第三章、区块链技术架构与发展趋势

何德彪

武汉大学

国家网络安全学院

目 录

目 录

3.1 区块链基础技术发展历程

区块链的诞生最早可以追溯到密码学和分布式计算。

1976 年，迪菲和赫尔曼发表了一篇开创性论文《密码学的新方向》（*New Directions in Cryptography*），首次提出公共密钥加密协议与数字签名概念，构成了现代互联网中广泛使用的加密算法体系的基石，同时这也是加密数字货币和区块链技术诞生的技术基础。

同年，哈耶克出版了《货币的非国家化》，哈耶克从经济自由主义出发，认为竞争是市场机制发挥作用的关键，而政府对货币发行权的垄断对经济的均衡造成了破坏，通过研究竞争货币制度的可行性和优越性，哈耶克提出非主权货币（货币非国家化）、竞争发行（由私营银行发行竞争性的货币，即自由货币）等概念，从理论层面引导去中心化加密数字货币技术的发展。

3.1 区块链基础技术发展历程

1977 年 4 月，罗纳德·李维斯特(Ron Rivest)、阿迪·萨莫尔(Adi Shamir)和伦纳德·阿德曼(Leonard Adleman)参加了犹太逾越节的聚会，喝了些酒。回到家后李维斯特怎么都睡不着，于是信手翻阅起心爱的数学书来，这时一个灵感从他脑海浮现出来，于是连夜整理自己的思路，一气呵成写出了论文 *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*，次日李维斯特将论文拿给阿德曼审阅讨论，已经做好了再一次被击破的心理准备，但这一次阿德曼却认输了，认为这个方案应该是可行的。在此之前阿德曼已经四十多次击破李维斯特和萨莫尔的算法。按照惯例，李维斯特按姓氏字母序将三人的名字署在论文上，也就是阿德曼、李维斯特、萨莫尔。这篇论文提出了大名鼎鼎的 RSA 算法，RSA 是一种非对称加密算法，后来在数字安全领域被广泛使用，这一工作成果被认为是《密码学新方向》的延续。

3.1 区块链基础技术发展历程

1979 年，Merkle Ralf 提出了 Merkle-Tree 数据结构和相应的算法，现在被广泛应用于校验分布式网络中数据同步的正确性，对密码学和分布式计算的发展起着重要作用，这也是比特币中用来做区块同步校验的重要手段。Merkle Ralf 是《密码学新方向》的两位作者之一 Hellman 的博士生（另一位作者 Diffie 是 Hellman 的研究助理），实际上《密码学的新方向》就是 Merkle Ralf 的博士生研究方向。

1982 年，莱斯利·兰伯特（Lamport）提出拜占庭将军问题，并证明了在将军总数大于 3f，背叛者个数小于等于 f 时，忠诚的将军们可以达成一致，标志着分布式计算理论和实践正逐渐走向成熟。

同年，大卫·乔姆公布了密码学支付系统 ECash，随着密码学的发展，具有远见的加密数字货币先驱们开始尝试将其运用到货币、支付等相关领域，ECash 是加密数字货币最早的先驱之一。

3.1 区块链基础技术发展历程

1985 年，Koblitz 和 Miller 各自独立发明了著名的椭圆曲线加密算法。由于 RSA 的算法计算量大，实际落地时遇到困难，ECC 的提出极大地推动了非对称加密体系真正进入生产实践领域并发挥巨大影响。ECC 算法标志着现代密码学理论和技术开始走向更加普遍的应用。

1997 年，Adam Back 提出了 Hashcash 算法，用于解决垃圾邮件（email spam）和 DoS（Denial-of-Service）攻击问题，Hashcash 是一种 PoW 算法，后来被比特币系统采纳使用。

1998 年，华裔工程师戴伟（Wei Dai）和尼克·萨博各自独立提出加密数字货币的概念，其中戴伟的 B-Money 被公认为比特币的精神先驱，而尼克·萨博的比特黄金（Bitgold）设想基本就是比特币的雏形，以至于至今仍有人怀疑萨博就是中本聪，但被尼克·萨博本人否定了。

3.1 区块链基础技术发展历程

21 世纪初，点对点分布式网络技术飞速发展，先后诞生了 Napster、BitTorrent 等流行应用，为加密数字货币实现夯实了技术基础。

2008 年 11 月，神秘的中本聪先生发表了论文，描述了一种完全去中心化的加密数字货币——比特币，而区块链则作为其底层技术进入公众视野。经过十年发展，区块链技术正逐渐成为最有可能改变世界的技术之一。

区块链的发展先后经历了密码货币、可编程区块链和价值互联网三个阶段，下面将分别对这几个阶段进行简要的介绍.

目 录

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

2009 年 1 月，在比特币系统论文发表两个月之后，比特币系统正式运行并开放了源码，标志着比特币网络的正式诞生。通过其构建的一个公开透明、去中心化、防篡改的账本系统，比特币开展了一场规模空前的加密数字货币实验。在区块链 1.0 阶段，区块链技术的应用主要聚集在加密数字货币领域，典型代表即比特币系统以及从比特币系统代码衍生出来的多种加密数字货币。

加密数字货币的“疯狂”发展吸引了人们对区块链技术的关注，对于传播区块链技术起到了很大的促进作用，人们开始尝试在比特币系统上开发加密数字货币之外的应用，比如存证、股权众筹等。但是比特币系统作为一个为加密数字货币设计的专用系统，存在如下的问题：

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

（1）比特币系统内置的脚本系统主要针对加密数字货币交易而专门设计，不是图灵完备的脚本，表达能力有限，因此在开发诸如存证、股权众筹等应用时，有些逻辑无法表达，而且比特币系统内部需要做大量开发，对开发人员要求高、开发难度大，因此无法进行大规模的非加密数字货币类应用的开发。

（2）比特币系统在全球范围内只能支持每秒 7 笔交易，交易记账后追加 6 个区块才能比较安全地确认交易，追加一个块大约需要 10 分钟，意味着大约需要 1 小时才能确认交易，不能满足实时性要求较高的应用的需求。

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

针对区块链 1.0 存在的专用系统问题，为了支持如众筹、溯源等应用，区块链 2.0 阶段支持用户自定义的业务逻辑，即引入了智能合约，从而使区块链的应用范围得到了极大拓展，开始在各个行业迅速落地，极大地降低了社会生产消费过程中的信任和协作成本，提高了行业内和行业间协同效率，典型的代表是 2013 年启动的以太坊系统。

举个例子：保险

有了智能合约系统的支持，区块链的应用范围开始从单一的货币领域扩大到涉及合约共识的其他金融领域，区块链技术首先在股票、清算、私募股权等众多金融领域崭露头角。比如，企业股权众筹一直是众多中小企业的梦想，区块链技术使之成为现实。区块链分布式账本可以取代传统的通过交易所的股票发行，这样企业就可以通过分布式自治组织协作运营，借助用户的集体行为和集体智慧获得更好的发展，在投入运营的第一天就能实现募资，而不用经历复杂的 IPO 流程，产生高额费用。

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

各种区块链系统采用不同的共识方法以提升区块链的性能，比如以太坊采用改进工作量证明机制将出块时间缩短到了 15 秒，从而能够满足绝大多数的应用，以太坊未来拟采用的 PoS 共识算法可进一步提升区块链的性能。

- 比特币是基于P2P架构的数字货币系统：它的架构总体上分为两部分，一部分是前端，包括钱包(Wallet)或图形化界面；另一部分是运行在每个节点的后台程序，包括挖矿、区块链管理、脚本引擎以及网络管理等功能.
- 详细架构如下图所述：

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

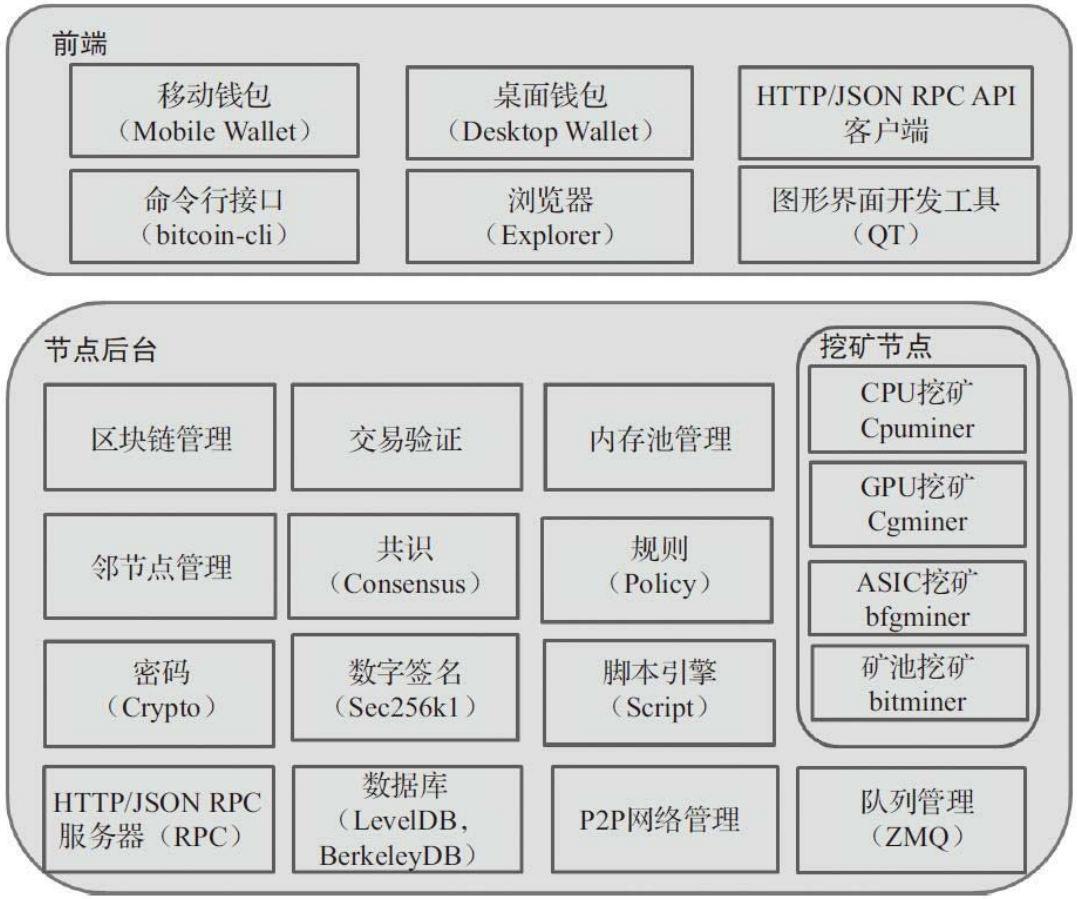

图3.1. 比特币系统架构示意图

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

一、比特币前端

1. 比特币前端-钱包

钱包保存用户的私钥数据库，并管理用户的余额，提供比特币交易(支付、转账)功能，一般可以分为冷钱包和热钱包.

- 冷钱包是指互联网不能访问到私钥的钱包：冷钱包往往依靠“冷”设备确保比特币私钥的安全，比如不联网的电脑、手机、写着私钥地址的小本本等.冷钱包避免了被黑客盗取私钥的风险，但是可能面临物理安全风险，比如电脑丢失损坏等.
- 热钱包是指互联网能购访问私钥的钱包：热钱包往往是在线钱包的形式.使用热钱包时，最好在不同平台设置不同密码，且开启二次认证，以确保自己的资产安全.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

从部署场景来说分，钱包可以分为4种：移动钱包、桌面钱包、互联网钱包，以及纸钱包.

- 移动钱包指的是运行在智能手机、移动终端上的轻量级钱包.
- 桌面钱包指的是运行在普通电脑上的钱包.
- 互联网钱包指的是依托第三方平台对用户密钥进行保护的钱包.
- 纸钱包指的是将私钥进行冷备份，可用于防范电脑或USB介质损坏所造成的私钥丢失，也能防范黑客通过网络攻击盗窃私钥，但需防范纸钱包被人物理偷窃或复制.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

2. 比特币前端- **HTTP/JSON RPC API**

- JSON-RPC是一种无状态，轻量级的远程过程调用协议(Remote Procedure Call, RPC).该规范主要定义与处理过程相关的数据结构和规则，它是与传输方式无关的，可在Socket，HTTP或在各种消息传送环境内使用.它只是将JSON(RFC 4627)作为数据格式.
- 比特币提供HTTP/JSON RPC API接口，供外部通过接口控制比特币节点.缺省情况下该服务端只允许来自同一机器的客户端访问. 使用不同语言编写的程序可以方便地通过HTTP/JSON RPC API接口访问比特币节点.例如，Bitcoin-JSON-RPC-Client是一个轻量级的Java客户端程序.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

3. 比特币前端-命令行工具**bitcoin-cli**

bitcoin-cli提供一个命令行工具来控制比特币节点.该命令行工具通过JSON RPC API接口访问比特币后台bitcoind.用户可以通过发命令来完成比特币的各项功能，例如查询余额、支付、转账等.

4. 比特币前端-比特币浏览器**bx**

比特币提供一个跨平台的C++libbitcoin库，该库支持比特币全节点服务端和浏览器(BitCoinExplorer)作为客户端命令行工具.比特币浏览器命令提供与bitcoin-cli一样的基本功能.但同时bx提供bitcoin-cli没有的一些密钥管理功能和处理工具.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

5. 比特币前端-图形开发工具(Qt)

- 比特币使用最广的客户端，是使用C++开源用户界面开发工具Qt所开发的桌面客户端.
- Qt是一个跨平台的C++图形用户界面应用程序框架.它提供给开发者建立图形用户界面所需的功能，广泛用开发GUI程序，也可用于开发非GUI程序.Qt是完全面向对象的，很容易扩展，并且允许真正的组件编程.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

二、比特币后端

比特币节点后端负责参与比特币网络的通信互联，维护区块链，验证区块、交易，广播、转播传递区块交易信息. 比特币的后台程序主要是由bitcoind，以及挖矿节点程序构成. 比特币核心bitcoin-qt实际上是包含前后端(除挖矿功能以外)的一体化节点.

1. 比特币节点后端-区块链管理

区块链管理的代码逻辑都在main.cpp程序中实现. 主要包括4个部分：

- **(1)下载区块链：**在比特币全节点第一次加入网络运行时，先要下载并验证整条区块链. “区块报头先行”(header first)，一个初始区块链下载方式，新的节点先从邻节点下载所有的区块报头，再并行地从多个邻节点同时下载不同区间的区块大大提升了整条区块链的下载速度.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

- **(2)接收区块链**：现有节点在开启时**(非第一次)**会先将整个区块链的索引从LevelDB调进内存. 需要注意的是，该区块链的索引不是单条的链，而可能是一个树，也就是说每个区块只有一个父区块，但可能有多个子区块，因为子区块形成暂时分叉，需要逐渐发现哪个子区块属于最长的链条.

- **(3)区块链验证**：在区块链管理中，连接区块函数ConnectBlock()是一个检测“双花”交易的关键.该函数对新接收的区块中的所有交易进行检测，验证是否每个交易的比特币来源都能在当前的“尚未花比特币”(Unspend Transaction Output，UTXO)记录中找到匹配.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

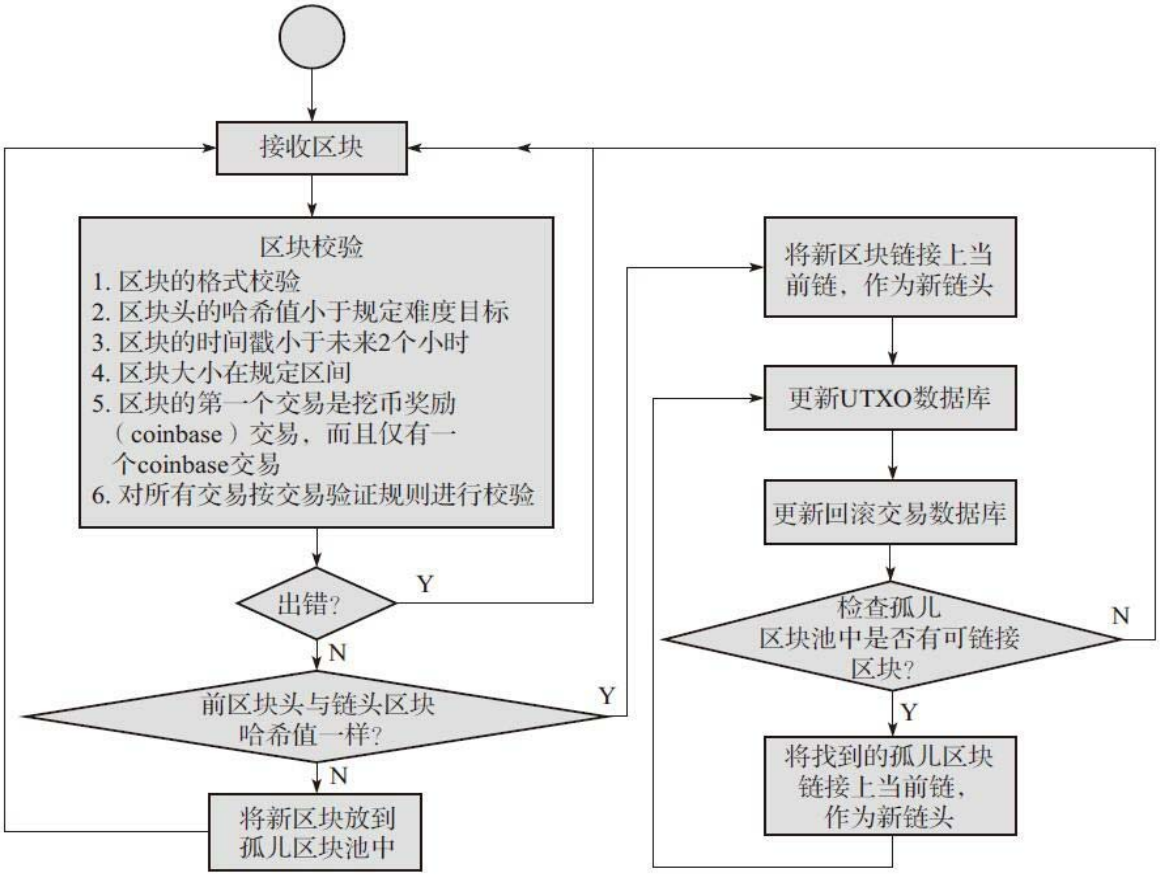

图3.2. 区块验证流程图

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

➢ **(4)重组区块链：**当节点发现网络中有一条不基于它当前区块链的一条更长区块链时，它需要断开现有的区块并对区块链进行重组.

◆大部分重组只是一个区块的重组，多发生在不同矿工几乎同时挖到合法的区块时.

◆断开区块、重组区块链涉及UTXO更改，被断开的区块中交易会回退到交易内存池，这个时候“回滚”记录就可以用来回滚断开区块中的交易.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

2. 比特币节点后端-区块验证

交易验证模块会基于以下条件检查收到的比特币交易是否合规：

- 交易的格式是数据结构必须正确；
- 交易的输入和输出不能为空；(**Coinbase交易没有输入**)
- 交易的大小不能超过定义的区块最大值MAX_BLOCK_SIZE(**最早为1MB，后来逐渐调整**)；
- 每个交易的输出，以及所有输出的合计，必须在一定范围内，也就是**大于0，小于2100万**；
- 交易输入的哈希值不能为零，不应该转Coinbase交易；
- nLockTime小于等于INT_MAX；

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

- 交易的字节大小要等于或大于100；
- 交易的签名操作数要小于签名操作的上限；
- 锁定脚本(scriptPubkey)必须是标准格式；
- 和收到的交易相匹配的交易必须能在当前交易池或是主链上某个区块中找到；
- 对交易的每个输入，如果其对应的UTXO输出能在当前交易池中找到，该交易必须拒绝(双花交易)，因为当前交易池是未经记录在区块链中的交易，而交易的每个输入应该来自确认的UTXO，如果在当前交易池中找到，那就是双花交易；
- 对交易的每个输入，如果其对应的UTXO输出不能在主链或当前交易池中找到，该交易是一个孤儿交易，应将该交易放入孤儿交易池中；

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

- 对交易中的每个输入，如果其对应的UTXO输出是一个挖矿初始(coinbase)交易，该初始交易应该获得**100个确认区块的确认(普通交易是6个确认区块的确认)**；
- 对交易中的每个输入，其对应的输出必须是UTXO(存在且没被花掉)；
- 用对应的输出交易来获得输入的值，检查每个输入的值及其合计，应该在允许的区间(0~2100万比特币)；
- 如果一个交易的输入合计小于输出总计，则拒绝该交易；
- 如果交易的费用太低，则拒绝该交易；
- 每个输入的解锁脚本(unlocking script)必须和相应输出的锁定脚本(locking script)共同验证交易的合规性.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

最后这个检查是比特币平台设计的精髓.比特币的一个很大的创新是依靠脚本来验证交易的合法性，即每一个将要花掉的比特币必须有相应的来源.

- 锁定脚本是由一连串堆栈命令和公钥哈希组成，公钥哈希即RIPEMD160(SHA256(公钥))，大小20字节；锁定脚本格式为：OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECKSIG
- 解锁脚本是由签名与公钥组成，这就保证了必须拥有私钥的用户才能对某一笔交易进行解锁.解锁脚本格式为：<Sig> <PubKey>

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

比特币是将解锁脚本和锁定脚本串起来一同执行，**如果最后结果是逻辑值“真”(TRUE)，则验证该交易的比特币来源是合法的**. 脚本执行的过程和每个脚本执行后的堆栈状态如下：

① 由于是堆栈式计算引擎，因此作为数值的<sig>和<pubKey>相继入栈.当执行脚本OP_DUP时，它将堆栈头的<pubKey>复制一份也压入堆栈.

② 脚本OP_HASH160执行，将把堆栈头的<pubKey>弹出，并用HASH160算法进行哈希处理，哈希结果放回堆栈.然后遇到<pubKeyHash？>数值，该数值也被压入堆栈.

③ 脚本OP_EQUALVERIFY把堆栈头端的两个哈希值都弹出，并进行比较.如果不一样，验证就出错，交易不合法.如果验证通过，堆栈只剩下<sig>和<pubKey>.

④ 最后脚本OP_CHECKSIG将两者弹出，执行检查签名的脚本，验证该交易签名是否是由拥有该公钥对应用户用其私钥签的，如果是，交易就是合法交易，否则就是不合法交易.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

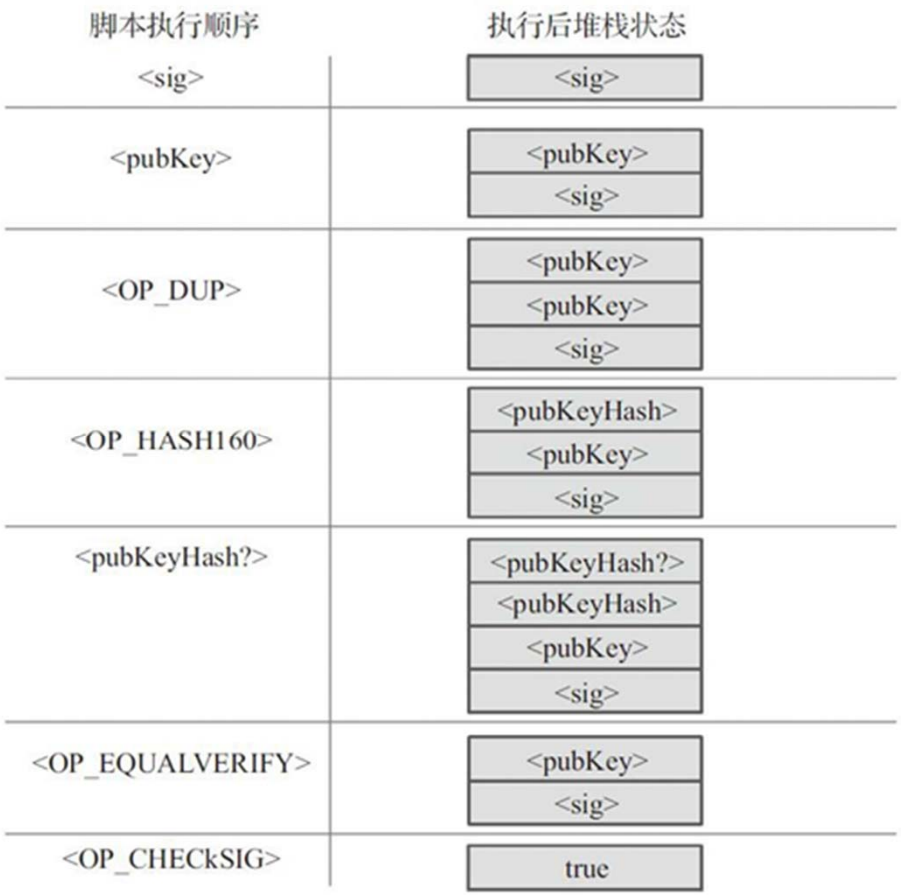

图3.3.脚本执行顺序和堆栈状态图

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

3. 比特币节点后端-内存池管理

比特币内存池(mempool)管理也就是交易池管理.

- 节点将通过验证的交易放在一个交易池中，准备放在一个挖到的区块中.当准备挖矿时，它按一定的优先级次序从交易池中选出交易.
- 优先级是按交易中的输入对应的**UTXO的“链龄”**和交易额的大小来划分的.越老UTXO的交易以及交易额越大的交易优先级会越高.
- 优先级(Priority)采用以下公式计算：Priority=Sum(Value of input×InputAge)/ Transaction Size，其中Value of input是按比特币基础单位**(satoshi，聪)**计算，1个satoshi等于一亿分之一(10^{-8})个比特币. InputAge按已在链上记录该交易的区块为起点，按后面有多少个区块来计算，也就是计量该区块在区块链的“深度”.交易的大小以字节为单位.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

- 当区块填满后，剩下的交易会留在内存池，等待下一个区块的到来.随着它们的“链龄”的逐渐增加，它们以后被选中的几率也会逐渐增加.
- 内存池的比特币的交易不会过期，但是内存池的交易不保存在硬盘上，当挖矿节点重启时，内存池的交易会被清空.
- 如果在一定时间内一个交易一直不能被矿工包括在区块链上，钱包软件需要重新发送该交易，可以附上较高的交易费，获得优先权.
- 在一些比特币节点的实现也维护一个独立的“孤儿”交易池.如果一个交易的输入相对应的UTXO不能被找到，也就是没有“父”交易，会被当作“孤儿”交易，暂时放在“孤儿”交易池.当父交易来到后，该“孤儿”交易会被从“孤儿”交易池移到内存池.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

4. 比特币节点后端-邻节点管理

当一个新比特币节点做初始启动(bootstrap)的时候，它需要发现网络中的其他节点，并与至少一个节点连接.

- 一般是与一个已知的节点在8333端口建立TCP连接.
- 连接的“握手”流程发送一个版本信息，包括：P2P协议版本，本节点支持的服务，当前时间，对方节点IP地址，本节点IP地址，比特币软件版本，以及本节点当前区块链的长度.
- 对方节点收到握手信息后会回复一个收到确认的信息.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

新节点如何发现邻节点：

- 第一个办法是用一些“DNS种子”查询DNS.
- 第二个办法是直接把一个已知的邻节点作为种子节点，然后通过它发现更多的邻节点.
- 如果一个连接上一段时间内没有信息交互，节点会定期发一些信息去维护连接.
- 如果一个节点和邻节点的连接在超过90分钟里没有联系，该邻节点会被认为下线，节点会寻找一个新的邻节点来进行连接.

比特币网络能动态地调节节点的连接，以保证比特币网络的正常运行.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

5. 比特币节点后端-共识管理

比特币里广义的共识管理(Consensus)应该包括挖矿、区块验证和交易验证规则.

- 由于比特币的关键是在陌生P2P环境建立共识机制，因此共识管理至关重要.
- 在比特币0.12.0版本中，一部分共识管理的代码已经移到consensus子目录.
- 目前在consensus子目录的共识程序有consensus.*、merkle.*、params.*和validation.*.

6. 比特币节点后端-规则管理

比特币的共识规则是所有节点都必须遵守的规则(policy)，而每个节点可以采用一些共识规则以外的个性化规则（比如一个节点可以拒绝保存、中转大于200KB的交易）.这部分的规则由规则管理模块实现，目前放在policy子目录中.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

7. 比特币节点后端-密码模块

密码模块(Crypto)主要是处理比特币地址，采用RIMEMD和SHA-256算法以及Base-58编码来生成比特币地址.

- 比特币的公钥是通过私钥产生的，然后采用Secure Hash Algorithm(SHA)算法SHA256和RACE Integrity Primitives Evaluation Message Digest(RIPEMD)算法RIPEMD160对公钥进行处理，最后通过Base58编码形成比特币地址.

8. 比特币节点后端-签名模块

比特币采用椭圆曲线数字签名算法(ECDSA)来实现数字签名以及生成公钥.ECDSA是一种非对称密码算法，是基于椭圆曲线离散对数问题困难性的一种签名算法.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

9. 比特币节点后端-脚本引擎

比特币的脚本语言是一种专门设计的、与“Forth”类似的、基于堆栈的编程脚本语言.

- 基于堆栈的语言的指令只按顺序执行一次，也就是说没有循环或跳转指令， 因此脚本的指令数可以决定一个程序运行时间的上限和所需的内存上限.
- 比特币的脚本语言非常小，只能有256个指令，每个指令是一个字节长.这256个指令中，75个是保留指令，15个已废弃.
- 脚本引擎是校验交易的运算平台，从对解锁脚本和锁定脚本的自动执行校验可以看出该引擎的重要作用.
- 新版本的比特币将脚本引擎放在script子目录中，将来可以变成可插拔引擎，使得引入新的功能更强大的引擎更为方便，不影响原有比特币的代码.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

- 目前script子目录里有interpreter.*、script.*和standard.*程序.script.h定义了脚本命令，interpreter.cpp解析、评估和较验脚本命令，standard.h定义了标准的交易.
- 比特币通常使用的指令如下：

OP_DUP	将堆栈头上的内容复制一份并压入堆栈
OP_HASH160	弹出堆栈头内容，先用SHA256对其做哈希处理，再用RIPEMD-160对结果做第二次哈希处理，结果压入堆栈
OP_EQUALVERIFY	弹出堆栈头的两项内容，如果两个内容一样，返回“真”值；否则返回“假”值
OP_CHECKSIG	用输入的公钥检查输入的签名，如果签名符合，返回“真”值，否则返回“假”值
OP_CHECKMULTISIG	用提供的多个公钥检查多重签名的正确性

图3.4. 比特币常用指令

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

10. 比特币节点后端-挖矿

比特币最早的挖矿程序是cpuminer，是通过CPU来挖矿的.

- 挖矿设备经过CPU、GPU、FPGA，到现在基本都使用ASIC挖矿设备.
- 针对SHA256挖矿算法优化的ASIC挖矿设备比其他挖矿设备有着性能上无可比拟的优势.
- Bfgminer程序支持FPGA和ASIC挖矿设备，是目前比较流行的挖矿程序. **最新代码见：http://www.bfgminer.org/**
- 那如何把网络的出块速度稳定在10分钟一个呢？比特币网络是通过调整挖矿难度的目标来达到这个目的. 难度计算公式是： New Difficulty = Old Difficulty × (Actual Time of **Last 2016 Blocks** / 20160minutes)

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

11. 比特币节点后端-**HTTP/JSON RPC服务端**

比特币在启动的时候，初始化程序init.cpp会启动HTTP/JSON RPC服务端的线程组.

- 该组件**对外提供HTTP和JSON RPC的接口**，外部程序可以通过JSONRPC接口来调比特币的API，达到控制比特币节点的功能.
- 该接口缺省是仅接收来自本机的客户端的连接请求.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

12. 比特币节点后端-**Berkeley DB和Level DB数据库**

- Berkeley DB是一个开源的文件数据库(Sleepycat Software公司开发)，介于关系数据库与内存数据库之间，使用方式与内存数据库类似，它提供的是一系列直接访问数据库的函数，主要用来备份用户的密钥.
- Berkeley DB是一个轻巧而又性能高的嵌入式数据库，可以保存任意类型的键/值对，可以为一个键保存多个数据，可以支持数千个并发线程同时操作数据库，支持最大256TB的数据.
- Level DB用来存储区块的索引和**UTXO(未花的比特币交易输出)记录**，是一个**Google实现的非常高效的键值(Key Value)数据库**，目前的版本1.2能够支持几十亿级别的数据量.
- Level DB的数据是冗余数据，可以用原始区块链数据来重建.如果没有Level DB的数据，比特币的校验和其他操作都会变得非常缓慢.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

11. 比特币节点后端-**P2P**网络管理

P2P网络管理的代码主要是**在P2P网络上实现和其他邻节点的通信功能**.这些通信功能包括：发现邻节点；连接并管理与邻节点的Socket连接；与邻节点交换不同的P2P消息(包含区块和交易)；有时在特殊情况下，会禁止异常行为的邻节点的连接.

- 大部分的P2P代码集中在net.h/net.cpp.邻节点IP地址管理代码是addrman.h/addrman.cpp.地址管理程序把地址存放在peers.dat数据库中，在启动的时候再把它调入内存.
- 比特币节点的缺省配置是主动连接8个邻节点，同时允许最多125个其他节点发起的连接请求.
- 比特币防止拒绝服务攻击(DoS)的方法主要是禁止异常行为邻节点的连接. 如果一个邻节点传送非常明显的错误信息，该连接将被断开，而且其IP地址会被禁止，其重新连接申请会被拒绝.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

比特币节点按功能分有几种：**全功能节点、基础全节点和SPV节点.**

- **全功能节点**带有钱包、RPC服务端，具有挖矿功能和进行节点校验区块和交易，并把区块和交易中转给与之相连接的邻节点.
- **基础全节点**也做区块和交易的交易和中转，但不挖矿，不带钱包和RPC服务端.
- **SPV(Simplify Payment Verification)节点**信任别的节点来对区块和交易作校验.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

11. 比特币节点后端-队列管理

比特币采用Zero MQ作为消息队列管理和消息分发工具. Zero MQ号称是“史上最快的消息队列”，是基于C语言开发的.

- ZMQ是一个简单好用的传输层，提供像框架一样的一个socket library，它使得Socket编程更加简单、简洁，性能更高.
- ZMQ是一个消息处理队列库，可在多个线程、内核和主机盒之间弹性伸缩.
- ZMQ不是一个服务器，更像一个底层的网络通信库，在Socket API之上做了一层封装，将网络通信、进程通信和线程通信抽象为统一的API接口.

3.2 区块链平台发展历程

3.2.1. 区块链1.0：密码货币

11. 比特币节点后端-队列管理

比特币采用Zero MQ作为消息队列管理和消息分发工具. Zero MQ号称是“史上最快的消息队列”，是基于C语言开发的.

- ZMQ是一个简单好用的传输层，提供像框架一样的一个socket library，它使得Socket编程更加简单、简洁，性能更高.
- ZMQ是一个消息处理队列库，可在多个线程、内核和主机盒之间弹性伸缩.
- ZMQ不是一个服务器，更像一个底层的网络通信库，在Socket API之上做了一层封装，将网络通信、进程通信和线程通信抽象为统一的API接口.

目 录

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

比特币的区块链架构主要围绕支持密码货币的实现，虽然它有一定的灵活性，但用来支撑密码货币以外的应用场景还显得非常局限.

区块链2.0的核心理念是把区块链作为一个可编程的分布式信用基础设施，支撑智能合约应用，对金融领域更广泛的场景和流程进行优化的应用，与过去比特币区块链作为一个虚拟货币支撑平台区别开来.

在比特币后，出现很多被称为区块链2.0的平台，其中最具代表性的是以太坊平台.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

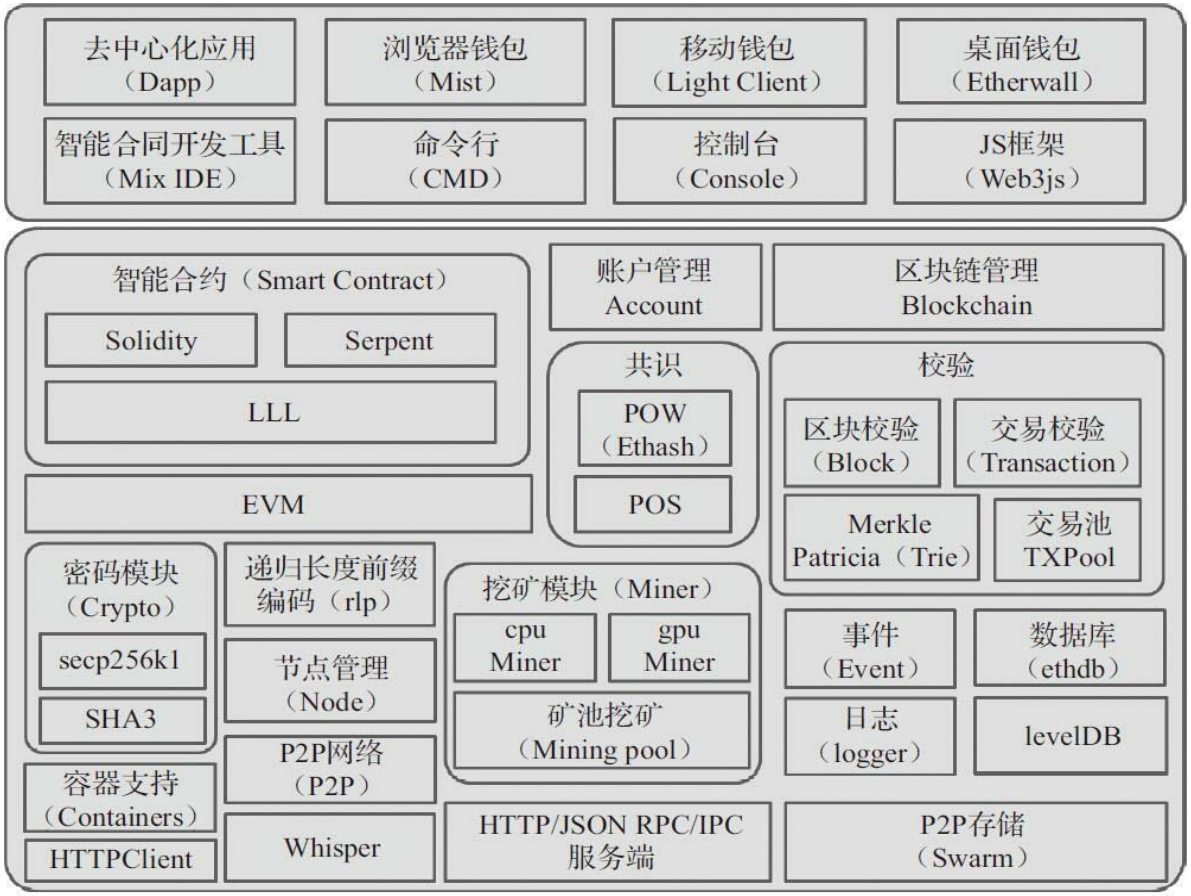

图3.5. 以太坊系统结构图

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

以下讨论以太坊和比特币架构不同的几个主要方面.

1. 账户设计

比特币没有账户的概念.每个用户的余额都是从他们在区块链上的UTXO计算出来的.以太坊的设计是将区块链作为一个通用的管理对象状态转换的去中心化平台，账户就是有状态的对象.以太坊则有两种类型的账户：一种是外部所有账户(EOA)，另一种是合约账户.

- 外部所有账户就是我们一般意义上的用户账户，它由私钥控制.
- 合约账户是一种特殊的可编程账户，合约存在以太坊区块链上，它是代码(功能)和数据(状态)的集合.合约受代码控制并由外部所有账户激活.
- 外部所有账户的状态就是余额，而合约账户的状态可以是余额、代码执行情况，以及合约的存储.
- 以太坊网络的状态就是所有账户的状态，该状态由每个区块的交易来更新，同时需在全网形成共识.用户和以太坊区块链的交互需要通过对账户的交易来实现.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

2. 区块链设计

比特币采用Merkle树来将交易的哈希值组成二叉树，顶层节点的哈希值相当于整个交易清单的指纹，可以用来校验交易清单.以太坊的区块链的每个区块**不但保存着交易清单，还保存最新的状态**.以太坊的状态包含一个键值表，其中键是地址，值是账户里声明的变量.

- 以太坊区块报文头中存放了**3个根哈希值**：一个是交易的Merkle根哈希值**(比特币只存这一个)**，另外一个是状态的根哈希值，还有一个是收据的根哈希值.
- 另外一个和比特币的不同是，以太坊的区块链中的**每个区块保存区块链号和区块难度**.
- 账户的状态经常被改变，新的账户也经常被插入，键在存储里也被经常插入和删除.还要求树的根哈希只是与树的数据有关，与更新的顺序无关. **Patricia(帕特里夏)树**是符合这些要求的数据结构，以太坊采用的Merkle Patricia树存储数据.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

Patricia树，或称压缩前缀树，是一种更节省空间的前缀树.对于基数树的每个节点，如果该节点是唯一的儿子的话，就和父节点合并.

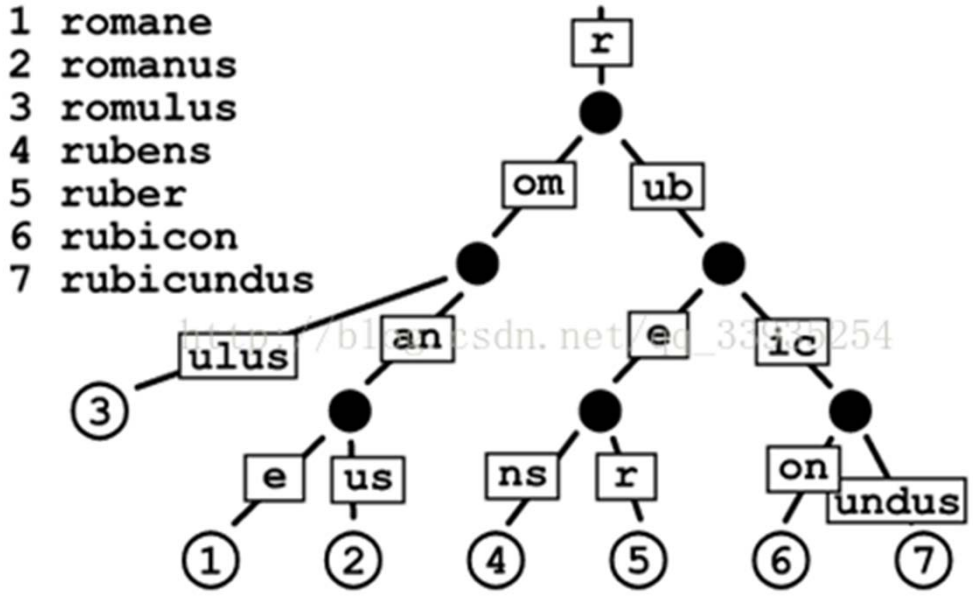

图3.6. Patricia树示意图

详见：https://blog.csdn.net/weixin_41545330/article/details/79394153

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

3. PoW机制

以太坊吸取了比特币的教训，专门设计了非常不利用计算的模型，它采用了I/O密集的模型，I/O慢，计算再快也没用，对专用集成电路则不是那么有效.以太坊的PoW(工作量证明)算法叫Ethash算法(是一个经过修改的Dagger-Hashimoto算法).

- 该算法对GPU友好.一是考虑如果只支持CPU，担心易被木马攻击；二是现在的显存都很大；
- 轻型客户端的算法不适于挖矿，易于验证；
- 快速启动算法中，主要依赖于Keccake256；
- 数据源除了传统的Block头部，还引入了随机数阵列DAG(Directed Acyclic Graph，有向无环图)(**Vitalik提出**)

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

- 一个世代(Epoch)，**含有30000Blocks**，一个世代里使用相同的随机数阵列.
- 随机数阵列分为三个层次，种子值，缓存值，数据.种子值很小.根据种子值生成缓存值，缓存层的初始值为16M，每个世代增加128K.在缓存层之下是矿工使用的数据值，**数据层的初始值是1G**，每个世代增加8M.
- 过程中使用了**Keccak-512算法**，结果为64Bytes.
- 核心部分使用的数据单位是128Bytes，数据层面每一个元素都依赖于缓存层的256个元素.

DAG 主要技术指标

- Epoch = 30000 Blocks
- Seed Hash[0] = KEC(32 Bytes 0),
  - Seed Hash[i]= KEC(SeedHash[i-1])
- Cache init = 16 MB, Cache Growth = 128 K 每世代,
- Data init = 1 GB Data Growth = 8M
- Hashbytes = 64 Bytes, KEC512
- Mixbytes = 128, Parents = 256

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

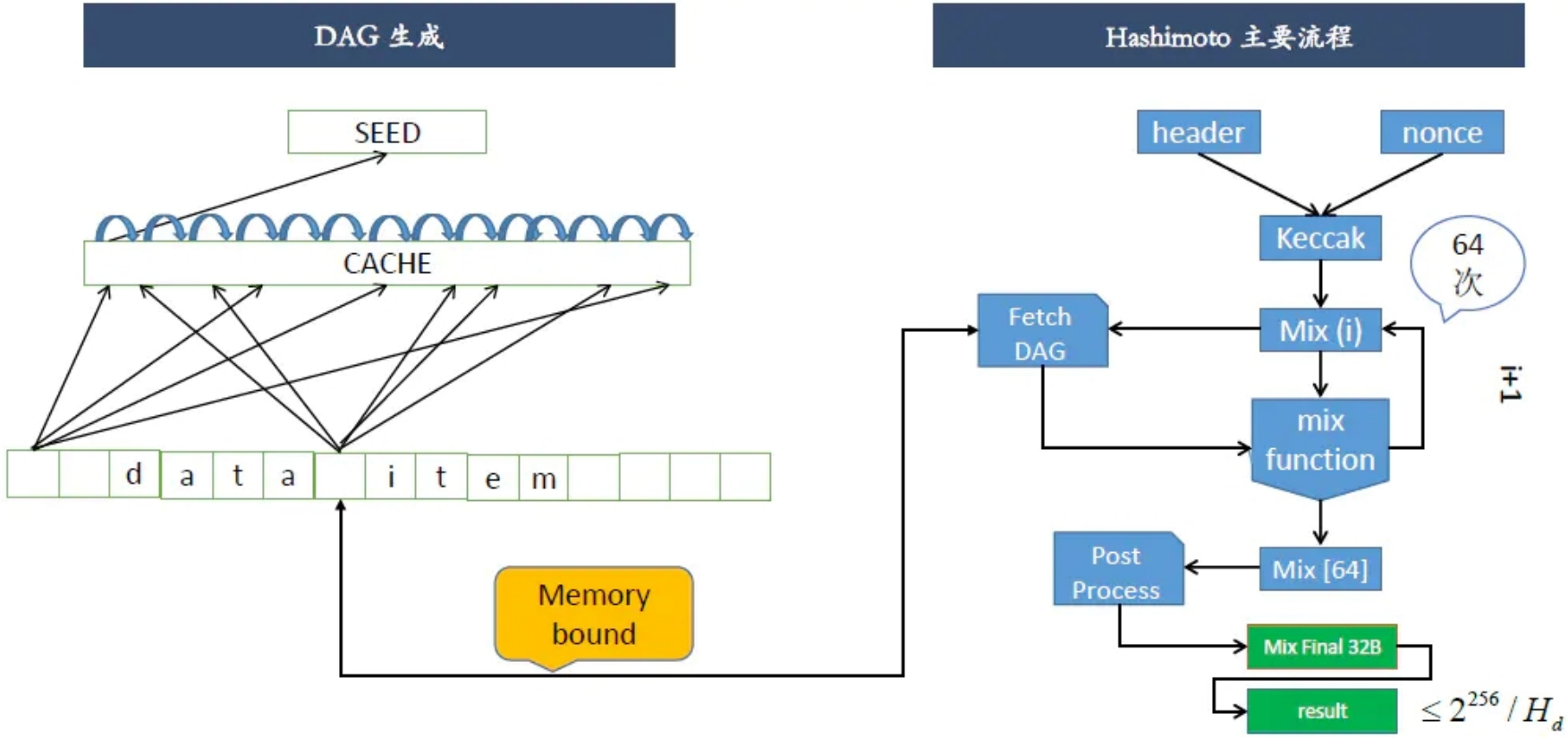

图3.7. ETHASH的框架

详见：https://www.jianshu.com/p/0c92d3b8173a

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

ETHASH框架主要分为两个部分，一是DAG的生成，二是用Hashimoto来计算最终的结果.

➢ **DAG主要流程**

DAG分为三个层次，种子层，缓存层，数据层.三个层次是逐渐增大的.

① 种子层很小，依赖上个世代的种子层.

② 缓存层的第一个数据是根据种子层生成的，后面的根据前面的一个来生成，它是一个串行化的过程.其初始大小是16M，每个世代增加128K.每个元素64字节.

③ 数据层就是要用到的数据，其初始大小1G，现在约2G，每个元素128字节.数据层的元素依赖缓存层的256个元素.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

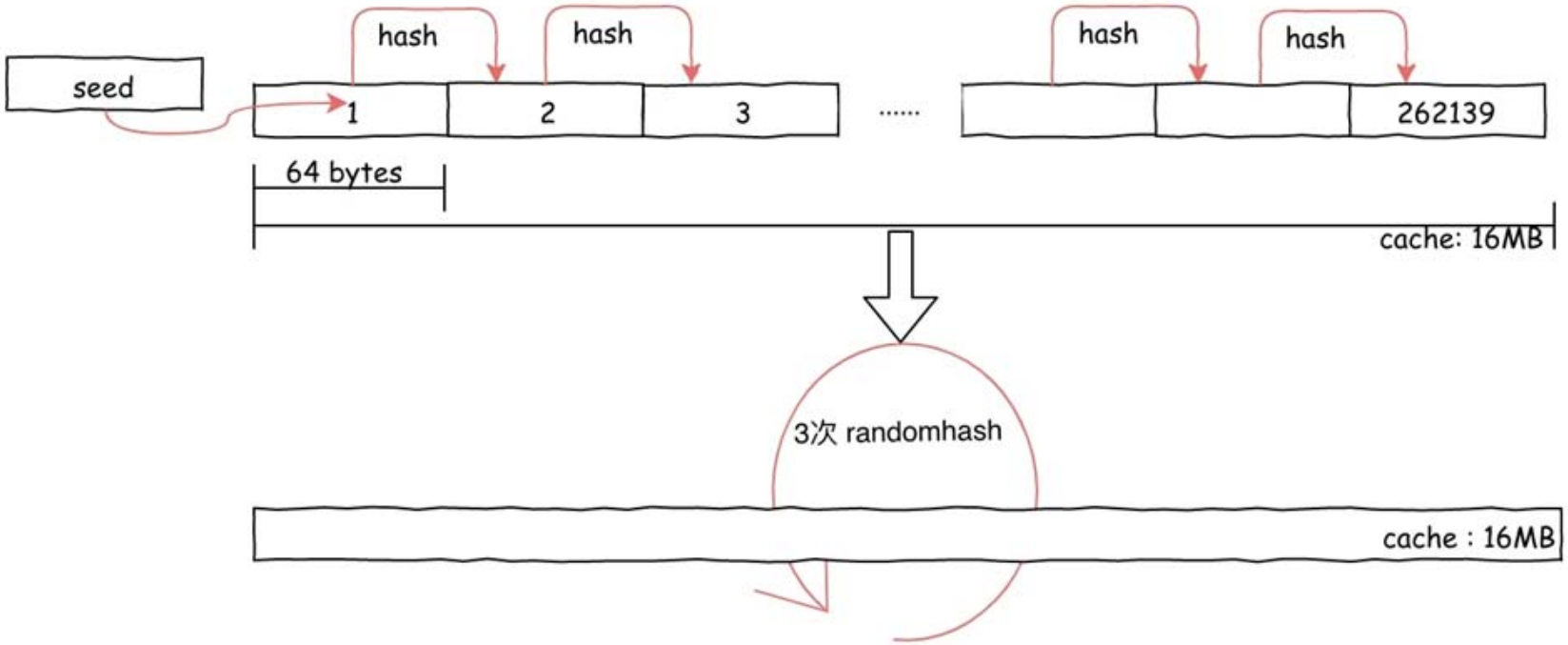

图3.8. 缓存层计算流程示意图

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

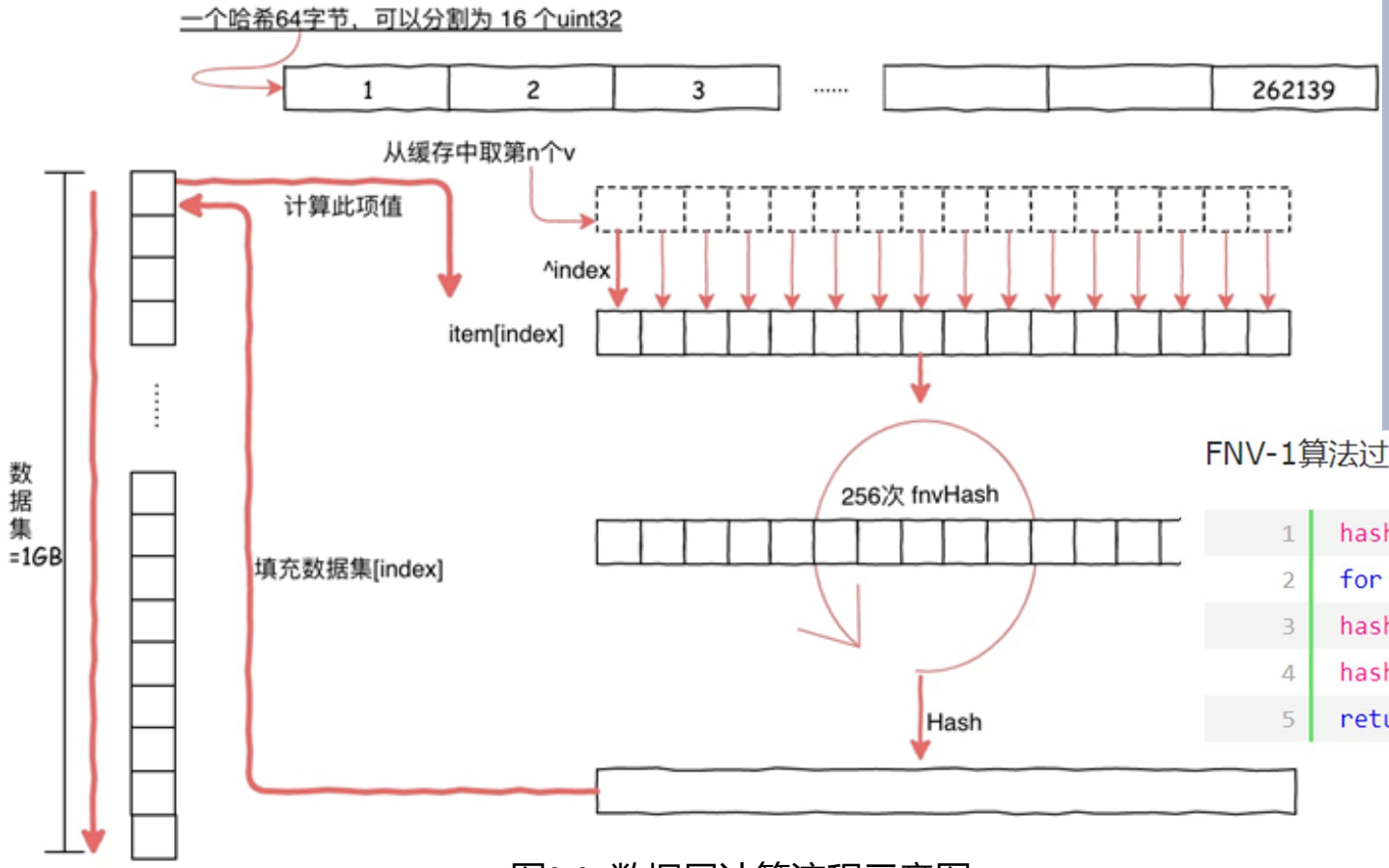

图3.9. 数据层计算流程示意图

FNV算法属于非密码学哈希函数，它最初由**Glenn Fowler和Kiem-Phong Vo**于1991年在IEEE POSIX P1003.2上首先提出，最后由Landon Curt Noll 完善，故该算法以三人姓的首

FNV-1算法过程如下：

```
1  hash = offset_basis
2  for each octet_of_data to be hashed
3  hash = hash * FNV_prime
4  hash = hash xor octet_of_data
5  return hash
```

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

➢ **Hashimoto 主要流程**

Hashimoto的整个流程是内存密集型，具体如下：

- 头部信息和随机数结合在一起，做一个**Keccak-512运算**，获得初始的单向散列值Mix[0]，128字节.
- 通过另外一个函数(Fetch DAG)，映射到DAG上，获取一个值与Mix[0]混合得到Mix[1]，循环64次后得到Mix[64]，128字节.
- 经过后处理过程，得到 mix final 值，32字节. 再经过计算，得出结果.
- 把它和目标值相比较，小于则挖矿成功. 难度值大，目标值小，就越难(前面需要的 0 越多).

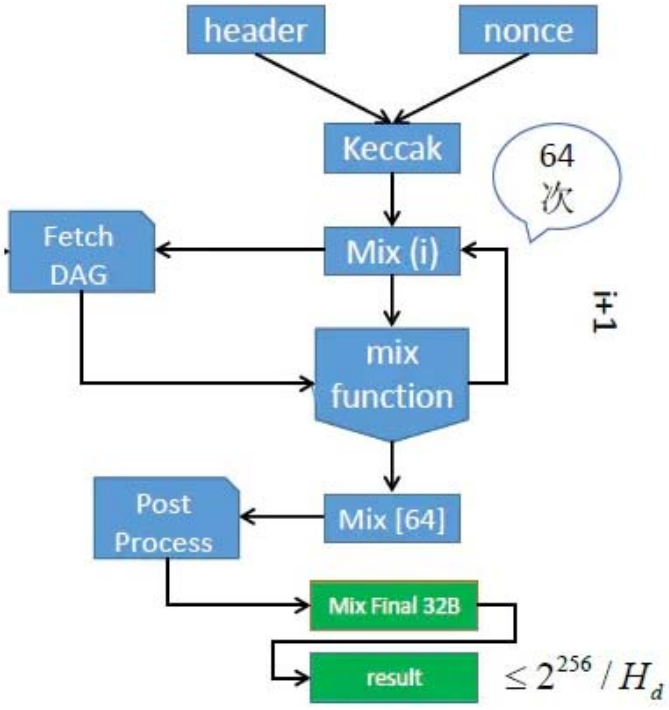

图3.10. Hashimoto主要流程示意图

注：为防止矿机，mix function函数也有更新过.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

➢验证流程

当矿工广播区块到网络中后，如何才能校验区块由完成一定工作量呢？

将采用 hashimotoLight 计算出在指定Nonce下的执行结果 result 和 digest.根据本地计算结果应该和区块的**MixDigest**值一致，且**result**低于给定的目标值 **target**，则说明Seal校验通过，表明该区块由完成一定量的工作量.

回到 hashimotoLight 方法，此方法是直接利用缓存实时计算出数据线来参与校验，和 hashimotoFull 类似. 因为数据集也是通过缓存生成，如果没有数据集可以直接使用缓存计算. 这样对于普通节点，只需要利用 **16MB** 的缓存便可以轻松完成 **PoW** 校验，按需生成所需要的数据集的数据项.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

4. 计算与图灵完备

以太坊作为通用的区块链平台，需要提供比比特币更强大的计算能力，选择了图灵完备的计算环境——以太坊虚拟机(Environment Virtual Machine EVM).这就意味着在EVM上可以做所有的能想得到计算，包括无限循环. 以太坊采用经济的方法来保证以太坊平台的安全：

- 以太坊要求每个交易要给出最大的计算步骤，交易的发起人要提供Gas作为交易费以供矿工把交易加进区块.
- 如果实际运行超过了该最大计算步骤，计算将被终止，而交易费会归挖到区块的矿工所有.
- 以太坊网络的每个节点都运行EVM并执行合约代码.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

5. EVM高级语言

比特币不提供高级语言的支持，以太坊则提供高级语言让用户编写智能合约.以太坊的高级语言最后会编译成在EVM中执行的EVM字节码(bytecode)，部署在以太坊区块链上.以太坊提供3种编程语言：**Solidity**、**Serpent**和**LLL**.

6. 以太坊P2P网络

(1)RLPx协议

以太坊节点间采用RPLx编码及认证的通信传输协议来传输消息包.节点可以自由地在任何TCP端口发布和接受连接，默认的端口是30303.

目前正式版的RLPx实现了以下功能：单一协议的UDP节点发现，ECDSA签名的UDP，加密握手/认证，节点持久性，加密/认证TCP，TCP帧处理.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

(2)Whisper协议

Whisper协议是DApp间通信的通信协议.Whisper结合了DHT (Distributed Hash Table)和数据包消息系统(如UDP)，因此同时具有以上两种协议的特性.Whisper提供多索引，非单一的记录，也就是说同一记录可以有多个键，有些键可能和别的记录一样.使用场景有以下几种：

- DApp需要把少量的信息发布出去，而这些发布的信息要保留相当一段时间.例如一个外汇交易所将一个货币的挂牌卖价发布出去，这个卖价可能需要保留几分钟或几天时间.
- DApp需要发信号给其他DApp，希望它们参与对某个交易的协同.
- DApp之间需要提供非实时的提示或通常的通信，例如聊天室应用等.
- DApp需要提供暗通信，也就是通信的双方除了知道对方的哈希值外，不知道对方更多的底细.

3.2 区块链平台发展历程

3.2.2. 区块链2.0：可编程区块链

7. 事件

以太坊中的事件是一个以太坊日志和事件监测协议的抽象.日志的记录中提供合约的地址，一组最多4个议题和一些任意长度的二进制数据.事件利用现有的应用程序二进制接口(ABI)功能来解析日志记录.

根据一个事件名和一些列的事件参数，可以分为两个系列：建立了索引的和没有索引的.建了索引的(最多有3个)是用来和事件的Keccak哈希签名一起作为议题的日志记录；没有建立索引的用来组成事件的字节数组.

目 录

3.2 区块链平台发展历程

3.2.3. 区块链3.0：可编程区块链

我们把超越货币、金融范围的区块链应用称为区块链3.0.区块链3.0可以为各种行业提供去中心化解决方案，应用领域扩展到人类生活的方方面面，在各类社会活动中红实现信息的兹证明，不再依靠某个第三人或者机构获取信任或者建立信任，实现信息的共享，包括在司法、医疗、物流等各个领域，区块链技术可以解决信任问题，提高整个系统的运转效率.

支持行业应用表示区块链平台必须具备企业级属，也就是安全性的考虑会更为突出.另外区块链3.0也需要图灵完备的智能合约平台，同时对网络和共识算法的性能、每秒交易数(TPS)都有比较高的要求.

目前业界还没有一个成熟的区块链3.0平台.

3.2 区块链平台发展历程

3.2.3. 区块链3.0：价值互联网

根据区块链3.0应用的一些通用需求而设计的一个初步的通用架构如下图：

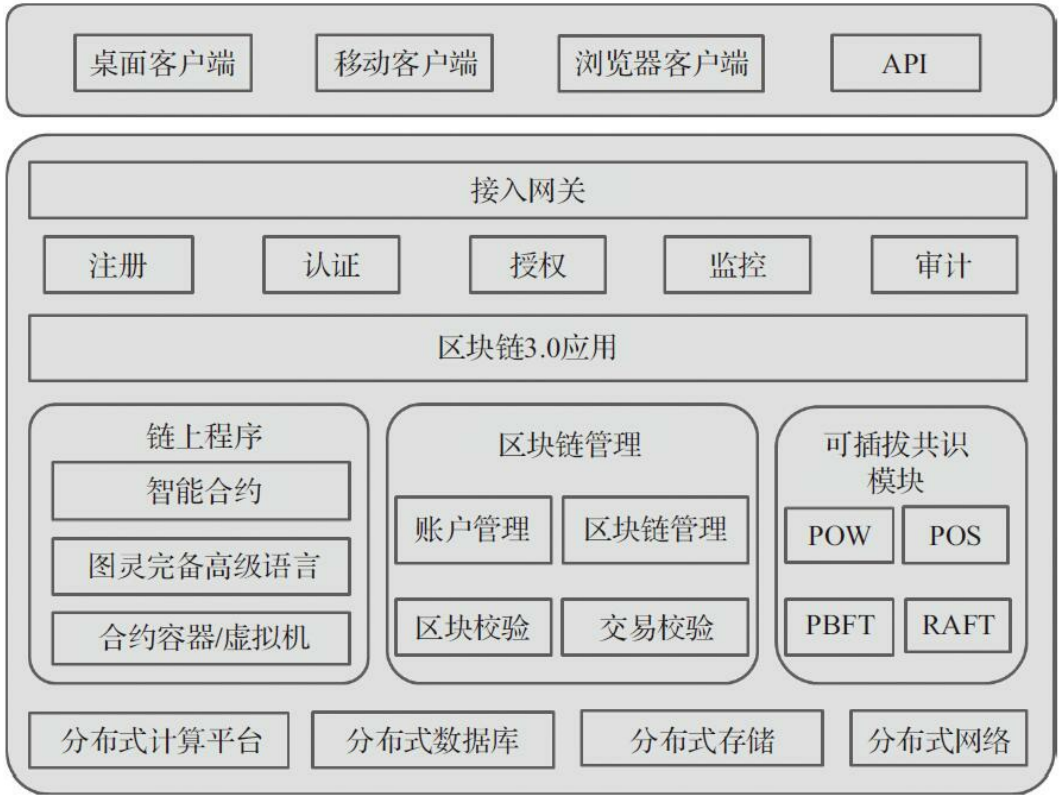

图3.11. 区块链3.0通用架构示意图

3.2 区块链平台发展历程

3.2.3. 区块链3.0：价值互联网

下面是区块链3.0的一些典型的应用.

- 自动化采购可以通过采用区块链方案，实现多方共同记账，共同监管，实现效率和透明度的提供，和提高抗风险能力.
- 智能化物联网应用采用区块链的方案，可以在一个分布式的物联网建立信用机制，利用区块链的记录来监控、管理智能设备，同时利用智能合约来规范智能设备的行为.
- 供应链自动化管理采用区块链的方案，可以登记每个商品的出处，提供一个共享的全局账本，追踪所有引起状态变化的环境.对生产过程、市场渠道的管理，以及政府监管都会有所帮助.
- 虚拟资产兑换、转移采用区块链的方案可以实现虚拟资产的公开、公正的转移，不受第三方影响，自动到账.

3.2 区块链平台发展历程

3.2.3. 区块链3.0：价值互联网

- 产权登记：包括不动产、动产、知识产权、物权、租赁使用权益、商标、执照、许可、各类票据、证书、证明、身份、名称登记等在内的产权登记，都可以采用区块链技术来登记，以保证公正、防伪、不可篡改，以及可审计等.

区块链将作为下一代互联网的重要组成部分，解决目前互联网存在的建立信用、维护信用成本高的问题，并将互联网从现在的信息互联网提升到价值互联网.

目 录

3.3 Hyperledger Fabric简介

3.3.1 Hyperledger Fabric项目介绍

随着比特币，以太坊和其他一些衍生技术的普及，区块链，分布式账本和分布式技术在企业用例的需求也在增长.但是，企业案例还需要一些特殊的性能特征，而这些性能特征是目前公有区块链技术不能提供的.

Hyperledger Fabric是一个开源的企业级许可分布式分类帐技术(DLT)平台，专为在企业环境中使用而设计.

Fabric具有高度模块化和可配置的架构，可为各种行业用例提供支持， 例如银行，金融，保险，医疗保健，人力资源，供应链甚至数字音乐交付，Fabric的应用在未来拥有广阔的前景.

3.3 Hyperledger Fabric简介

3.3.1 Hyperledger Fabric项目介绍

Hyperledger项目是一个大型的开源项目，希望通过各方合作，共同促进和推进区块链 技术在商业应用方面的发展.在组成结构上，包含了很多相关的具体子项目.项目官方地址托管在Linux 基金会网站，代码托管在 Gerrit上，并通过GitHub提供代码镜像.

Hyperledger项目在管理所属子项目时采用了一种生命周期的形式，赋予每个项目一个生命周期，方便项目的运行和管理.整个生命周期分为5个阶段，分别是**提案(proposal)阶段、孵化(incubation)阶段、活跃(active)阶段、弃用(deprecated)阶段以及最后终止(End of Life)阶段**.每个项目在开发运行过程中，一个时间点只会对应着一个阶段.当然，项目不一定会按照以上阶段顺序发展，项目可能会一直处于某个阶段，也可能会因为一些特殊原因在多个阶段之间进行变换.

目前，Hyperledger项目下共有**14个子项目**在运行中，详细信息如下表所示.

3.3 Hyperledger Fabric简介

3.3.1 Hyperledger Fabric项目介绍

项 目 名	状 态	依 赖	描 述
Hyperledger Aries	孵化	Fabric	区块链点对点互动的基础设施
Hyperledger Burrow	孵化		许可的以太坊智能合约区块链
Hyperledger Caliper	孵化		区块链基准框架
Hyperledger Cello	孵化	Fabric，（Sawtooth, Iroha）	区块链管理/运营
Hyperledger Composer	孵化	Fabric，（Sawtooth, Iroha）	用于构建区块链业务网络的开发框架/工具
Hyperledger Explorer	孵化	Fabric，（Sawtooth, Iroha）	区块链Web UI
Hyperledger Fabric	活跃		Golang中的分布式账本
Hyperledger Grid	孵化	Fabric	用于构建包含分布式账本组件的供应链解决方案的平台。
Hyperledger Indy	活跃		用于分散身份的分布式账本
Hyperledger Iroha	活跃		C++中的分布式账本
Hyperledger Quilt	孵化		区块链，DLT和其他类型账本的互操作性解决方案
Hyperledger Sawtooth	活跃		具有多语言支持的分布式账本
Hyperledger Transact Home	孵化		事务执行平台
Hyperledger Ursa	孵化		一个共享的加密库

3.3 Hyperledger Fabric简介

3.3.1 Hyperledger Fabric项目介绍

我们重点来关注Fabric，**Fabric是一种区块链技术的**的分布式共享账本技术.比起其他的区块链技术实现，它组件的开发与使用.其总帐上的数据，由多方参与节点易信息永远**无法被篡改**，并支持通过时间戳进行**溯源查**

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 Linux或Windows 机器上，也可以实现虚拟化.容器是完全使用沙箱机制，相互之间不会有任何接口.

- Fabric引入了**成员管理服务**，因此每个参与者在进入前均需要**允**许访问系统，同时引入**多通道多账本**的设计来增强安全性和私密性.
- Fabric采用了强大的**Docker容器技术**来运行服务，支持比以太坊更便捷、更强大的智能合约服务.
- Fabric可以支持多语言的合约编写，例如GO、Java和Node.js.

3.3 Hyperledger Fabric简介

3.3.1 Hyperledger Fabric项目介绍

Hyperledger Fabric是分布式账本技术(DLT)的独特实现，它可在模块化的区块链架构基础上提供企业级的安全性、可扩展性以及高性能.当前Fabric的最新版本与最早的v0.6版本相比，在安全、保密、部署、维护、实际业务场景需求等方面都进行了很多改进：

- 架构设计上的Peer节点的功能分离、多通道的隐私隔离、共识的可插拔实现等
- 功能上引入Raft崩溃容错共识服务、改进可维护性和可操作性、加入私有数据支持等

Fabric具有以下特性：

- **身份管理(Identity management)**：Fabric区块链是一个许可链网络，因此Fabric提供了一个成员服务(Member Service)，用于管理用户ID并对网络上所有的参与者进行认证.在 Hyperledger Fabric区块链网络中，成员之间可以通过身份信息互相识别，但是他们并不知道彼 此在做什么，这就是Fabric提供的机密性和隐私性.

3.3 Hyperledger Fabric简介

3.3.1 Hyperledger Fabric项目介绍

- **隐私和保密(Privacy and confidentiality)**：Hyperledger Fabric允许竞争的商业组织机构和其他任意对交易信息有隐私和机密需求的团体在相同的许可链网络中共存.其**通过通道来限制消息的传播路径**，为网络成员提供了交易的隐私性和机密性保护.在通道中的所有数据，包括交易、成员以及通道信息都是不可见的，并且对于未订阅该通道的网络实体都是无法访问的.

- **高效的性能(Efficient processing)**：Hyperledger Fabric按照节点类型分配网络角色.为了提供更好的网络并发性和并行性，**Fabric对事务执行、事务排序、事务提交进行了有效的分离**.先于排序之前执行事务可以使得每个Peer节点同时处理多个事务，这种并发执行极大地提高了Peer节点的处理效率，加速了交易到共识服务的交付过程.

3.3 Hyperledger Fabric简介

3.3.1 Hyperledger Fabric项目介绍

- **模块化设计(Modular design)**：Hyperledger Fabric实现的模块化架构可以为网络设计者提供功能选择.例如，特定的身份识别、共识和加密算法可以作为可插拔组件插入Fabric中，因此任何行业都可以采用通用的区块链架构，并确保其网络可跨市场、监管和地理边界进行互操作.

- **可维护性和可操作性(Serviceability and operations)**：日志记录的改进以及健康检查机制以和运营指标的加入，使得v1.4版本在可维护行和可操作性上实现了巨大飞跃.新的RESTful运营服务为生产运营商**提供三种服务来监控和管理对等节点和共识服务节点运营**.
  - 第一种服务使用日志记录/logspec端点，允许操作员动态获取和设置对等节点和共识服务节点的日志记录级别；
  - 第二种服务使用健康检查/healthz端点，允许运营商和业务流程容器检查对等节点和共识服务节点的活跃度和健康情况；
  - 第三种服务使用运营指标/metrics端点，允许运营商利用Prometheus记录来自对等节点和共识服务节点的运用指标.

3.3 Hyperledger Fabric简介

3.3.2 Hyperledger Fabric架构解读

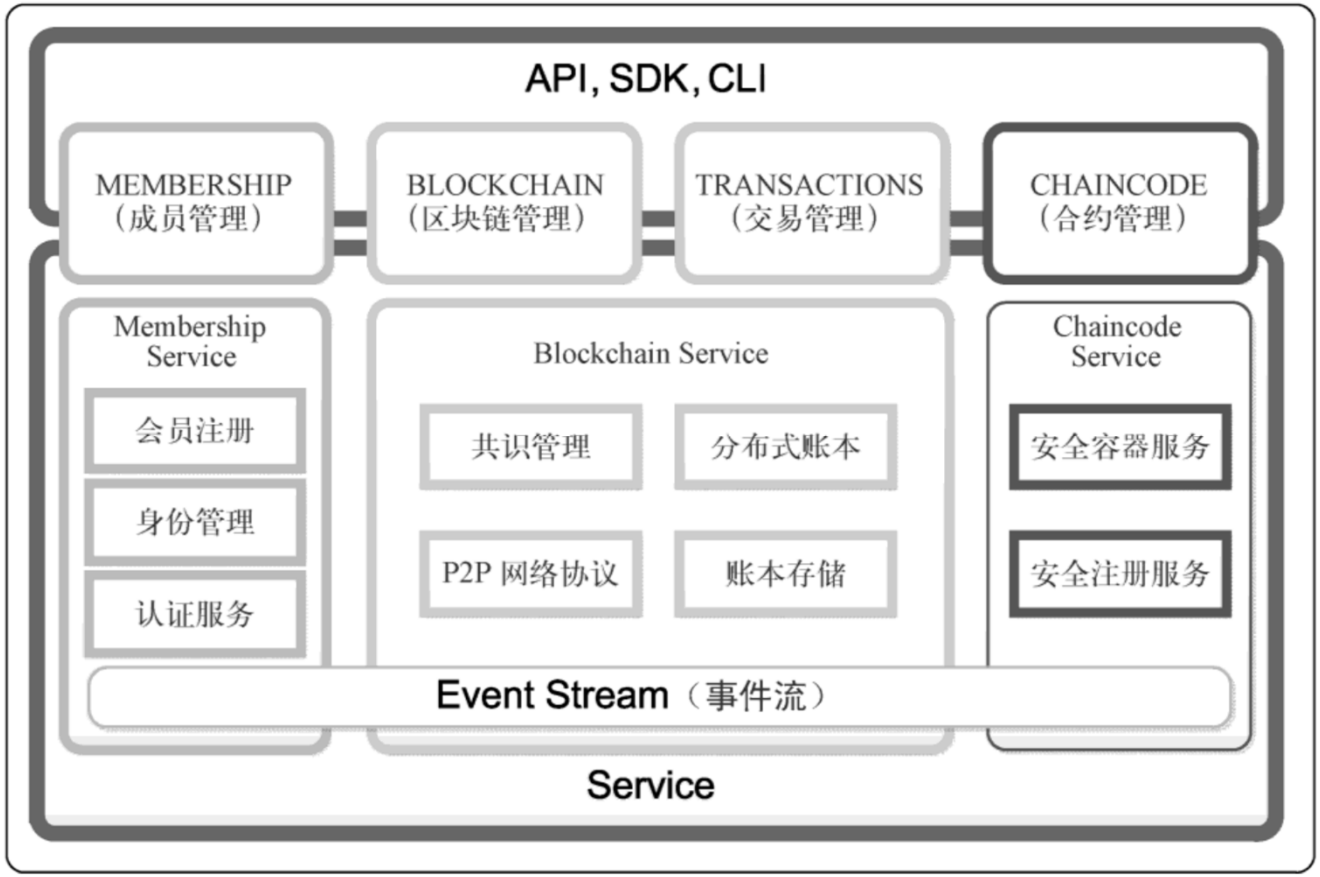

图3.12. Fabric架构示意图

Fabric主要由3个服务模块部组成，分别是**成员服务**(Membership Service)、**区块链服务**(Blockchain Service) 和**合约服务**(Chaincode Service). 在逻辑架构图中，还能看到事件流贯穿三大服务组件间，它的功能是为各个组件的异步通信提供技术支持.

3.3 Hyperledger Fabric简介

3.3.2 Hyperledger Fabric架构解读

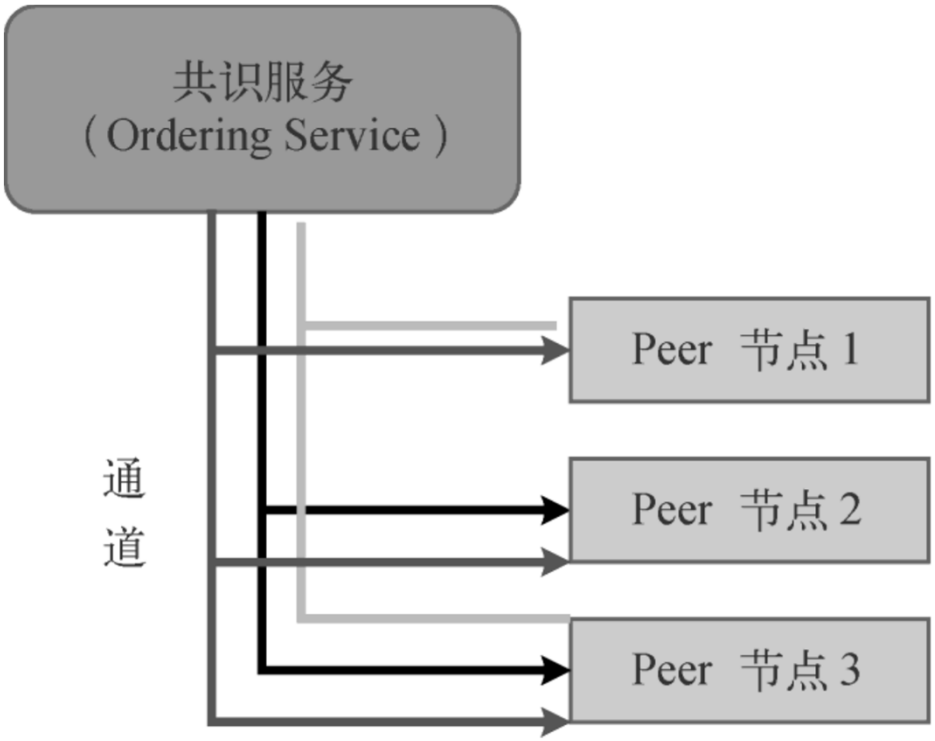

图3.13. Fabric通道示意图

通道:在v1.0之后的版本中，Fabric引入了新的通道概念，共识服务上的消息传递支持多通道，使得Peer节点可以基于应用访问控制策略来订阅任意数量的通道.

- **Peer节点的子集可以被应用程序指定架设相关通道**，指定相同通道的Peer节点组成集合提交该通道的交易，而且只有这些Peer节点可以接收相关交易区块，与其他交易完全隔离.
- **Fabric支持多链与多通道**，即系统中可以存在多个通道以及多条链.应用根据业务逻辑决定将每个交易发送到指定的一个或多个通道，不同通道上的交易不会存在任何联系.

3.3 Hyperledger Fabric简介

3.3.2 Hyperledger Fabric架构解读

私有数据支持：从v1.2开始，Fabric能够在账本中创建私有数据集，允许通道上组织的子集能够认可、提交或查询私有数据，不用创建单独的通道就能实现通道上的一组组织的数据向其他组织保密的功能.

- 私有数据存储在授权组织的对等节点上的私有状态数据库中(有时候被称为“side”数据库)，能被授权节点上的链码通过gossip协议访问.共识服务不涉及私有数据，也无法看到它们.
- 私有数据的哈希值能够被认可、排序并写入通道上每个节点的帐本中，可作为交易的证据，用于状态验证，还可用于审计.

3.3 Hyperledger Fabric简介

3.3.2 Hyperledger Fabric架构解读

节点分类：Fabric中与多种类型的节点，具体分类和功能介绍如下：

- 客户端节点：客户端是最终用户操作的实体，它必须连接到某个peer节点或者orderer节点，与整个区块链网络进行通信.
- CA节点：CA节点接收客户端的注册申请，返回注册密码用于登录，以便获取身份证书.在区块链网络上所有的操作都会验证用户的身份.
- PEER节点：每个Peer节点可以担任如下多种角色： Endorser peer (背书节点)，Leader Peer(主节点)，Committer Peer(记账节点)，Anchor Peer(锚节点).

注意：每个Peer节点必定是一个记账节点，除记账节点外，它还可以担任其它一到多种角色，即某个节点可以同时是记账节点和背书节点，也可以同时是记账节点、背书节点、主节点，锚节点.

3.3 Hyperledger Fabric简介

3.3.2 Hyperledger Fabric架构解读

- **Endorser Peer (背书节点)**：所谓背书(Endorsement)，就是指特定Peer执行交易并向生成交易提案(proposal)的客户端应用程序返回YES/NO响应的过程.只有在应用程序向节点发起交易背书请求时才成为背书节点，其他时候是普通的记账节点，只负责验证交易并记账.
- **Leader Peer(主节点)**：主节点负责和Orderer排序服务节点通信，从排序服务节点处获取最新的区块并在组织内部同步.
- **Committer Peer(记账节点)**：负责验证从排序服务节点接收的区块里的交易，然后将块提交(写入/追加)到其通道账本的副本.
- **Anchor Peer(锚节点)**：锚节点主要用来同步同一通道中各组织间的信息.
- **Orderer(排序服务节点)**：排序服务节点接收包含背书签名的交易，对未打包的交易进行排序生成区块，广播给Peer节点.

3.3 Hyperledger Fabric简介

3.3.2 Hyperledger Fabric架构解读

Hyperledger Fabric 典型的交易流程如下图所示：

图3.14. Fabric交易流程示意图

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

1. Fabric模块-成员服务

区块链网络中的每一个参与者(包括客户端应用程序，记账节点、排序服务节点等)，要想参与区块链网络，都必须具有封装在X.509数字证书中的数字身份.这些身份非常重要，因为它们**确定了参与者在区块链网络中对资源的访问权限**.

要使身份可以验证，它必须来自可信任的权威机构.成员服务提供商(MSP)在Fabric中就充当权威机构的角色，默认使用**X.509证书作为身份**，采用传统的公钥基础结构(PKI)分层模型.

大家知道X.509证书吗？用过X.509证书吗？

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

PKI(Public Key Infrastructure，公钥基础设施)的目标就是实现不同成员在不见面的情况下进行安全通信(建立信任)，Fabric采用的模型是基于可信的第三方机构，也就是证书颁发机构(Certification Authority，CA)签发的证书.CA会在确认申请者的身份后签发证书，同时会在线提供其所签发证书的最新吊销信息，这样使用者就可以验证证书是否仍然有效.

证书是一个包含公钥、申请者相关信息以及数字签名的文件.数字签名保证了证书中的内容不能被任何攻击者篡改，而且验证算法可以发现任何伪造的数字签名.通常情况下，PKI体系包含证书颁布机构(CA)、注册机构(RA)、证书数据库和证书存储实体.

- RA是一个信任实体，它负责对用户进行身份验证以及对数据、证书或者其他用于支持用户请求的材料进行合法性审查；
- CA则会根据RA的建议，给指定用户颁发数字证书，这些证书由根CA直接或分层进行认证.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

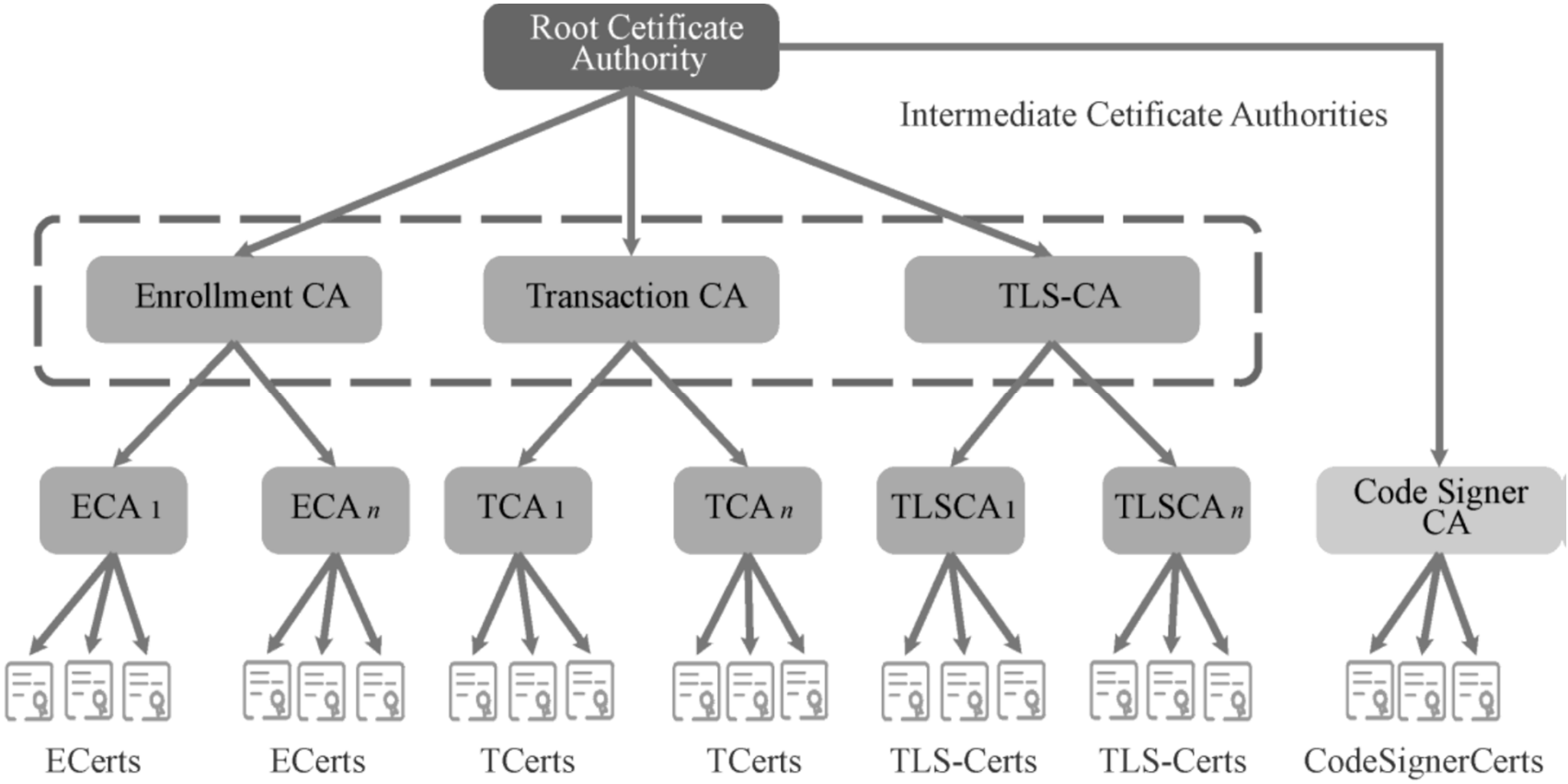

图3.15. Fabric成员服务实体示意图

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

对上图中的实体进行进一步介绍说明：

- **Root Certificate Authority(Root CA)**：根CA，代表PKI体系中信任的实体，同时也是PKI体 系结构中的最顶层认证机构.
- **Registration Authority(RA)**：注册机构，是一个可信任的实体，通过它可以确定希望参与 许可区块链的用户的有效性和身份信息.它通过与用户进行带外通信的方式以验证用户的身份和角色.同时RA还需要负责创建注册所需的注册凭证.
- **Enrollment Certificate Authority(ECA)**：在验证用户提供的注册凭证后，ECA负责发出注册证书(ECerts).
- **Transaction Certificate Authority(TCA)**：在验证用户提供的注册凭证后，TCA负责发出交易证书(TCerts).

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

- **TLS Certificate Authority (TLS-CA)**：负责颁发TLS(Transport Layer Security，传输层安全协议)证书和凭据，以允许用户使用其网路 .
- **Enrollment Certificates(ECerts)**：ECerts是长期证书，针对所有角色颁发.
- **Transaction Certificates(TCerts)**：TCerts是每个交易的短期证书.它们是由TCA根据授权的用户请求颁发的.此外，TCerts可以被配置为不携带用户身份的信息.它们使得用户不仅可以匿名地参与系统，还可以防止事务的可链接性.
- **TLS-Certificates(TLS-Certs)**：TLS-Certs携带其所有者的身份，用于系统和组件之间进行通信及维护网络级安全性.
- **Code Signer Certificates(CodeSignerCerts)**：负责对代码进行数字签名来标识软件来源及软 件开发者的真实身份，以此保证代码在签名之后不被恶意篡改.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

具体的用户注册流程做一个简单的介绍，成员的注册分为两个过程：

➢ 离线过程

(1) 每个用户或者Peer节点必须向RA注册机构提供身份证件(ID证明)，同时这个流程必须通过带外数据(out-of-band，OOB)进行传输，以提供RA为用户创建(和存储)账户所需的证据.

(2) RA注册机构返回用户有关的**用户名和密码，以及信任锚(包含TLS-CA Cert)**.如果用户可以访问本地客户端，那么客户端可以将TLS-CA证书作为信任锚的一种方式.

➢ 在线过程

(3) 用户连接客户端以请求登录系统，在这一过程中，用户将用户名和密码发送给客户端.

(4) 用户端接着代表用户向成员服务发送请求，成员服务接受请求.

(5) 成员服务将包含几个证书的包发送给客户端.

(6) 一旦客户端验证完成所有的加密材料是正确有效的，它就会将证书存储于本地数据库中并通知用户，用户注册完成.

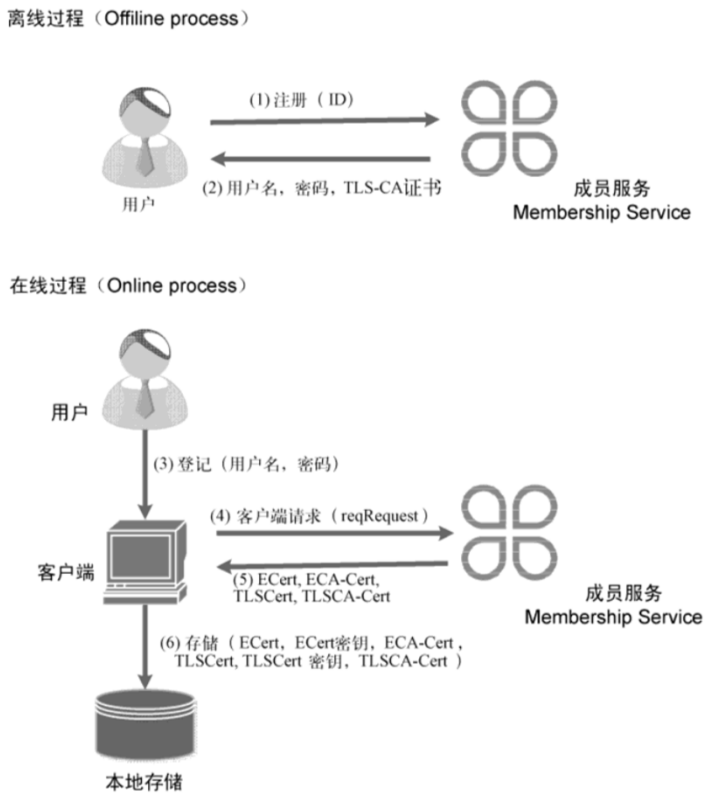

图3.16. 用户注册流程示意图

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

现在来看看这些身份如何用于表示区块链网络的可信成员.这是会员服务提供商(MSP)发挥作用的地方：它通过列出其成员的身份来识别应该信任哪些根CA和中间CA是信任域的成员，或者哪些CA被授权为成员签发有效的身份.

MSP的强大不仅仅是列出谁是网络参与者或频道成员.MSP可以在它所代表的组织范围内识别参与者可能扮演的特殊角色，为网络和通道的访问权限设置奠定基础.在区块链网络中，有两个地方会出现MSP，一个是**本地MSP**，一个是**通道MSP**.

- 本地MSP用来定义节点和用户的权限，定义了本地层面的成员哪些有管理权哪些有参与权.
- 如果一个组织的节点想要加入某个通道，那么本地MSP也要加入通道的配置中，一个**通道中的所有节点共享通道MSP的视图**.下面来看一个例子.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

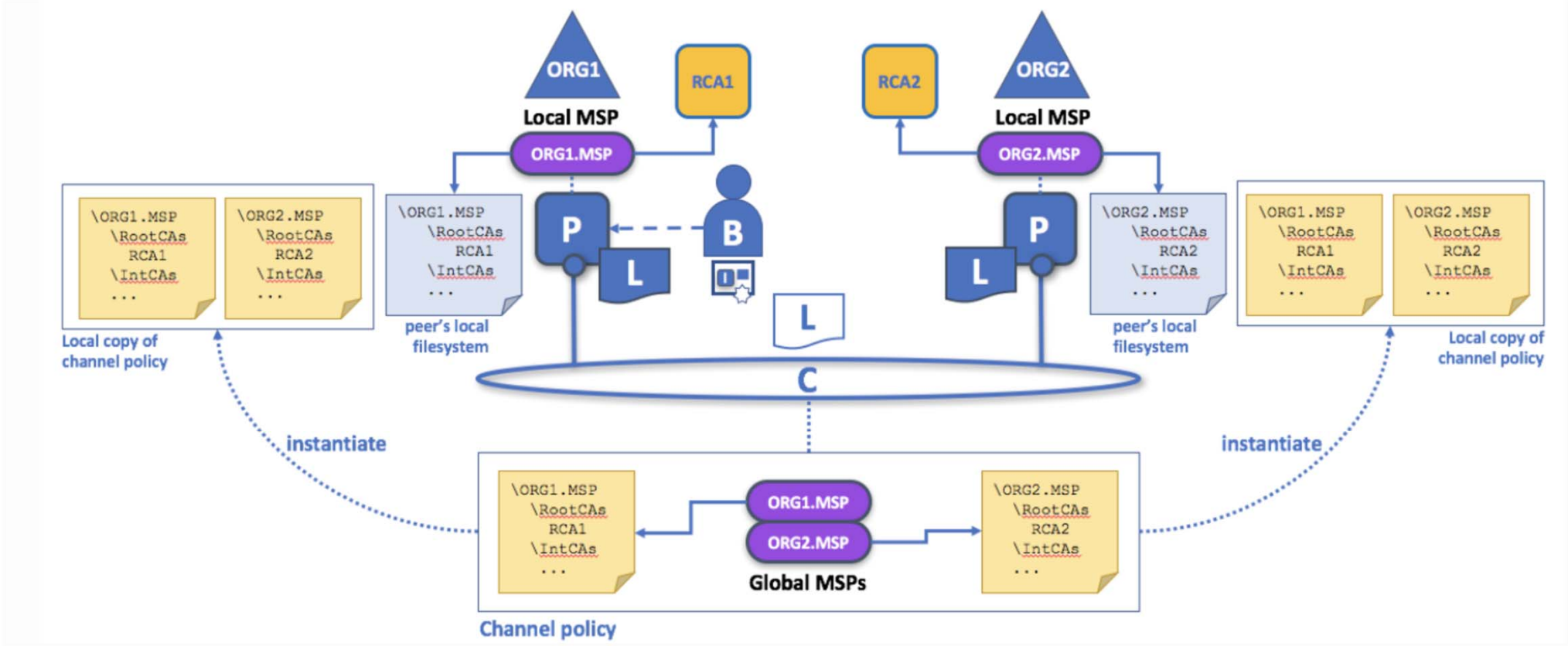

图3.17. MSP工作流程示意图

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

用户**B的身份由RCA1签发**，存储在本地MSP中，当B想要连接到Peer并尝试在peer上安装智能合约时，需要执行一下操作：

- Peer先检查ORG1-MSP，以验证B的身份确实是ORG1的成员．验证成功后将允许install命令实现链码的安装.
- B希望在通道上也实例化智能合约，那么通道上的所有组织必须都同意.因此，Peer必须先检查通道的MSP，然后才能成功提交此命令.

本地MSP仅在应用的节点或用户的文件系统上定义.因此，在物理上和逻辑上，每个节点或用户只有一个本地MSP.

由于通道MSP可用于通道中的所有节点，因此它们在通道配置中进行逻辑定义一次.其实，**通道MSP也在通道中为每个节点的文件系统实例化**，并通过一致性保持同步.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

2. Fabric模块-区块链服务

Fabric的区块链服务包含4个模块：共识管理、分布式账本、账本存储以及P2P网络协议.

- **共识管理**用于在多个节点的分布式复杂网络中使消息达成共识.
- **分布式账本与账本存储**负责管理区块链系统中所有的数据存储，比如交易信息、世界状态等.
- **P2P网络协议**则是网络中节点的通信方式，负责Fabric中各节点间的通信与交互.

(1)P2P网络

在Fabric的网络环境中，节点是区块链的通信实体.存在三类不同的节点，分别是客户端节 点(Client)、Peer节点(Peer)以及共识服务节点(Ordering Service Nodes或者Orderers).

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

客户端节点代表着终端用户实体.

- 必须连接到Peer节点后才可以与区块链进行通信交互.
- 可以根据它自己的选择来连接到任意的Peer节点上，创建交易和调用交易.
- 在实际系统运行环境中，客户端负责与Peer节点通信提交实际交易调用，与共识服务通信请求广播交易的任务.

Peer节点负责与共识服务节点通信来进行世界状态的维护和更新.

- 它们会收到共识服务广播的消息，以区块的形式接收排序好的交易信息，然后更新和维护本地的世界状态与账本.
- Peer节点可以额外地担当背书节点的角色，负责为交易背书.每个合约代码程序都可以指定一个包含多个背书节点集合的背书策略.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

(2)共识服务

Fabric网络中的Orderers节点聚集在一起形成了共识服务.

- 它可以看作一个提供交付保证的通信组织.
- 共识服务为客户端和Peer节点提供了一个共享的通信通道，还为包含交易的消息提供了一个广播服务的功能.

共识服务可以有不同的实现方式，在v1.0版本中，Fabric将共识服务设计成了可插拔模块，可以根据不同的应用场景配置不同的共识选项.目前，Fabric提供了3种模式实现：**Solo、Kafka和Ratf**.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

- Solo是一种部署在单个节点上的简单时序服务，它只支持单链和单通道.
- Kafka是一种支持多通道分区的集群共识服务，可以支持CFT(Crash Faluts Tolerance).它容忍部分节点宕机失效，但是不能容忍恶意节点.
- Raft遵循“领导者和追随者”模型，每个通道都选举一个“领导者”，它的决定将被复制给“追随者”.只要总节点数与失效节点数目满足n>=2f+1，它就允许包括领导者在内的部分节点宕机失效.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

(3)分布式账本

区块链技术从其底层构造上分析，可以将其定义为一种共享账本技术.账本是区块链的核心组成部分，在区块链的账本中，存储了所有的历史交易和状态改变记录.

- 在Fabric中，每个通道都对应着一个共享账本，而每个连接在共享账本上的Peer节点，都能参与网络和查看账本信息，即它允许网络中的所有节点参与和查看账本信息.
- 账本上的信息是公开共享的，并且在每个peer节点上，都维持着一份账本的副本.

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

3. Fabric模块-合约代码服务

Fabric合约代码服务提供了一种安全且轻量级的方式，沙箱验证节点上的合约代码执行，提供安全容器服务以及安全的合约代码注册服务.

其运行环境是一个锁定和安全的容器，合约代码首先会被编译成一个独立的应用程序，运行于隔离的Docker容器中.在合约代码部署时，将会自动生成一组带有签名的智能合约的Docker基础镜像.在Docker容器中，Peer节点与合约代码交互过程如图所示.

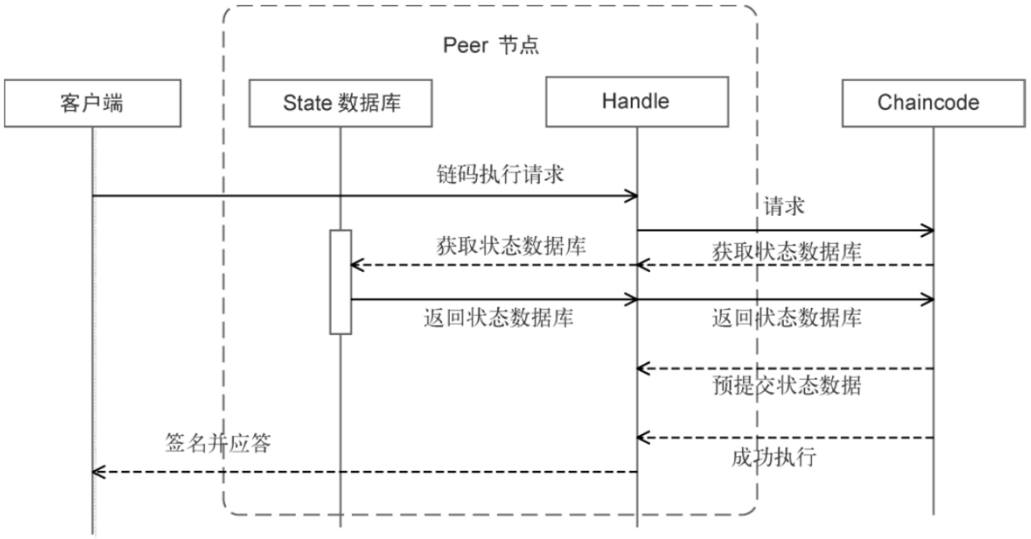

图3.18. 合约代码执行流程示意图

3.3 Hyperledger Fabric简介

3.3.3 Hyperledger Fabric架构组成

步骤如下：

- Peer节点收到客户端发来的合约代码执行请求后，通过gRPC与合约代码交互，发送一个合约代码消息对象给对应的合约代码.
- 合约代码通过调用Invoke()方法，执行getState()操作和putState()操作，向Peer节点获取账本状态数据库和发送账本预提交状态数.若要读取和写入私有数据，则通过getPrivateDate()和putPrivateDate()方法.
- 合约代码执行成功后将输出结果发送给Peer节点，背书节点对输入和输出信息进行背书签名，完成后应答给客户端.

谢谢！