth.getTransaction('0xc5eee3ae9cf10fbee05325e3a25c3b19489783612e36cb55b054c2cb4f82fc28')
{

    blockHash: '0xdb85c62ef50103f08e9220b59d6c08cbfb52e61d84926dedb3fe9b6940e6bbea',
    blockNumber: 290081,
    from: '0x1dcb8d1f0fcc8cbc8c2d76528e877f915e299fbe',
    Gas: 90000,
    GasPrice: 50000000000',
    hash: '0xc5eee3ae9cf10fbee05325e3a25c3b19489783612e36cb55b054c2cb4f82fc28',
    input: '0x',
    nonce: 34344,
    to: '0x702bd0d370bbf0b97b66fe95578c62697c583393',
    transactionIndex: 0,
    value: 5000111390000000000'

}
```

20. 转账或者合约调用交易的处理过程

Step 1: 发送者发起转帐交易或合约调用交易请求

Step 2: 对等节点检验、存储和转发交易

检查交易，计算交易费用，从发送方账户中减去相应费用，并计算可能的最大交易费用，在本地区块链上从发送方账户中减去相应费用。将交易请求存放在存储池中，并转发给其他节点。

Step 3: 有“挖矿”功能的节点打包交易并执行合约代码，竞争记账权

Step 4: 获得记账权的节点将新区块发送给其它节点

Step 5: 维护区块链状态的节点验证区块并保存在本地区块链副本中

21. 创建智能合约的交易处理过程

Step1：发送者发起创建智能合约的交易请求

Step2：对等节点检验、存储和转发交易，有“挖矿”功能的节点打包交易并部署合约

Step3：获得记账权的节点发送新区块至以太坊

Step4：维护区块链状态的节点验证区块，将智能合约部署在本地区块链上

22. PoW 共识——Ethash 算法

算法特点：挖矿效率基本与 CPU 无关，与内存大小、带宽正相关

基本思想：一小一大两个数据集

小：初始大小为 16M 的缓存（cache）。容量大小每 3 万个区块更新一次。

大：初始大小为 1G 的数据集 DAG。矿工为了能更快的挖矿需要保存大的数据集，以免重复计算耽误时间。大的数据集中的元素都是通过小的 cache 计算得到的

挖矿过程：寻找一个 nonce 值，使哈希值H区块头部≤目标值

验证过程：

全节点：内存中保存了大的 DAG，只需循环计算 64 次后将最重得到的哈希值与目标值比较即可

轻节点：首先通过小的 cache 计算出大的 DAG，之后的计算过程与全节点一样

挖矿难度：

在区块头部中包含$difficulty$字段，定义了当前块的“出块难度”。验证时，将算得到的哈希值作为整数与$2^{256}/difficulty$比较，小于该值才是有效的哈希。

23. PoS(Proof of Stake)共识

基本思想：基于网络参与者目前所持有的数字货币的数量和时间进行利益分配

PoS 算法描述：以太坊区块链网络由一组验证者决定，任何持有以太币的用户都能发起一笔特殊形式的交易，将他们的以太币锁定在一个存储中，从而使自己成为验证者，然后通过一个当前验证者都能参与的共识算法，完成新区块的产生和验证。

两种类型：基于链的 PoS 和拜占庭容错的 PoS

基于链的 PoS 算法：在每个时隙内伪随机地从验证者集合中选择一个验证者，给予验证者创建新区块的权利，但是验证者要确保该块指向最多的块(指向的上一个块通常是最长链的最后一个块)。

拜占庭容错的 PoS 算法：验证者有权提出块并且给被提出的块投票。在每一轮中，每一个验证者都为某一特定的块进行“投票”，最后所有在线和诚实的验证者都将“商量”被给定的块是否可以添加到区块链中，并且意见

不能改变。

$hash(block\_header) \leqslant target \times coinage$（$coinage =$ 币的个数×币的剩余使用时间）

优点：

不需要为了保证区块链的安全而消耗大量的电力资源。

矿工"从消耗大量资源的挖矿行为中解放出来，将算力资源转向区块链技术的开发应用上，促进了区块链技术的发展。

随着规模经济（指扩大生产规模引起经济效益增加的现象）的消失，中心化所带来的风险减小。

实施的奖励惩罚措施使得各种恶意攻击变得极其昂贵，安全性比 PoW 更高。

缺点：

当出现多条相互竞争的准主链时，大多数的验证者会尝试在所有的准主链上出块，这可能导致永远无法达成共识。——无成本利益关系（Nothing at Stake）问题

惩罚机制 1：如果验证者在不同的链上创建块，则在某个事后的时间点将能证明他们错误行为的记录包含在区块链中，并对他们做出扣除押金的惩罚。

惩罚机制 2：惩罚验证者在错链上出块的行为。

Casper 协议：

验证者抵押一定比例的以太币作为保证金

验证者对新块进行投票以决定它是否有效——下注

根据投票结果形成大多数人意见（即多数人认为新块有效则新块就是有效的），之前投票新块有效的用户将会收回保证金并获得奖励

验证者如果作恶，保证金将被没收

智能合约

24. 基本概念

存储在区块链网络上的一段代码，定义了所有使用合约的各方同意的有关合同条款的全部信息。输入满足要求的条件后，智能合约自动执行所有相应的预设代码，输出相应的期望结果。智能合约的行为由合约代码控制，智能合约的账户存储保存了合约的状态

以太坊的 P2P 网络中，每个全节点包含一个以太坊虚拟机（EVM），可执行合约代码。

25. 优势

智能合约条款由代码确定，不容易产生歧义

智能合约存储和部署在区块链网络中，网络中各节点独立维护账本副本，合约内容很难被篡改。此外，区块链账本中保存了合约的执行记录，可以作为永久的交易凭证。

智能合约的创建和执行都依赖于区块链协议，合约执行的强制力可以保证。

26. 智能合约的操作

创建智能合约：

编写智能合约

编译成字节码

部署到区块链

调用智能合约：发起一笔指向智能合约地址的交易

27. 存储方式：

栈（stack）：

以太坊虚拟机上的所有运算都运行在栈上

栈中每一个元素的长度是 256 位

账户存储（storage）：

作为账户的一个属性保存在区块链上，不会随着合约执行结束而被释放

一个稀疏的散列表，键和值的长度都是 256 位

修改存储内容需要消耗较多的 Gas

内存：

以太坊虚拟机在运行代码时临时分配的一块空间，会随着合约调用的结束自动释放

以字节作为基本存储单位

当现有的区域用完时，内存空间会以 32 字节为单位进行拓展，调用者需要为这部分空间支付 Gas

28. 指令集和消息调用

指令集

基础指令：常用的算术运算、位运算、逻辑运算和比较运算等

区块链特有的指令：用于合约访问区块号和区块时间戳的指令等

以太坊虚拟机的基础数据单元是 256 位，所有指令都以此为单位来传递数据

智能合约的编译就是将高级语言编写的合约代码转换为指令集表示的字节码

消息调用

智能合约之间的调用。调用消息的结构与交易类似，但消息调用属于交易执行的一部分，不会在区块链中产生一条新的交易记录

当发起一个消息调用时，智能合约可以决定为这次调用分配多少 Gas

代理调用（delegate call）：只从目标合约获取代码并执行，不改变当前的上下文环境

29. 日志

开发者可以在合约代码运行过程中记录各种事件产生的日志

区块链不保存完整的日志文件，而只在交易收据中保存日志的哈希值

以太坊全节点在同步区块数据时会执行交易，同时记录下产生的日志

以太坊轻节点使用布隆过滤器搜索日志

30. Solidity 语言

用于编写智能合约的高级语言，语法类似 Javascript。使用 Solidity 语言编写智能合约避免了直接编写底层的以太坊虚拟机代码，提高了编码效率，也具有更好的可读性

结构：

状态变量：永久存储在合约账户存储中的值，用于保存合约的状态

函数：合约代码的执行单位，一个合约中可包含多个函数

函数修改器：用于改变函数的行为，在函数执行前或执行后插入其他逻辑

事件：以太坊日志协议的高层次抽象，用于记录合约执行过程中发生的各种事件和状态变化

变量类型：

值类型：在每次赋值或者作为参数传递时都会创建一份拷贝

引用类型：

状态变量与部分类型的局部变量默认保存在账户存储中

函数的参数和其它简单类型的局部变量保存在内存中

值类型：

布尔类型（bool）：可能的取值为常量 true 和 false，支持!（逻辑非）、&&（逻辑与）、||（逻辑或）、==（等于）、!=（不等于）等运算符

整数类型：有符号整数int，无符号整数$uint$。支持通过后缀指明变量使用多少位进行存储，后缀必须是 8 ~ 256 范围内 8 的整数倍，例如$int8$、$int256$。如果没有显示指明后缀，int默认表示$int256$，$uint$默认表示$uint256$

枚举类型（enums）：用户自定义的类型，用于声明一些命名的常数。可与整数类型之间显示地进行类型转换，不能自动进行隐式转换。枚举类型的成员默认从 0 开始，依次递增

地址类型：

地址类型（address）：长度为 20 字节，拥有一些成员方法和变量。

引用类型：数组

固定长度的静态数组、运行时可动态改变长度的动态数组

storage 类型（账户存储中的数组）：数组元素可以是任意类型

memory 类型（内存中的数组）：数组元素不能是映射

成员变量和函数：

length：获取数组长度

push：账户存储中的动态数组以及 bytes 类型的变量可以通过调用 push 方法在数组尾部添加元素，返回值为数组新的长度

bytes 和 string：bytes 通常用于表示任意长度的字节数据，而 string 用于表示任意长度的字符数据(UTF-8 编码)；string 不支持 length 成员和下标访问；两者可以互相转换

引用类型：结构体（struct）

开发者根据需要自定义变量类型

结构体可以作为映射或者数组中的元素，其本身也可以包含映射和数组等类型

结合体本身的成员不能是结构体，嵌套自身会导致无限循环

引用类型：映射（mapping）

一种键值对映射关系的存储结构：$mapping(KeyType => ValueType)$

$KeyType$：除了映射、动态数组、合约、枚举类型、结构体以外的任何类型

$ValueType$：可以是任意类型，包括映射本身

没有长度概念

映射并不存储键的数据，只存储它的$Keccak$-256 散列值

类型转换

隐式转换：转换时必须符合一定条件，不能导致信息丢失。例如，uint8 可以转换为 uint16，但是 int8 不可以转换为 uint256，因为 int8 可以包含 uint256 中不允许的负值。

显示转换：可以使用构造函数语法，显式地将数据类型转换为另一种类型。

运算符

优先级	描述	运算符
1	后缀增量和减量操作符	`++`, `--`
	新创建表达式	`new <typename>`
	数组下标	`<array>[<index>]`
	成员访问	`<object>.<member>`
	函数调用	`<func>(<args...>)`
	小括号	`(<statement>)`
2	前缀递增和递减操作符	`++`, `--`
	一元负运算符(Unary minus)	`-`
	一元运算(nary operations)	`delete`
	逻辑非	`!`
	位求反	`~`
3	取幂/乘方	`**`
4	乘法，除法和取余	`*`, `/`, `%`
5	加减法	`+`, `-`
6	位移运算符	`<<`, `>>`
7	按位与	`&`
8	按位异或	`^`
9	按位或	`|`
10	不等式运算	`<`, `>`, `<=`, `>=`
11	等式操作符	`==`, `!=`
12	逻辑与	`&&`
13	逻辑或	`||`
14	三元运算符	`<conditional> ? <if-true> : <if-false>`
	赋值运算符	`=`, `|=`, `^=`, `&=`, `<<=`, `>>=`, `+=`, `-=`, `*=`, `/=`, `%=`
15	逗号运算符	`,`

delete：赋值运算

内置单位、全局变量和函数

货币单位：一个字面量的数字可以使用wei、$finney$、$szabo$和$ether$等后缀表示不同的额度，不加任何后缀则默认单位为wei。例如，“$2ether == 2000finney$”的结果是$true$。

时间单位：不同的时间单位可以以秒为基本单位进行转换

区块和交易属性：有一些方法和变量可以用于获取区块和交易的属性。

block.blockhash(uint blockNumber) returns(bytes32)	获取特定区块的哈希值，只对不包括当前区块的256个最近的区块有效
block.coinbase	类型为address，表示当前区块“矿工”的账号地址
block.difficulty	类型为uint，表示当前区块的挖矿难度
block.gaslimit	类型为uint，表示当前区块的Gas限制
block.number	类型为uint，表示当前区块编号
block.timestamp	类型为uint，以UNIX时间戳的形式表示当前区块的产生时间
msg.data	类型为bytes，表示完整的调用数据

异常处理

assert(bool condition)	当条件不为真时抛出异常，用于处理内部的错误
require(bool condition)	当条件不为真时抛出异常，用于处理输入或者来自外部模块的错误
revert()	中断程序执行并且回退状态改变

数学和加密函数

addmod(uint x，uint y，uint k) returns(uint)	计算$(x+y)\%k$，加法支持任意精度，结果即使超过2^{256}也不会被截取
mulmod(uint x，uint y，uint k) returns(uint)	计算$(x*y)\%k$，乘法支持任意精度，结果即使超过2^{256}也不会被截取
keccak256(...) returns(bytes32)	计算 $Ethereum-SHA-3(Keccak-256)$ 哈希值
sha3(...) returns(bytes32)	$keccak256()$方法的别名
sha256(...) returns(bytes32)	计算 $SHA-256$ 哈希值

控制结构语句

选择结构：当if 后的条件语句结果为$true$，则会执行相应的代码，否则将继续$else$ 后面的条件判断。其中$else\ if$ 与$else$ 都是可选的。

循环结构：while 循环、for 循环

函数

一个函数可以有多个参数，也可以有多个返回值，如果没有对返回值进行赋值，默认值为 0

内部调用：调用同一合约中的函数。对应 EVM 指令集中的 JUMP 指令，内存不会被回收

外部调用：调用其他合约实例的方法。会创建一个消息发送给被调用的合约。

函数可见性修饰词

external：用于修饰函数，表示函数为一个外部函数，外部函数是合约接口的一部分，这意味着只能通过其他合约发送交易的方式调用外部函数。

public：用来修饰公开的函数/变量，表明该函数/变量既可以在合约外部访问，也可以在合约内部访问。

internal：内部函数/变量，表示只能在当前合约或者继承自当前合约的其他合约中访问。

private：私有函数和变量，只有当前合约内部才可以访问。

constant 函数

在声明一个函数时，可以使用$constant$或者$view$关键字告诉编译器这个函数进行的是只读操作，不会造成其他状态变化。

造成状态变化的语句包括：修改变量的值、触发事件、创建其他合约、调用任何非$constant$函数等。

对于外部可见的状态变量，会自动生成一个对应的$constant$函数，称为访问函数，对于数组和映射这类变量，其访问函数接受表示下标值的参数。

fallback 函数

合约中默认隐式存在的函数，不能接受任何参数并且不能拥有返回值。

当一个合约收到无法匹配任何函数名的函数调用或者仅仅用于转账的交易时，fallback 函数将会被自动执行，默认的行为是抛出异常。

可以使用$function(){\cdots}$ 这样的方式重写$fallback$函数

函数修改器（Function Modifiers）

可以在一定程度上改变函数的行为，比如可以在函数执行前自动检查参数是否合法。

函数修改器可以被继承，也可以被派生类覆盖重写。

异常处理

以太坊使用状态回退机制处理异常

Solidity 语言提供了两个函数$assert$和$require$来检查条件，当条件不满足的时候抛出一个异常

$assert$函数：用于检查变量和内部错误

$require$函数：用于确保程序执行的必要条件是成立的

事件和日志

以太坊客户端可以使用 JavaScript 的回调函数监听事件循环结构

当事件触发时，会将事件及其参数存储到以太坊的日志中，并与合约账户绑定。

日志无法在合约中访问

可以给事件建立索引以方便查找日志

除使用 event 外，还可以使用一些底层接口来记录日志，这些接口可以用$log0$、$log1$、$log2$、$log3$ 和$log4$这些函数来访问。$logi$可以接受$i+1$ 个$bytes32$ 类型的参数，其中第一个参数用作日志的数据部分，其他参数作为$topic$保存下来。

智能合约的继承

Solidity 支持继承与多重继承

当一个通过继承产生的合约被部署到区块链上时，区块链上只创建了一个合约，所有基类合约的代码都会在子类合约中有一份拷贝

派生合约的构造函数需要提供基类合约构造函数的所有参数，实现的方法有两种：

直接在继承列表中指定

在定义派生类构造函数时提供

允许使用抽象合约（一个合约只有函数声明而没有函数的具体实现）

抽象合约可以作为基本合约被继承

第六章 超级账本

架构

1. Hyperledger Fabric
   模块化架构的分布式账本平台，支持不同组件的可插拔实现
   提供高度的机密性、灵活性和可扩展性，支持高度复杂应用
2. Fabric 的逻辑架构
   应用程序角度：
      身份管理：成员必须经过许可才能加入网络
      账本管理：授权用户可通过多种方式查询账本
      交易管理：账本数据只能通过交易执行更新
      智能合约：通过链码执行交易，实现业务逻辑
   底层角度：
      成员服务（包括会员注册，身份保护、内容保密、交易审计等功能）
            为整个区块链提供身份管理、隐私、保密和可审计的服务
            基本功能：注册、登记、申请证书等
            公钥基础设施（PKI）+去中心化共识机制
      共识服务（负责节点间共识管理、账本的分布式计算、账本的存储及节点间 P2P 协议功能的实现）
            为区块链的主体功能提供底层支撑
            基本功能：分布式账本的计算和储存、节点间共识服务、背书验证管理、节点间网络传输协议
            可根据具体需求设置共识协议
      链码服务（提供了实现智能合约所需的接口，为智能合约的安装、运行、部署提供了环境）
            链码：基于标准的一段代码，实现具体业务逻辑
            链码与底层账本解耦：链码更新不需要迁移账本数据
            可使用 Go、Java、Node.js 等语言开发
      安全和密码服务（实现密钥生成、哈希运算、签名验签、加密解密等基础功能）
            证书、加密签名
            多通道隔离
   逻辑架构的设计特点
      完备的权限控制和安全保障
         成员必须获得许可才能加入网络
         通过证书、加密、签名等手段保证安全
         通过多通道功能保证只有参与交易的节点能访问账本数据
      模块化设计，可插拔架构
         状态数据库可采用 Level DB、Couch DB 或其他的 key-value 数据库
         身份管理（identity management）可采用自定义模块
         共识机制和加密算法均可根据实际情况选择替换
      高性能，可扩展，较低的信任要求
         交易处理划分为背书、排序、验证提交三个阶段，不同阶段由不同的节点参与，不需要全网节点参与
         对等（peer）节点和排序节点（orderer）可以独立扩展、动态增加
         只有背书节点（endorsers）和记账节点（committers）能看见交易内容，只需要较低的信任要求即可保证安全

节点

3. Fabric 的节点构成
   客户端节点
      必须连接到某个 peer 节点或排序节点与区块链网络通信
      向背书节点提交交易提案（proposal）

向排序节点广播交易

CA 节点

提供基于数字证书的身份信息

可颁发或撤销成员的身份证书

排序服务节点

对未打包的交易进行排序：交易的先后时序需达成共识

确定交易顺序之后广播给 peer 节点

支持多通道

Peer 节点

区块链去中心化网络的对等节点。账本和链码的载体

功能划分

记账节点（committer）：检验交易的合法性；更新和维护账本

背书节点（endorser）：对交易提案进行校验、模拟执行和背书，与链码绑定

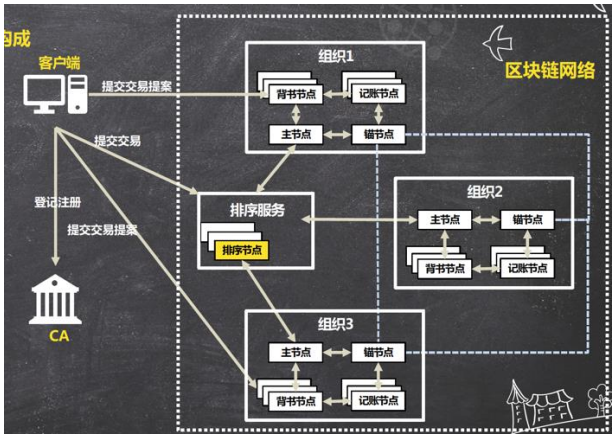
主节点（leader）：与排序节点通信

锚节点（anchor）：在一个通道上可被所有其他 peer 节点发现

每一个 peer 节点持有一个或多个账本，以及一个或多个链码

应用程序通过 peer 访问账本和链码

刚创建的 peer 中既无账本也无“链码”，待后续安装



4. Fabric 中的通道

通道（Channel）：对信息进行隔离的逻辑结构。一个节点可加入多个通道，不同通道的账本是隔离的

两种类型：

系统通道（System Channel）：排序节点通过系统通道管理应用通道

应用通道（Application Channel）：用户的交易信息通过应用通道传递

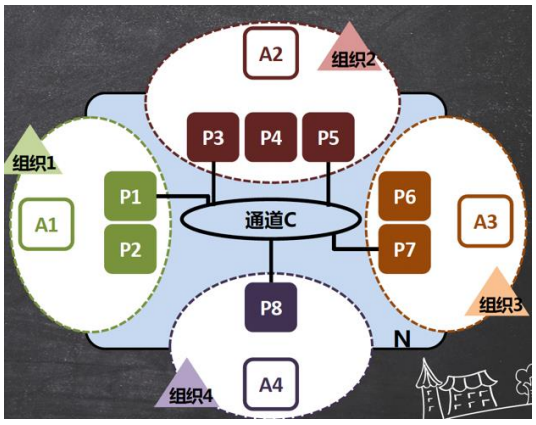
5. Fabric 中的组织

组织是区块链网络所需资源的拥有者和提供者。每个组织有自己的应用程序。

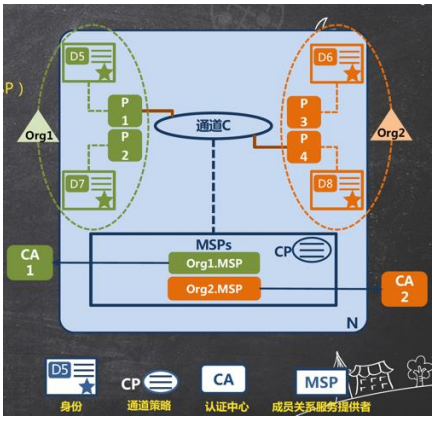
锚节点

每个组织至少一个

与其它组织交换共享账本的数据

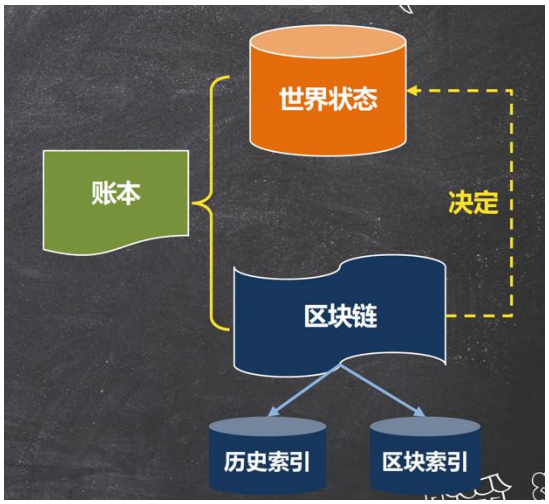


6. Fabric 里面的应用程序
   应用程序通过 peer 访问账本和链码
   两类交互
      账本查询：3 次会话
      账本更新：5 次会话
7. Fabric 中的节点身份
   数字签名表明了 peer 所在的组织
   Membership Service Provider（MSP）
      决定 peer 的访问权限
      一个 peer 只能关联一个 MSP
   所有节点都需身份认证



账本

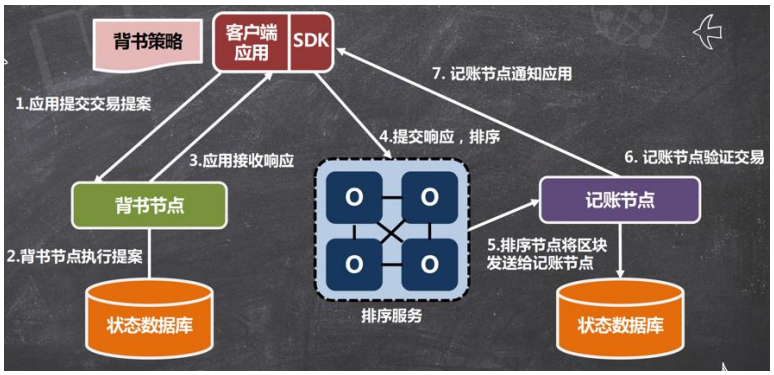
8. Fabric 的账本构成
   数据以分布式账本的形式存储
   每个通道有唯一的账本
   账本构成：区块链（blockchain）
      记录全部交易日志，一旦写入无法修改
      交易日志：每个交易代表一个查询或更新世界状态的操作
      区块头部：区块所有交易的哈希，上一个区块头部的哈希
      以文件形式保存：主要操作为数据追加
   区块的存储
      文件块形式：blockfile_######
      文件块大小：64M （v1.0）
   区块的读取
      区块文件流（blockfileStream）：用于读取文件块
      区块流（blockStream）：在一个文件块中读取区块
      区块迭代器（blocksItr）：在整个区块链上读取区块
   区块索引
      快速定位区块，将查询条件与区块位置建立映射关系
      区块高度、区块哈希和交易哈希→ 区块文件编号+偏移量+区块数据长度
   历史状态索引



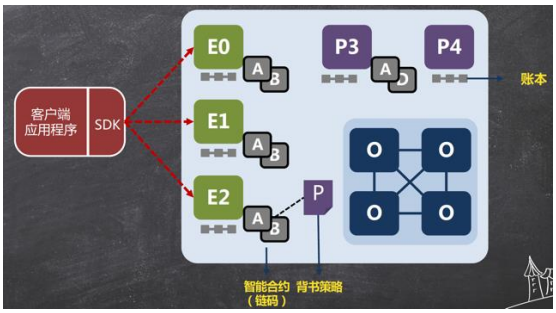
   账本构成：世界状态（world state）
      保存账本当前的状态值，状态值通常以 Key-Value 对表示

交易

9. Fabric 中的交易流程



Step 1:应用程序提交交易提案



背书策略（endorsement policy）

对交易进行背书的条件

构成：主体（principal）+门槛（threshold gate）

主体：期望的签名来源实体，MSP+Role

门槛：整数 t（阈值）和 n 个主体，表示从这 n 个主体中获取 t 个签名

基础表达式形式 EXPR(E[, E...])：EXPR 可以是 AND 或者 OR 逻辑符

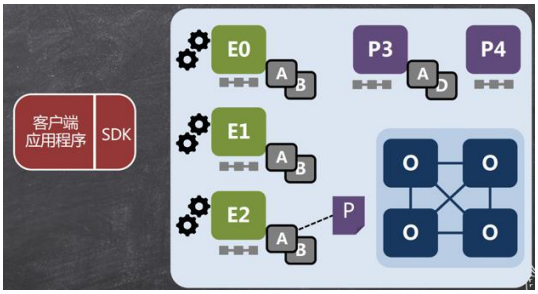
需要三个实体都提供签名　　AND('Org1.member', 'Org2.member', 'Org3.member')

需要两个实体任一提供签名　OR('Org1.member', 'Org2.member')

需要 Org1 的 admin 提供签名，或者 Org2、Org3 的 member 同时提供签名

OR('Org1.admin', AND('Org2.member', 'Org3.member'))

Step 2:背书节点模拟提交交易提案



验证交易提案请求

交易提案格式是否正确

交易在之前并未提交过

提交提案的客户端签名是否有效

提交提案的请求者是否在该通道中有相应的执行权限

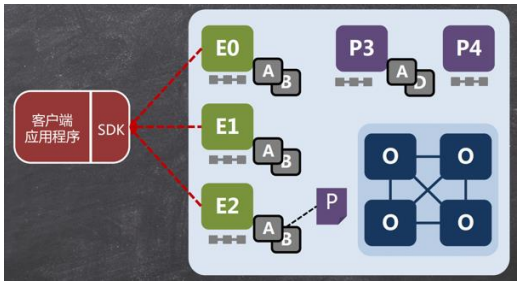
模拟执行

响应值，读写集（读集，写集，版本号）

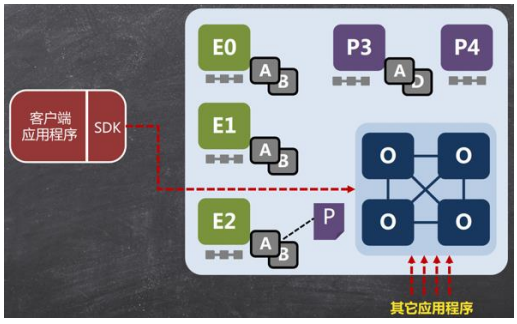
背书签名

返回给客户端

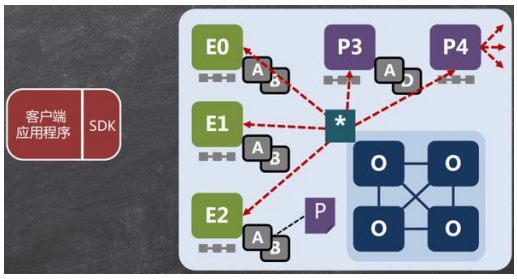
Step 3:应用程序接收响应



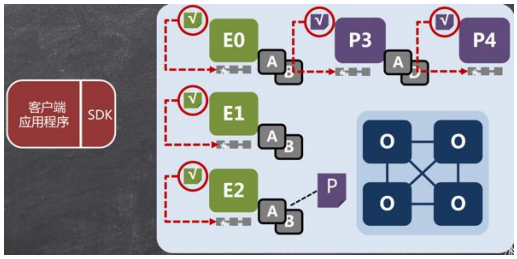
Step 4:排序服务节点接收交易，进行排序



Step 5:排序节点发送给记账节点



Step 6:记账节点验证交易



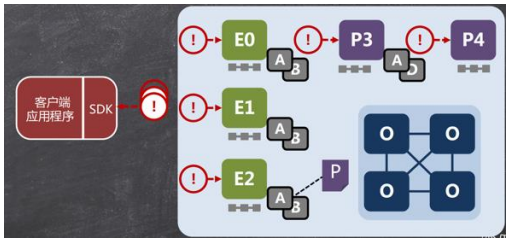
交易数据的完整性

是否重复交易

背书签名是否符合背书策略的要求

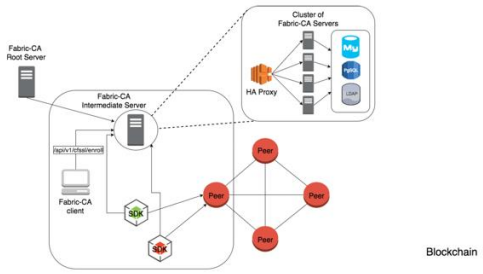
交易读写集是否通过多版本并发控制（Multiversion Concurrency Control ， MVCC）的校验：当前交易的读集状态必须与世界状态以及当前未被持久化的前序交易执行后的状态要保持一致。如果不一致，这笔交易就是无效的

Step 7:记账节点通知应用程序



身份

10. Fabric 中的身份管理
    Fabric 系统的参与者：节点，客户端应用程序，管理员等
    身份决定了参与者的具体权限：主体（principal）
    可验证的身份
        成员服务提供者（Membership Service Provider）
            身份格式
            签名算法
            签名验证算法
            一组规则
            一组 admin 身份集合
        公钥基础设施（Public Key Infrastructure）
            包含与当事人相关的属性的电子文档
            公钥包含在证书中
            通过加密技术防止证书被篡改
            数字签名
11. 公钥基础设施的基本要素
    数字证书、公钥与私钥、证书颁发机构、废弃证书列表
12. 证书颁发机构（Certificate Authority）
    受系统信任的权威机构
    证书由 CA 签名
    CA 自身的证书
13. 根 CA、中间 CA 和信任链
    中间 CA 的证书由根 CA 颁发
    中间 CA 可以给下一级中间 CA 颁发证书
    不同的组织可能使用不同的根 CA，或有着相同根 CA 的多个中间 CA
14. Fabric CA
    Fabric 内置的 CA 组件，负责 Fabric 网络内所有实体（Identity）身份的注册，负责数字证书的签发、续签或撤销



15. 证书废弃列表（Certificate Revocation Lists）
    一个证书引用列表
    第三方想验证另一方的身份时，首先检查颁发证书的 CA 的 CRL，以确保证书没有被吊销
    已吊销的证书可能没有过期
16. 成员服务提供商 MSP
    识别被信任的 CA，定义信任域中的成员
    识别参与者的特定角色
    为定义访问权限奠定基础
    将 MSP 映射到组织
        组织：受管理的成员集合
        一个组织可以有多个成员组
    组织单元和 MSP

一个组织可分为多个组织单位（organizational unit），每个组织单位负责不同业务
CA 颁发证书时会指定身份所属的业务线

本地 MSP

定义节点（peer 节点或排序节点）和成员
每个节点或用户只有一个本地 MSP

通道 MSP

定义通道级别的管理及参与者权限
逻辑上定义一次，每个节点的本地文件系统上存有通道 MSP 的副本

MSP 等级

网络 MSP（Network MSP）
通道 MSP（Channel MSP）
对等 MSP（Peer MSP）
排序 MSP（Orderer MSP）

共识算法

17. 共识

共识算法：分布式系统中大部分节点对于某个提案（proposal）达成一致意见的过程。
确定性状态机模型（又称状态机复制，state-machine replication）：各个节点从相同的状态开始接收相同顺序的指令，则可以保证相同的结果状态。关键在于对多个事件的顺序进行共识，即排序。

18. Fabric VS Bitcoin

Fabric 中的记账节点都是可信的—— 不需要 PoW 算力证明
各节点的交易信息统一由排序节点处理—— 避免分叉问题

19. Fabric 的共识过程

背书阶段

客户端应用程序提交交易提案，根据背书策略发送给指定的背书节点
背书节点调用链码执行提案
背书节点调用系统链码对执行结果签名，返回响应给客户端应用程序

排序阶段

排序服务接收已背书过的交易，根据共识算法配置策略确定交易的顺序
将交易打包到区块中
将区块广播给通道中的成员

验证阶段

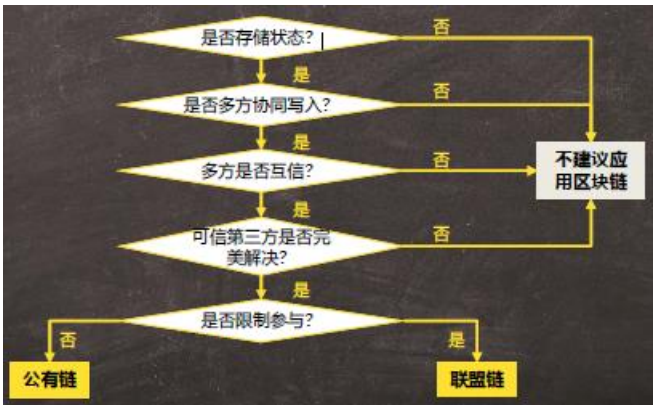
Peer 节点接收到广播的区块后对交易进行检验
交易通过验证后，区块加入区块链中

潜在的校验失败

1.语法错误：无效输入、未验证的签名、重复的交易等
2.逻辑错误：导致重复交易或版本控制的交易

第七章 区块链的应用

区块链应用判断标准



1. 适合区块链的应用场景
   存在去中心化、多方参与和写入数据需求
   对数据真实性有较高的要求
   存在初始情况下相互不信任的多个参与者建立分布式信任的需求
2. 存储状态
   区块链账本：存储“交易”状态
   不适合上链的数据：
      不需要共享的数据
      过于庞大的数据、更新后过于频繁的数据
   适合上链的数据：
      需要共享的数据
      需要具备可信度、不能被篡改的数据
      需要可追溯的数据
3. 去中心化
   中心化系统的弊端
      权力过于集中，容易出现“腐败”
      数据中心容易成为性能瓶颈，容易造成“数据孤岛”
      抗攻击能力差
   去中心化系统
      数据多副本备份
      多方相互监督，实现数据自治
      各参与方有预先规定的写入权限，相互制衡
4. 建立信任
   互联网：传递信息，难以传递“信任”
   区块链有助于建立信任，成本低
   若各方互信，则不需要用区块链建立信任
   传统的信任机制：信任根基不牢固
      自建的信息系统
      可信第三方（Trusted Third Party, TTP）
5. 可信第三方
   建立信任关系的纽带
   TTP 的缺陷
      门槛高
      接入运营的复杂度高
      权力过于集中，容易出现“腐败”

抗攻击能力强

若 TTP 的缺陷都可以接收，则不需要应用区块链

6. 限制参与

公有链

准入要求不高

联盟链

准入要求较高

有一定的信任基础，彼此不完全信任对方

各方相互监督

典型应用

7. 电子证照

行业现状及痛点

证照数据质量不高

证照标准不健全

缺乏证照汇聚机制

区块链价值

电子证照数据可信共享互换

证照全生命周期管理：验证、发证、管证

为电子证照授权应用、跨域互认证提供支撑

8. 防伪溯源

行业现状及痛点

溯源系统简单，不支持私有化部署

不能根据应用场景及需求灵活适配标识技术

无法同时满足不同部门的应用要求

区块链价值

保证溯源信息完整性

穿透式端到端流通监管

交易透明化，更加快捷高效