

# Neohope's Blockchain Book

Neohope

Published  
with GitBook



# 目錄

版本控制	1.1
概述	1.2
本书内容	1.2.1
读者定位	1.2.2
本书约定	1.2.3
项目对比	1.2.4
名词释义	1.2.5
基本原理	1.3
区块链	1.3.1
智能合约	1.3.2
交易所	1.3.3
系统架构	1.3.4
基础算法	1.4
默克尔树	1.4.1
Bloom过滤器	1.4.2
公钥私钥及椭圆曲线	1.4.3
共识算法	1.5
分布式系统	1.5.1
Paxos与Raft算法	1.5.2
拜占庭容错算法	1.5.3
Bitcoin	1.6
整体架构	1.6.1
状态转换	1.6.2
区块确认	1.6.3
分叉处理	1.6.4
P2P	1.6.5
脚本	1.6.6
钱包	1.6.7
使用	1.6.8
编译(Ubuntu)	1.6.9
私链搭建(Linux)	1.6.10
私链搭建(Docker)	1.6.11
Ethereum	1.7
整体架构	1.7.1
状态转换	1.7.2
区块确认	1.7.3
使用	1.7.4
Remix IDE环境搭建(Ubuntu16)	1.7.5
智能合约编写01	1.7.6
智能合约编写02	1.7.7

智能合约编写03	1.7.8
智能合约编写04	1.7.9
编译(MacOS)	1.7.10
私链搭建	1.7.11
 EOS	
整体架构	1.8.1
使用	1.8.2
智能合约使用(单机单节点单钱包)	1.8.3
智能合约使用(单机单节点多钱包)	1.8.4
编写简单的智能合约	1.8.5
编写有数据存储的智能合约	1.8.6
编译(Linux)	1.8.7
私链搭建(单机多节点)	1.8.8
私链搭建(多机多节点)	1.8.9
 Fabric	1.9
整体架构	1.9.1
身份管理	1.9.2
共识达成	1.9.3
使用	1.9.4
智能合约介绍	1.9.5
智能合约使用01	1.9.6
智能合约使用02	1.9.7
编译(MacOS)	1.9.8
私链搭建(Ubuntu)	1.9.9
在私链中添加一个机构	1.9.10
 Cosmos	1.10
整体架构	1.10.1
Zone与Hub	1.10.2
使用	1.10.3
 IPFS	1.11
整体架构	1.11.1
使用	1.11.2
编译(MacOS)	1.11.3
私链搭建	1.11.4
然后呢？	1.12

## 版本控制

### NEOHOPE

版本号	修改内容	修改人	修改时间
0.1.0	初步整理了Bitcoin、EOS、IPFS的信息	H	2018-04-15
0.2.0	完善了Bitcoin、EOS、IPFS的信息	H	2018-04-21
0.3.0	新增了Ethereum的信息	H	2018-04-22
0.4.0	整理了Fabric的信息	H	2018-05-21
0.5.0	完善了Bitcoin、Ethereum的部分信息	H	2018-06-02
0.6.0	初步整理了Cosmos的信息	H	2018-06-03
0.7.0	调整了图片命名规则, 补充了部分内容	H	2018-06-10

## 概述

本书整合了[几大流行区块链的技术文档资料](#), 包括[区块链的使用](#)、[智能合约的编写及使用](#)、[私链搭建](#)、[源码的编译](#)等内容。

本书绝大多数资料来自于各区块链的白皮书、技术文档等, 我们对相关内容做了整理和分类。

我们对书中的源码并没有所有权, 仅作为参考资料。

# 本书内容

本书是NEOHOPE整理的参考资料，包括了主流区块链项目的一些入门信息。

## 本书包括的内容

- 主流区块链项目的介绍、使用、合约编写、私链搭建
- 包括：bitcoin、ethereum、eos、fabric、ipfs

## 本书不包括的内容

- 区块链基础知识：概念、来源、中本聪、野史，因为这样的书有很多，而且内容重复率很高
- 炒币任何知识
- 挖矿任何知识
- 系统操作基础知识
- 编程基础知识

## 读者定位

### 适于以下读者

- 对区块链有一定了解，并喜欢Code First的开发的人员
- 希望了解智能合约，并喜欢Code First的开发的人员
- 希望搭建私链，进一步搞清原理的开发人员

### 本书需要的前置内容

- Linux操作基础知识
- 编程基础知识
- 区块链基础知识

### 不适用于以下读者

- 希望找一本入门书的读者
- 希望炒币发财的读者
- 技术速成的读者

## 本书约定

### 须知

- 请区分公链、测试链、私链的账号信息，账号不要一致，存储文件夹不要一致
- 请保管好自己的账号、密码、私钥
- 书中内容有问题，请改善而不要仅仅是抱怨，欢迎贡献内容

# 项目对比

## 常见区块链

项目名称	链类型	匿名性	共识机制	开发语言	合约语言
Bitcoin	公链	匿名	PoW	CPP	只是使用合约实现了业务逻辑
Ethereum	公链/ 联盟链	匿名 或私有	PoW	Go/CPP/Python	Solidity/Serpent/LLL
Fabric	联盟链	共有 或认 证	No-op/ PBFT(classic, batch, sieve)	Go/Java	Chaincode
Sawtooth Lake	联盟链	共有 或认 证	PoET/Quorum Voting	Python/CPP/Rust	Python/Cpp/Go/Java/JS/Rust
EOS	公链	匿名	BFT-DPOS	CPP	CPP/Web Assembly
IPFS	公链	匿名	PoSt	Go	-
Cosmos	公链/ 生态	匿名	PoS/Tendermint BFT	Go	-

PoW: Proof of Work, 工作量证明

PoS: Proof of Stack, 权益证明

DPoS: Delegated Proof of Stake, 委托式权益证明

PoET: Proof of Elapsed Time, 时间消逝证明(特殊硬件依赖)

PoSt: Proof of Storage(Proof-of-Replication + Proof-of-Spacetime), 存储证明

BFT: Byzantine Fault Tolerance, 拜占庭容错

PBFT: Practical Byzantine Fault Tolerance, 实用拜占庭容错

# 名词释义

## 区块链相关

### 区块

Block(区块)：是一种特殊的数据结构，一个区块就是若干交易数据的集合，它会被标记上时间戳和之前一个区块的独特标记。区块头经过哈希运算后会生成一份工作量证明或权益证明，从而验证区块中的交易。有效的区块经过全网络的共识后会被追加到主区块链中。

### 创世区块

Genesis Block(创世区块)，区块链的第一个区块，一般用于初始化，不带有交易信息。

### 区块链：

Blockchain(区块链)，由多个区块链接而成的链表结构，除了创世区块，每个区块头部都包括前继区块内容的Hash值，也就是说链上每个区块都是可以验证的，链上的区块顺序也是可以验证的。

### 地址

区块链地址其实就是一串字符和数字组成的字符串。就像别人向你的email地址发送电子邮件一样，他可以通过你的地址向你发送代币。

### 分叉：

在节点验证区块链发生冲突时，会发生分叉(forking)。也就是说在某一区块之后，网络中有多于一个区块链，且每个区块链由一些矿工验证。分叉共有三种：普通分叉、软分叉和硬分叉。

### 普通分叉：

是由于两个或者多个矿工几乎同时发现了一个区块引起的暂时冲突。网络将选择难度值最高的分叉。比如，在挖矿过程中，已经确认生产11800区块，但两个矿工同时生成了11801区块，矿工A在该块下又生成了1个区块；支持矿工B的节点有在该块下生成了2个区块；一般来说会支持矿工B所在的分叉，并最终被整个网络接受。

### 软分叉：

更改区块链源代码后，会引起冲突，如果必须要求有50%以上算力的矿工升级以解决冲突，叫作软分叉；比如，如果更新源代码使旧区块/交易的一部分失效，则有50%以上算力的矿工升级后可以解决，这样新的区块链将有更大难度值，最后被整个网络接受。

### 硬分叉：

更改区块链源代码后，会引起冲突，如果必须要求所有矿工升级以解决冲突，叫作硬分叉。硬分叉的一个例子是，如果更新源代码是为了更改对矿工的回报，则全部矿工需要升级以解决冲突。以太坊自发布以来经历了多次硬分叉和软分叉。

### 代币：

通过密码学技术，在区块链上发行的电子代币。用于交易、投票、支付矿工费用、虚拟奖励等各种用途。

### 钱包

钱包指保存代币地址和私钥的软件，可以用它来接收、发送、储存你的代币。

## 交易：

简单地说，交易指把代币从一个地址转到另一个地址。更准确地说，一笔“交易”指一个经过签名运算的，表达价值转移的数据结构。每一笔“交易”都经过区块链网络传输，由矿工节点收集并封装到区块中，永久保存在区块链上。

## 交易所：

就是法币与代币兑换、代币与代币兑换的场所。

## 智能合约：

即以计算机程序的方式来缔结和运行各种合约。运行在区块链上的提前约定的合同；

## 燃料

燃料（Gas），控制某次交易执行指令的上限。每执行一条合约指令会消耗固定的燃料。当某个交易还未执行结束，而燃料消耗完时，合约执行终止并回滚状态。

## Mining(挖矿)

通过不断的暴力尝试来找到一个字符串，使得它加上一组交易信息后的Hash值符合特定规则（例如前缀包括若干个0），找到的人可以宣称新区块被发现，并获得系统奖励的数字货币。

## Miner(矿工)

参与挖矿的人或组织。

## Mining Machine(矿机)

专门为数字货币挖矿而设计的设备，包括基于CPU挖矿、GPU挖矿、FPGA挖矿、专用芯片挖矿等多种实现。

## Mining Pool(矿池)

采用团队协作方式来集中算力进行挖矿，对产出的数字货币进行分配。

## 挖矿难度

就是基于PoW机制的链，在矿工挖矿时，需要每过一段时间调整计算难度（例如增加前缀中0的个数），来应对日益增强的算力。

## 加解密相关

### Hash

任意长度的二进制值（明文）映射为较短的固定长度的二进制值（Hash值），并且不同的明文很难映射为相同的Hash值。

目前流行的Hash算法包括MD5、SHA-1和SHA-2。

### 数字摘要

顾名思义，数字摘要是对数字内容进行Hash运算，获取唯一的摘要值来指代原始数字内容。数字摘要是解决确保内容没被篡改过的问题（利用Hash函数的抗碰撞性特点）。

数字摘要是 Hash 算法最重要的一个用途。在网络上下载软件或文件时，往往同时会提供一个数字摘要值，用户下载下来原始文件可以自行进行计算，并同提供的摘要值进行比对，以确保内容没有被修改过。

## 对称加密

加解密的密钥是相同的，代表算法包括 DES、3DES、AES、IDEA 等。代表算法包括：RSA、ElGamal、椭圆曲线（ECC）系列算法。

## 非对称加密

加密密钥和解密密钥是不同的，分别称为公钥和私钥。

## 数字签名

数字签名是一种以电子形式存在于数据信息之中的，或作为其附件或逻辑上有联系的数据，可用于辨别数据签署人的身份，并表明签署人对数据信息中包含的信息的认可技术。数字签名技术是将摘要信息用发送者的私钥加密，与原文一起传送给接收者。接收者只有用发送者的公钥才能解密被加密的摘要信息，然后用HASH函数对收到的原文产生一个摘要信息，与解密的摘要信息对比。如果相同，则说明收到的信息是完整的，在传输过程中没有被修改，否则说明信息被修改过，因此数字签名能够验证信息的完整性。同时数字签名用到了发送者的私钥，是很难伪造的，也就解决了不可抵赖的问题。

## PKI

Public Key Infrastructure，基于公钥体系的安全基础架构。

## CA

Certificate Authority，负责证书的创建、颁发，在 PKI 体系中最为核心的角色。

## CRL

Certification Revocation List，证书吊销列表，包含所撤销的证书列表。

## CSR

Certificate Signing Request，证书签名申请，包括通用名、名称、主机、生成私钥算法和大小、CA 配置和序列号等信息，用来发给 CA 服务以颁发签名的证书。

## 共识相关：

### Consensus( 共识 )

分布式系统中多个参与方对某个信息达成一致，多数情况下为对发生事件的顺序达成一致。

### 共识机制：

分布式系统达成共识所采用的算法。

## PoS

Proof of Stake，股份持有证明，拥有代币或股权越多的用户，挖到矿的概率越大。

## PoW

Proof of Work, 工作量证明, 在一定难题前提下求解一个 SHA256 的 Hash 问题。

### **零知识证明( zero knowledge validation)**

证明者在不向验证者提供任何有用的信息的前提下, 使验证者相信某个论断是正确的。例如, A 向 B 证明自己有一个物品, 但 B 无法拿到这个物品, 无法用 A 的证明去向别人证明自己也拥有这个物品。

### **Byzantine Failure( 拜占庭错误)**

指系统中存在除了消息延迟或不可送达的故障以外的错误, 包括消息被篡改、节点不按照协议进行处理等, 潜在地会对系统造成针对性的破坏。

## **安全相关**

### **MTBF**

Mean Time Between Failures, 平均故障间隔时间, 即系统可以无故障运行的预期时间。

### **MTTR**

Mean Time to Repair, 平均修复时间, 即发生故障后, 系统可以恢复到正常运行的预期时间。

### **MVCC**

Multi-Version Concurrency Control, 多版本并发控制。数据库领域的技术, 通过引入版本来实现并发更新请求的乐观处理, 当更新处理时数据版本跟请求中注明版本不一致时则拒绝更新。发生更新成功则将数据的版本加一。

### **SLA/SLI/SLO**

Service Level Agreement/Indicator/Objective, 分别描述服务可用性对用户的承诺, 功能指标和目标值。

### **Sybil Attack(女巫攻击)**

少数节点通过伪造或盗用身份伪装成大量节点, 进而对分布式系统系统进行破坏

### **Double-Spend Attack**

双花攻击/双重支付攻击:指的是攻击者通过伪造凭证、信息不对称、时间延迟等技术手段, 用同一笔钱消费多次的一种攻击方式。

## **分布式**

### **Distributed( 分布式)**

非单体中央节点的实现, 通常由多个个体通过某种组织形式联合在一起, 对外呈现统一的服务形式。

### **Decentralization( 去中心化) :**

无需一个独立第三方的中心机构存在, 有时候也叫多中心化。

### **DLT**

Distributed Ledger Technology, 分布式账本技术。包括区块链、权限管理等在内的实现分布式账本的技术

## **DApp**

去中心化应用( Decentralized App)

## **DAO**

去中心化自治组织( Decentralized Autonomous Organization) , 基于区块链的按照智能合约联系起来的松散自治群体。

## **DAC**

去中心化自治公司( Decentralized Autonomous Company)

## **P2P**

点到点的通信网络, 网络中所有节点地位均等, 不存在中心化的控制机制。

## 计算机专业术语

### **Turing-complete ( 图灵完备 )**

指一个机器或装置能用来模拟图灵机( 现代通用计算机的雏形 ) 的功能, 图灵完备的机器在可计算性上等价。

## **CDN**

Content Delivery Network, 内容分发网络。利用在多个地理位置预先配置的缓存服务器, 自动从距离近的缓存服务器进行对请求的响应, 以实现资源的快速分发。

## **Merkle Tree**

梅克尔树

## **Merkle Patricia Tree**

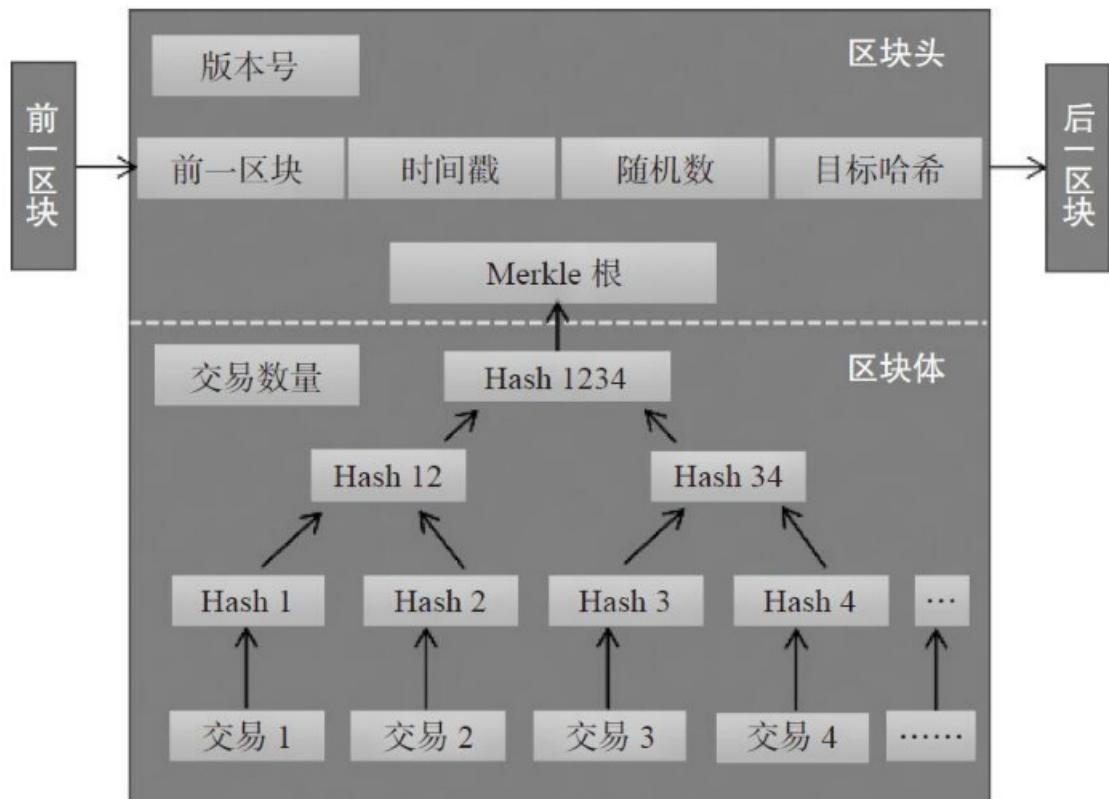
梅克尔帕特里夏树

## 基本原理

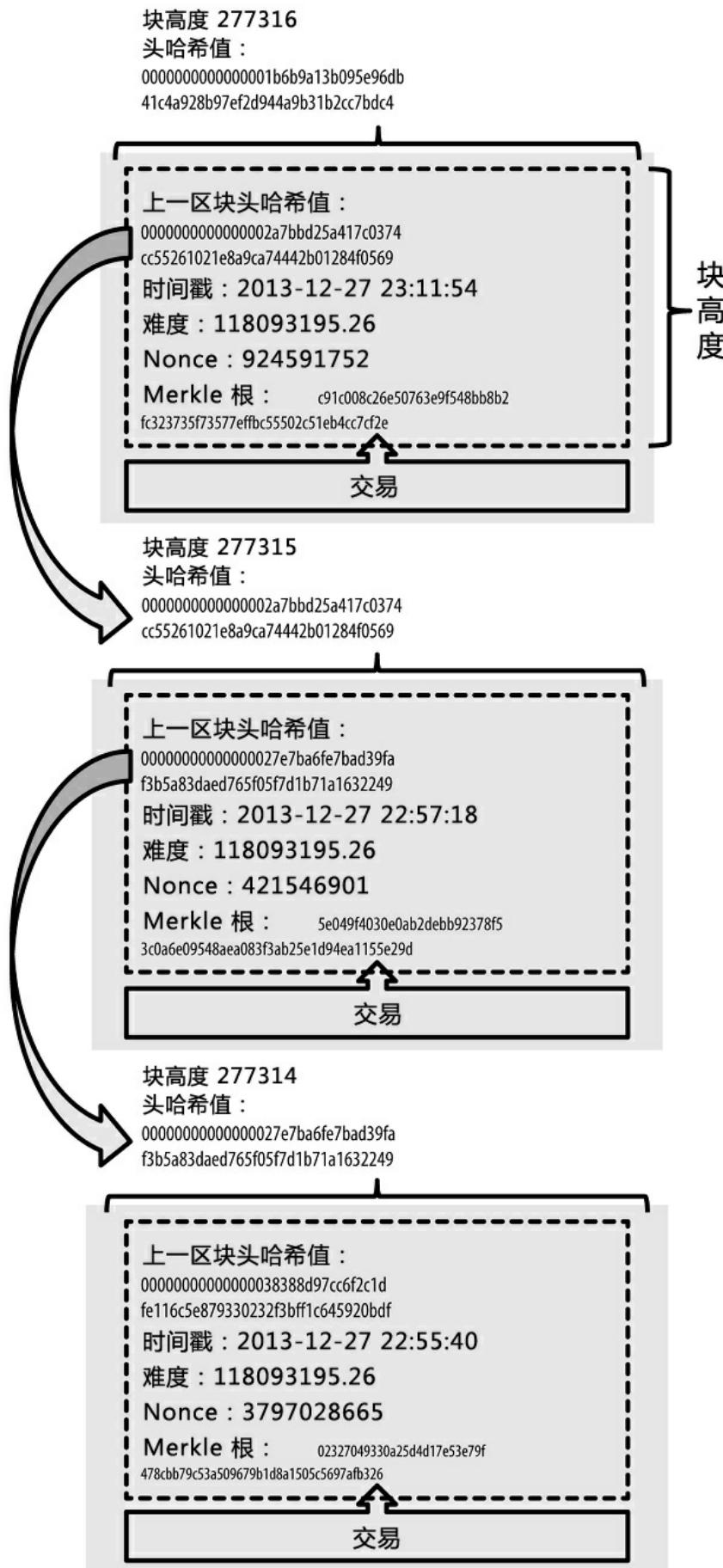
本章主要介绍区块链的基本原理。

# 区块链

## 比特币的区块



## 区块链构造



## 比特币的区块链

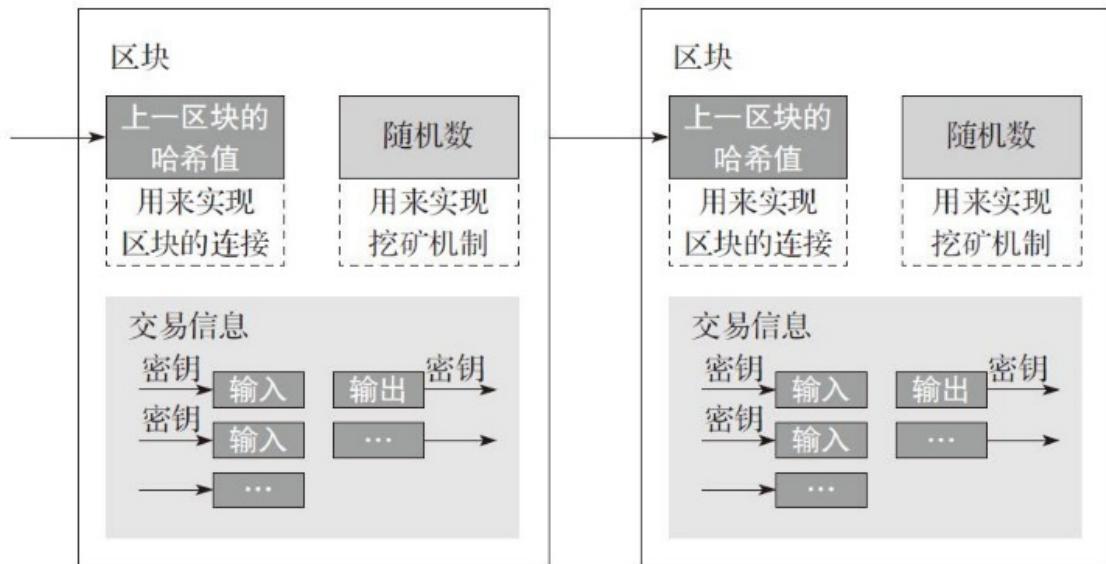


图2-10 区块链的链接模型

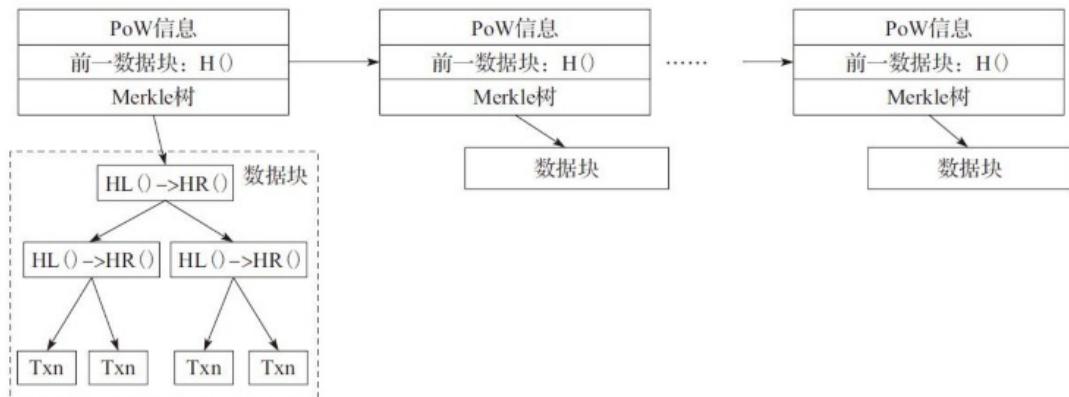


图5-4 区块链的简化结构

## 比特币的区块链验证流程

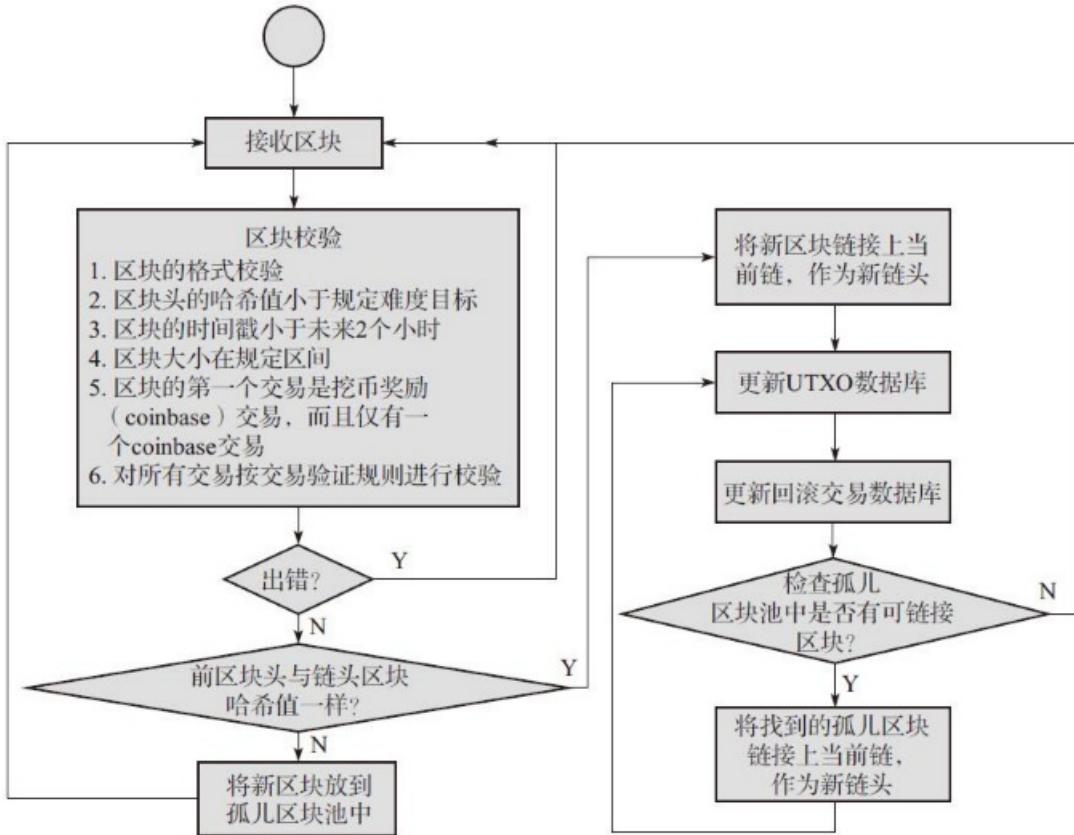


图3-6 区块验证流程图

## 比特币的交易

## 智能合约

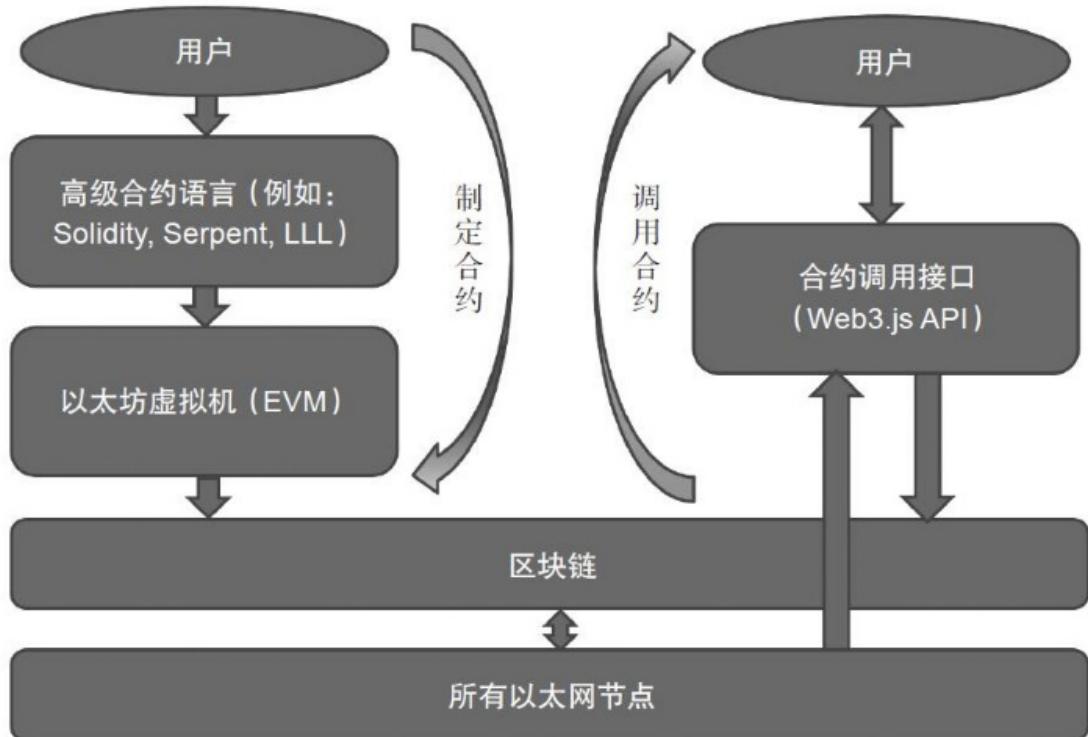


图7-4 以太坊合约的部署和调用

## 交易所

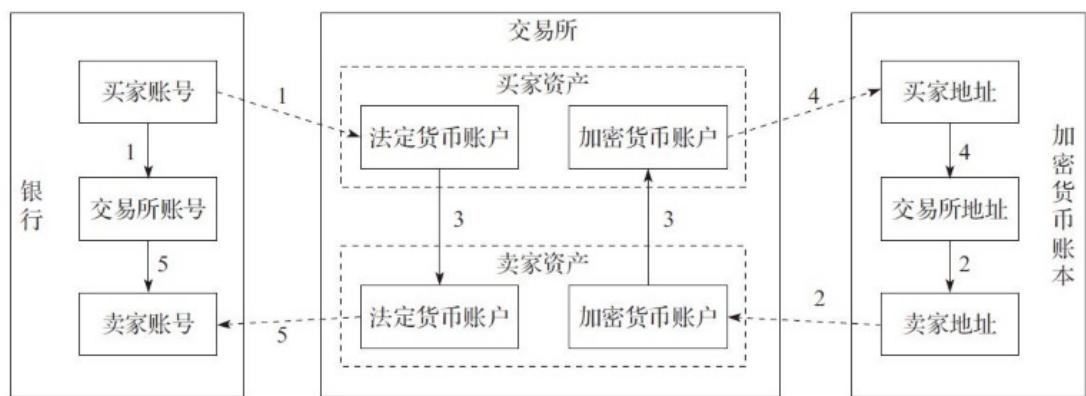


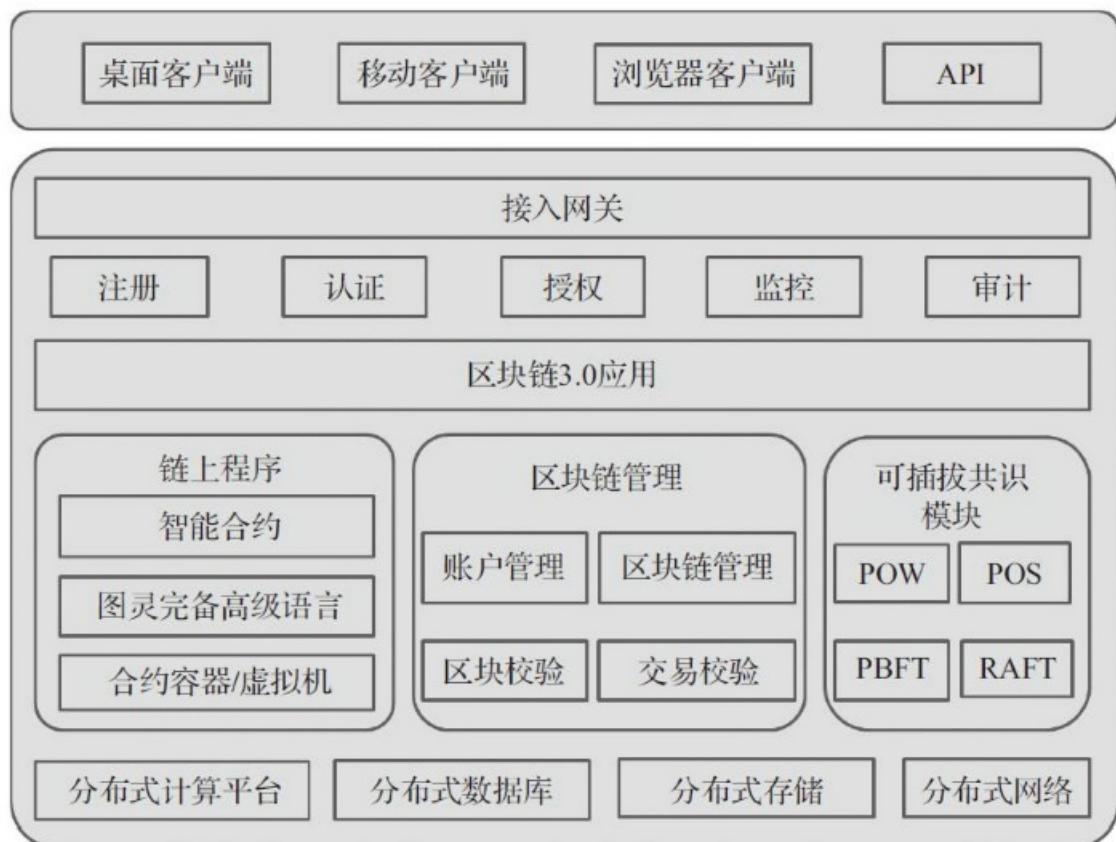
图9-2 加密货币交易流程的简化模型

## 系统架构

### 基础架构



图2-9 区块链基础架构



## 基础算法

# 默克尔树

## Merkle树

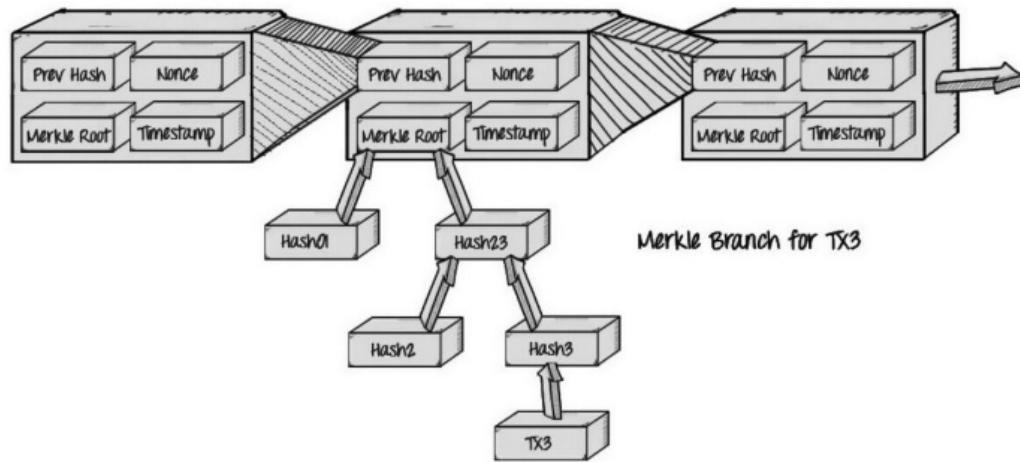


图2-11 比特币的Merkle证明树

默克尔树（又叫哈希树）是一种二叉树，由一个根节点、一组中间节点和一组叶节点组成。最下面的叶节点包含存储数据或其哈希值，每个中间节点是它的两个孩子节点内容的哈希值，根节点也是由它的两个子节点内容的哈希值组成。

进一步的，默克尔树可以推广到多叉树的情形。

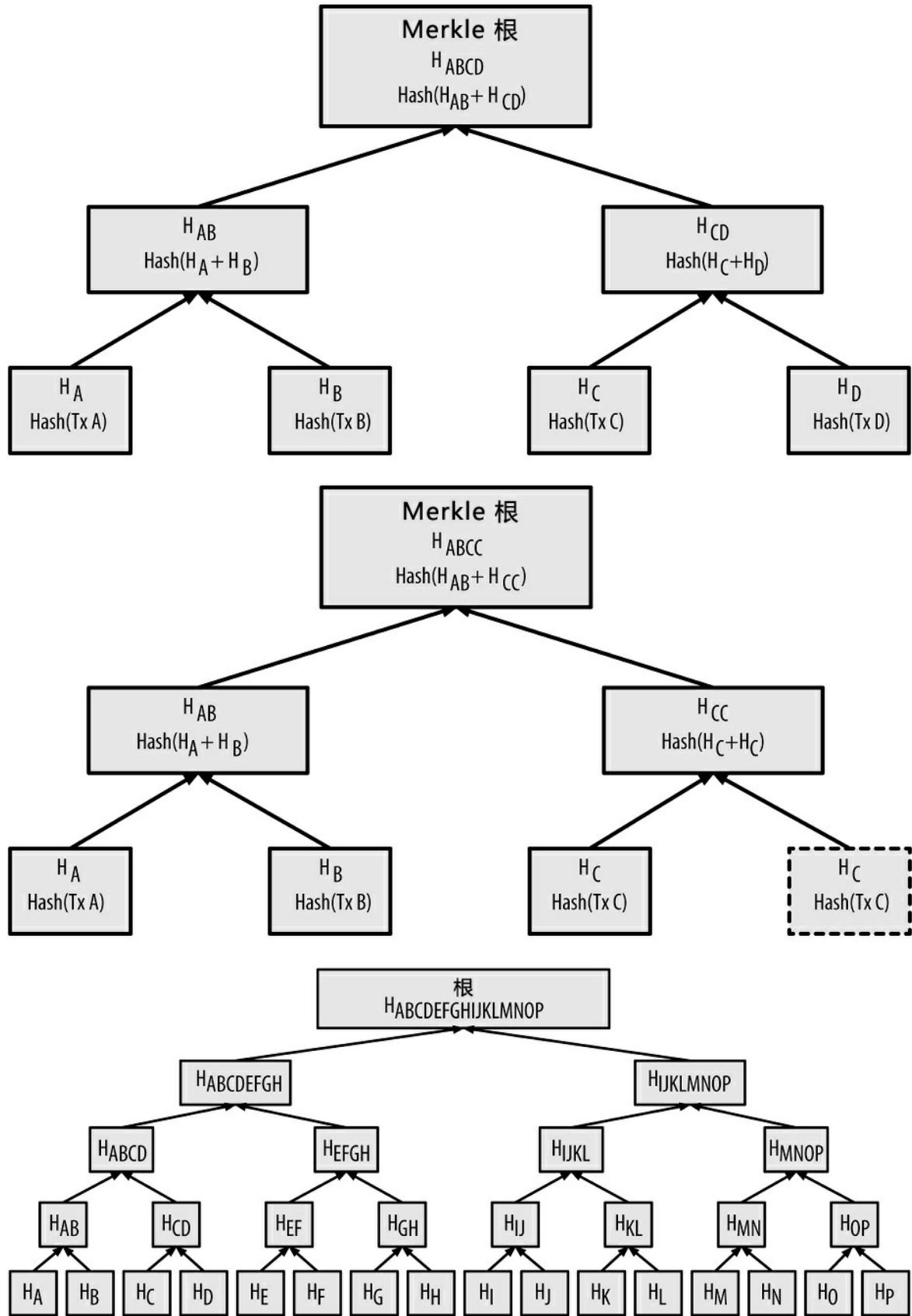
默克尔树的特点是，底层数据的任何变动，都会传递到其父节点，一直到树根。

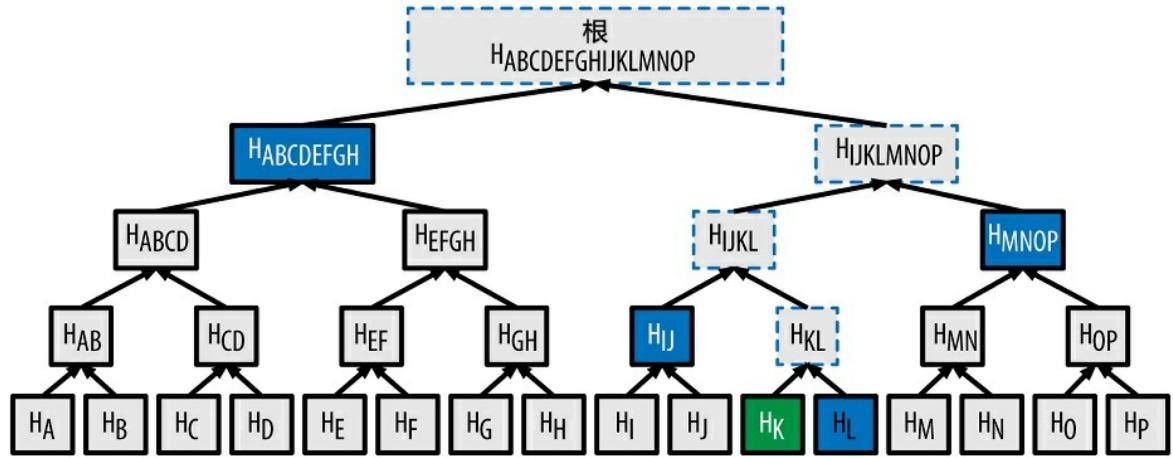
默克尔树的典型应用场景包括：

快速比较大量数据：当两个默克尔树根相同时，则意味着所代表的数据必然相同。

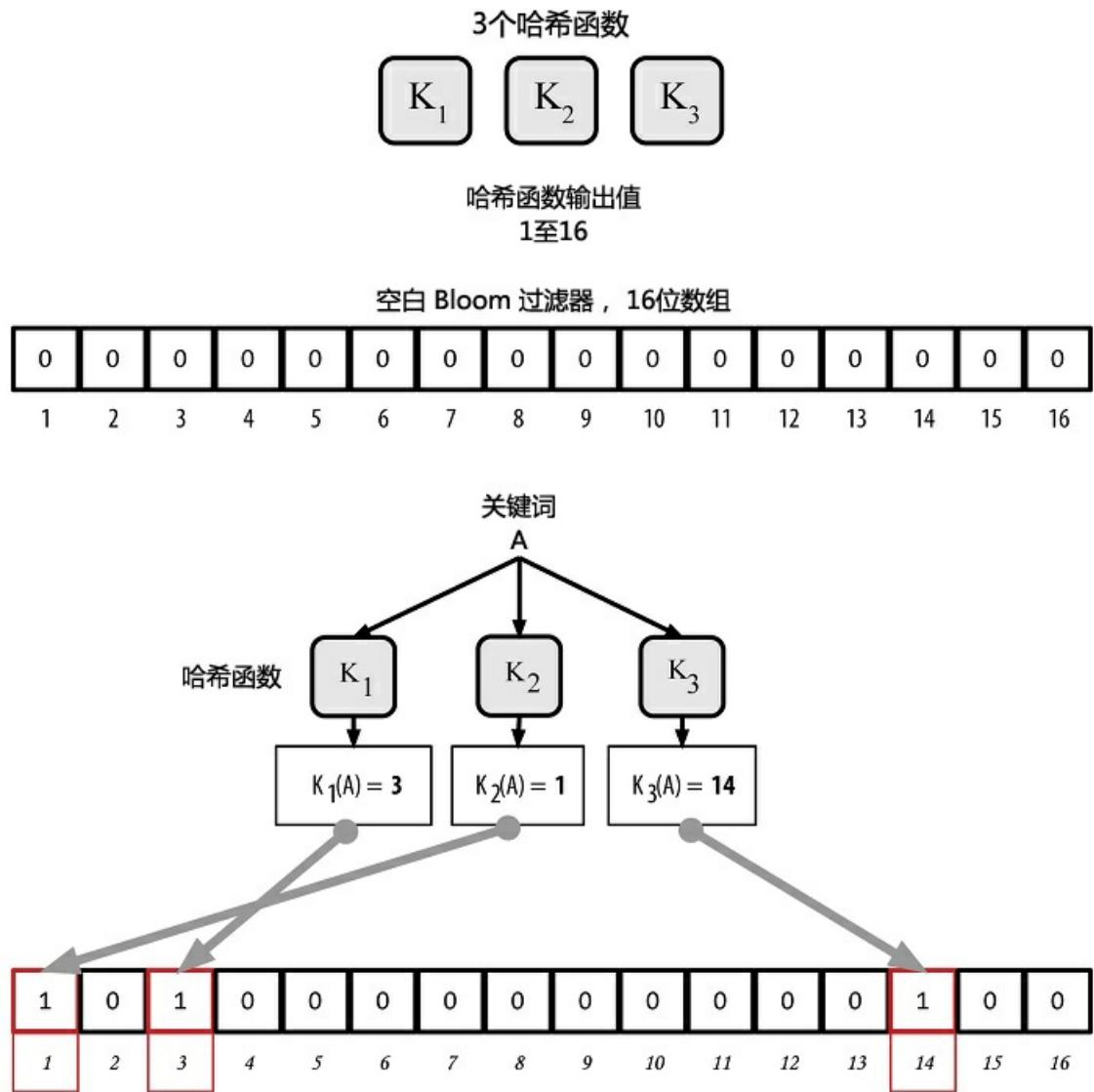
快速定位修改：例如上例中，如果 D1 中数据被修改，会影响到 N1, N4 和 Root。因此，沿着 Root --> N4 --> N1，可以快速定位到发生改变的 D1；

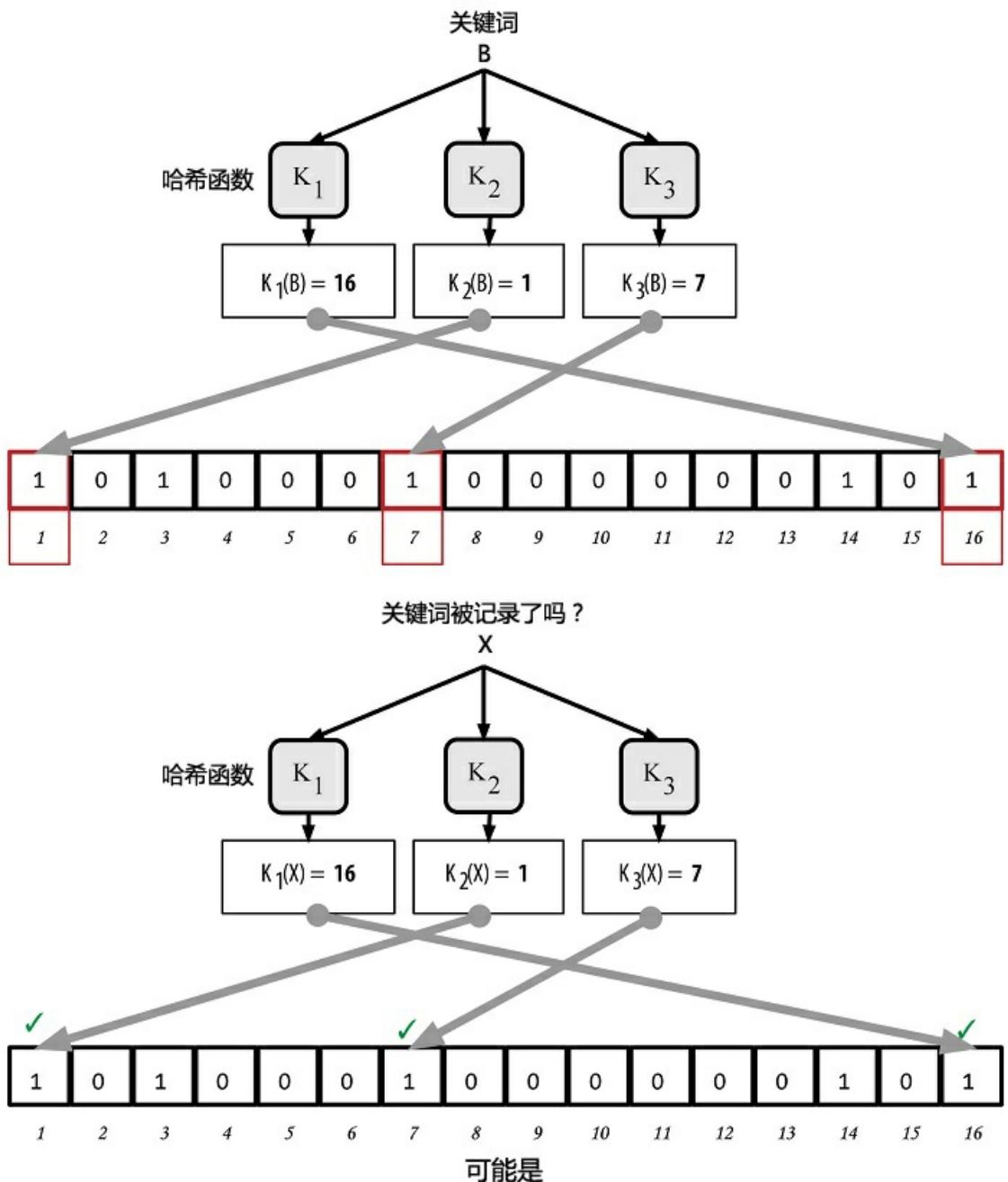
零知识证明：例如如何证明某个数据 (D0.....D3) 中包括给定内容 D0，很简单，构造一个默克尔树，公布 N0, N1, N4, Root, D0 拥有者可以很容易检测 D0 存在，但不知道其它内容。



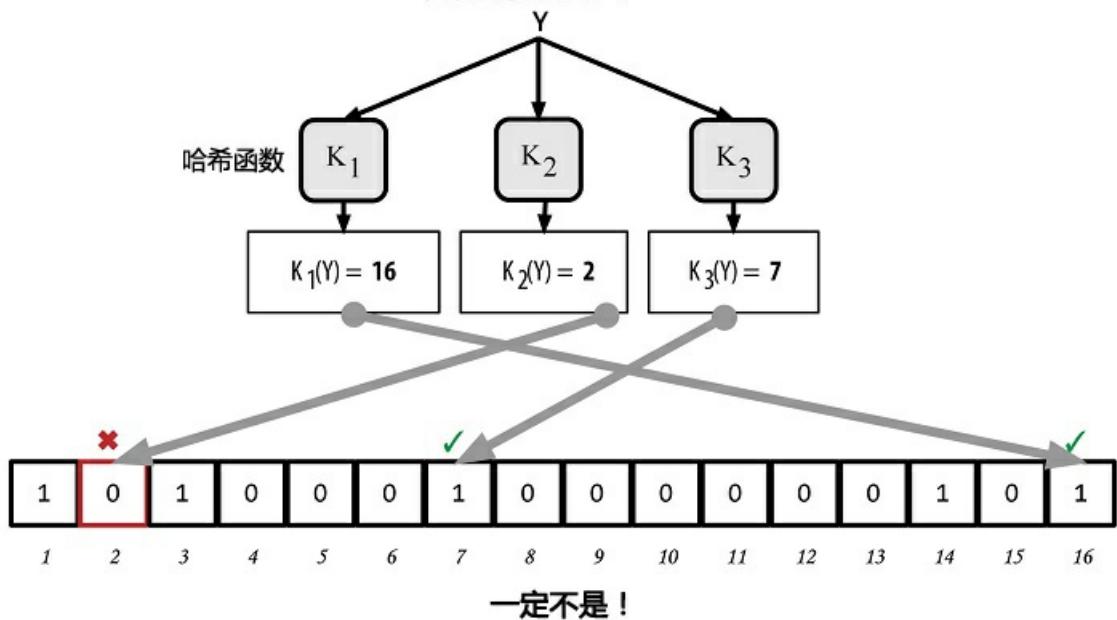


## Bloom过滤器

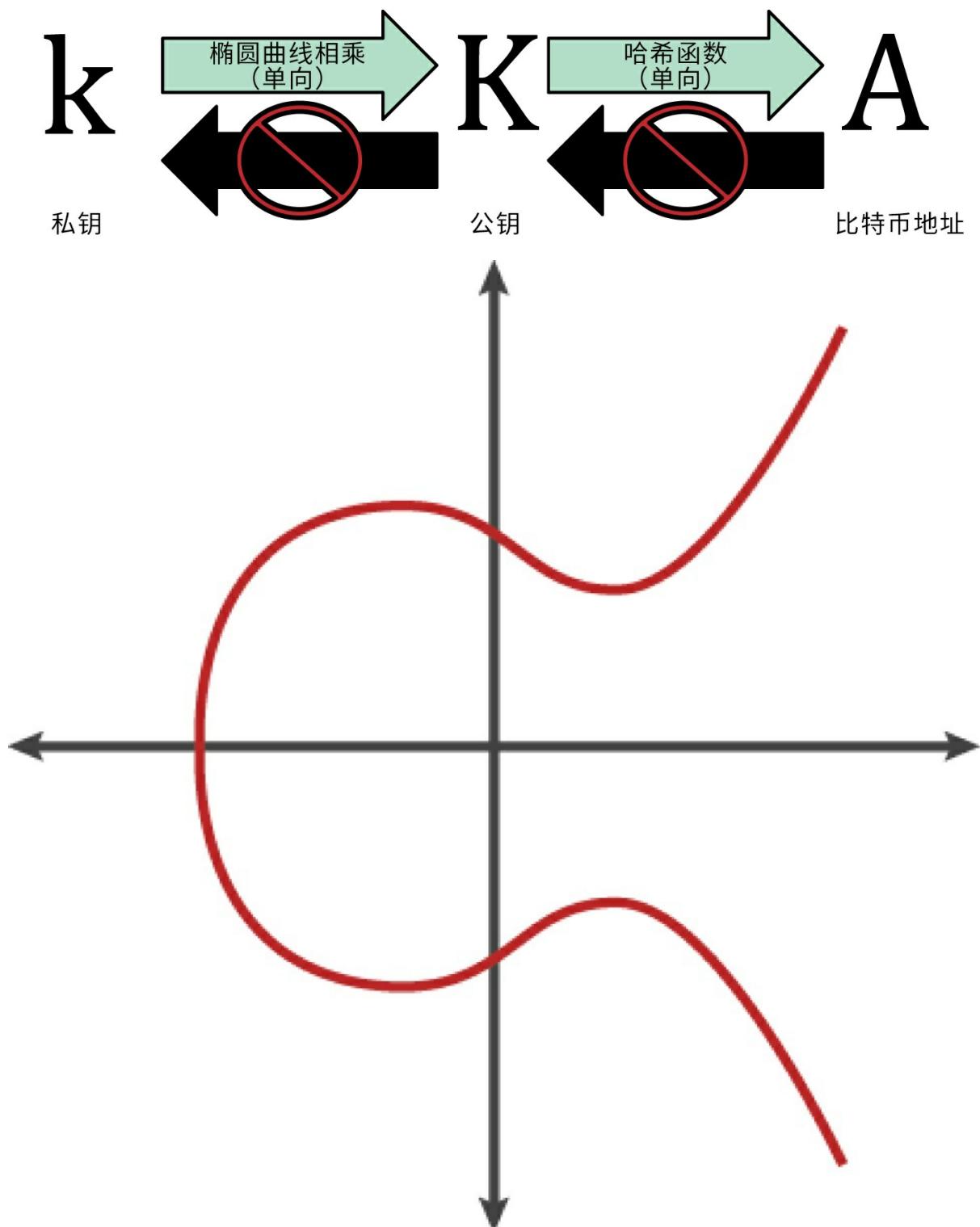


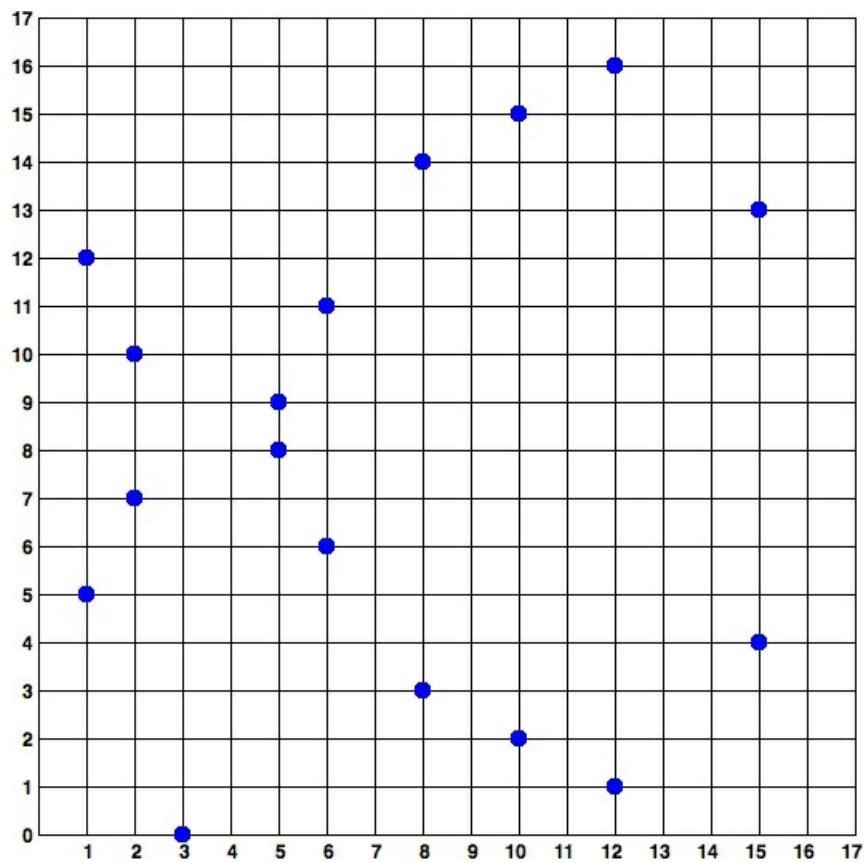


关键词被记录了吗？

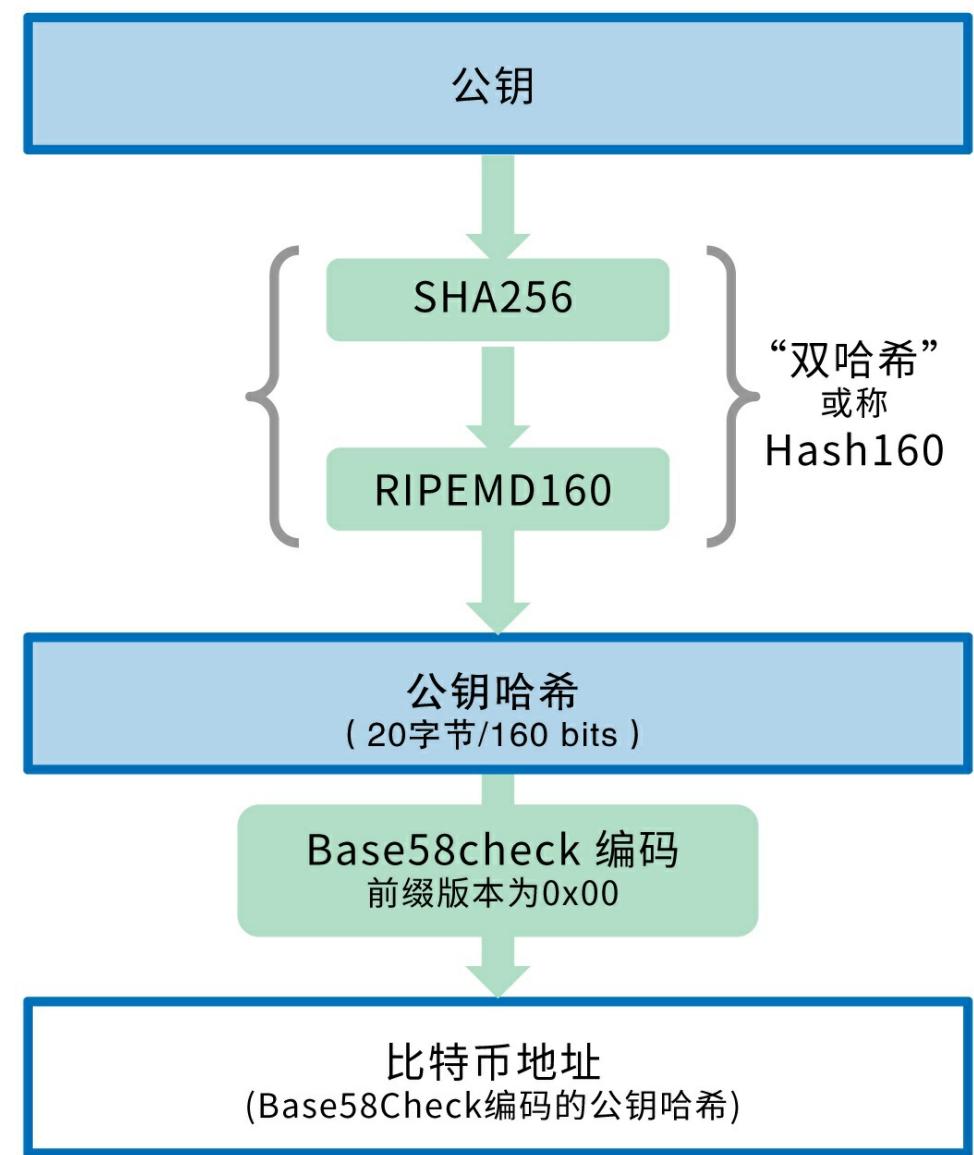


## 公钥私钥及椭圆曲线

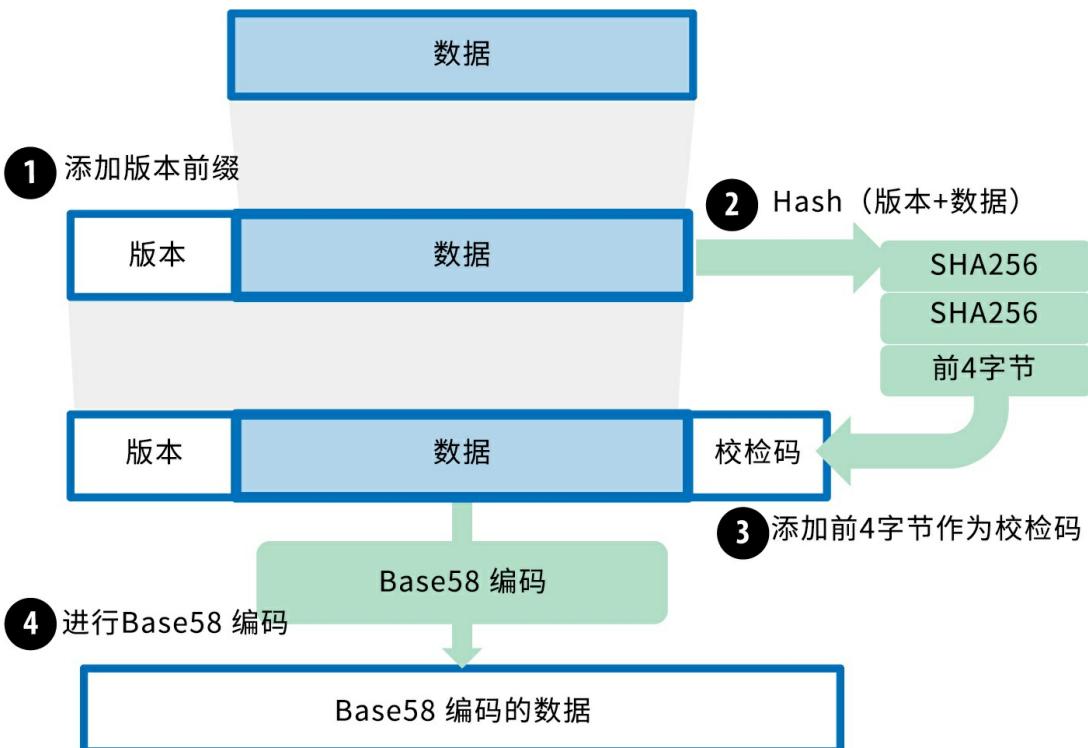




## 从公钥到比特币地址



## 从公钥到比特币地址



## 公钥压缩

[  $x$  ,  $y$  ]

公钥  
在曲线上的  
 $x$  和  $y$   
坐标



04  $x$   $y$

非压缩公钥  
的十六进制  
前缀为04

$y$ 为偶数

$y$ 为奇数

02  $x$

若 $y$  为偶数  
则压缩公钥  
的十六进制  
前缀为02

03  $x$

若 $y$  为奇数  
则压缩公钥  
的十六进制  
前缀为03

## 共识算法

待补充

# 分布式系统

## 分布式系统一致性要求

可终止性 (Termination) : 一致的结果在有限时间内能完成;

共识性 (Consensus) : 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。

## 强一致性与弱一致性

典型的强一致性算法通常有，顺序一致性 (Sequential Consistency) 和 线性一致性 (Linearizability Consistency) 等，但性能很差，实现复杂。

于此相对的，大家发现很多系统对一致性要求其实并没有那么高，只需要在一定的条件下让系统可以达到最终一致性 (Eventual Consistency) 就可以了，这种情形叫做弱一致性 (Weak Consistency)。

## FLP不可能原理

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。

提出该定理的论文是由 Fischer, Lynch 和 Patterson 三位作者于 1985 年发表，该论文后来获得了 Dijkstra 奖。

FLP 不可能原理实际上告诉人们，不要浪费时间去为异步分布式系统设计在任意场景下都能实现共识的算法。理解这一原理的一个不严谨的例子是：

三个人在不同房间，进行投票（投票结果是 0 或者 1）。三个人彼此可以通过电话进行沟通，但经常会有人时不时地睡着。比如某个时候，A 投票 0，B 投票 1，C 收到了两人的投票，然后 C 睡着了。A 和 B 则永远无法在有限时间内获知最终的结果。如果可以重新投票，则类似情形每次在取得结果前发生

FLP 原理实际上说明对于允许节点失效情况下，纯粹异步系统无法确保一致性在有限时间内完成。

## CAP理论

CAP理论：分布式计算系统不可能同时确保一致性 (Consistency)、可用性 (Availability) 和分区容忍性 (Partition)，设计中往往需要弱化对某个特性的保证。其中：

一致性 (Consistency)：任何操作应该都是原子的，发生在后面的事件能看到前面事件发生导致的结果，注意这里指的是强一致性；

可用性 (Availability)：在有限时间内，任何非失败节点都能应答请求；

分区容忍性 (Partition)：网络可能发生分区，即节点之间的通信不可保障。

比较直观地理解，当网络可能出现分区时候，系统是无法同时保证一致性和可用性的。要么，节点收到请求后因为没有得到其他人的确认就不应答，要么节点只能应答非一致的结果。

好在大部分时候网络被认为是可靠的，因此系统可以提供一致可靠的服务；当网络不可靠时，系统要么牺牲掉一致性（大部分时候都是如此），要么牺牲掉可用性。

既然 CAP 不可同时满足，则设计系统时候必然要弱化对某个特性的支持。

弱化一致性：对结果一致性不敏感的应用，可以允许在新版本上线后过一段时间才更新成功，期间不保证一致性。例如网站静态页面内容、实时性较弱的查询类数据库等，CouchDB、Cassandra 等为此设计。

**弱化可用性**: 对结果一致性很敏感的应用, 例如银行取款机, 当系统故障时候会拒绝服务。MongoDB、Redis 等为此设计。Paxos、Raft 等算法, 主要处理这种情况。

**弱化分区容忍性**: 现实中, 网络分区出现概率减小, 但较难避免。某些关系型数据库、ZooKeeper 即为此设计。实践中, 网络通过双通道等机制增强可靠性, 达到高稳定的网络通信。

## ACID 原则

ACID即 Atomicity(原子性)、Consistency(一致性)、Isolation(隔离性)、Durability(持久性)。ACID 原则描述了对分布式数据库的一致性需求, 同时付出了可用性的代价。

Atomicity: 每次操作是原子的, 要么成功, 要么不执行;

Consistency: 数据库的状态是一致的, 无中间状态;

Isolation: 各种操作彼此互相不影响;

Durability: 状态的改变是持久的, 不会失效。

一个与之相对的原则是 BASE( Basic Availability, Soft state, Eventually Consistency), 牺牲掉对一致性的约束(最终一致性), 来换取一定的可用性

## DLS原理

容错的上限:

(1) 在部分同步(partially synchronous)的网路环境中(即网路延迟有一定的上限, 但我们无法事先知道上限是多少), 协议可以容忍最多 $1/3$ 的拜占庭故障(Byzantine fault)。

(2) 在异步(asynchronous)的网路环境中, 具确定性质的协议无法容忍任何错误, 但这篇论文并没有提及randomized algorithms在这种情况下可以容忍最多 $1/3$ 的拜占庭故障。

(3) 在同步(synchronous)的网路环境中(即网路延迟有上限且上限是已知的), 协议可以容忍100%的拜占庭故障, 但当超过 $1/2$ 的节点为恶意节点时, 会有一些限制条件。

要注意的是, 我们考虑的是"具认证特性的拜占庭模型(authenticated Byzantine)", 而不是"一般的拜占庭模型";具认证特性指的是将如今已经过大量研究且成本低廉的公私钥加密机制应用在我们的演算法中。

## Paxos与Raft算法

Paxos 问题是指分布式的系统中存在故障 (fault)，但不存在恶意 (corrupt) 节点场景 (即可能消息丢失或重复，但无错误消息) 下的共识达成 (Consensus) 问题。因为最早是 Leslie Lamport 用 Paxos 岛的故事模型来进行描述而命名。

### Paxos

1990 年由 Leslie Lamport 提出的 Paxos 共识算法，在工程角度实现了一种最大化保障分布式系统一致性（存在极小的概率无法实现一致）的机制。Paxos 被广泛应用在 Chubby、ZooKeeper 这样的系统中，Leslie Lamport 因此获得了 2013 年度图灵奖。

故事背景是古希腊 Paxos 岛上的多个法官在一个大厅内对一个议案进行表决，如何达成统一的结果。他们之间通过服务人员来传递纸条，但法官可能离开或进入大厅，服务人员可能偷懒去睡觉。

Paxos 是第一个被证明的共识算法，其原理基于两阶段提交并进行扩展。

作为现在共识算法设计的鼻祖，以最初论文的难懂（算法本身并不复杂）出名。算法中将节点分为三种类型：

proposer：提出一个提案，等待大家批准为结案。往往是客户端担任该角色；

acceptor：负责对提案进行投票。往往是服务端担任该角色；

learner：被告知结案结果，并与之统一，不参与投票过程。可能为客户端或服务端。

并且，算法需要满足 safety 和 liveness 两方面的约束要求（实际上这两个基础属性是大部分分布式算法都该考虑的）：

safety：保证决议结果是对的，无歧义的，不会出现错误情况。

决议 (value) 只有在被 proposers 提出的 proposal 才能被最终批准；

在一次执行实例中，只批准 (chosen) 一个最终决议，意味着多数接受 (accept) 的结果能成为决议；

liveness：保证决议过程能在有限时间内完成。

决议总会产生，并且 learners 能获得被批准 (chosen) 的决议。

基本过程包括 proposer 提出提案，先争取大多数 acceptor 的支持，超过一半支持时，则发送结案结果给所有人进行确认。一个潜在的问题是 proposer 在此过程中出现故障，可以通过超时机制来解决。极为凑巧的情况下，每次新一轮提案的 proposer 都恰好故障，系统则永远无法达成一致（概率很小）。

Paxos 能保证在超过  $1/2$  的正常节点存在时，系统能达成共识。

### 单个提案者+多接收者

如果系统中限定只有某个特定节点是提案者，那么一致性肯定能达成（只有一个方案，要么达成，要么失败）。提案者只要收到了来自多数接收者的投票，即可认为通过，因为系统中不存在其他的提案。

但一旦提案者故障，则系统无法工作。

### 多个提案者+单个接收者

限定某个节点作为接收者。这种情况下，共识也很容易达成，接收者收到多个提案，选第一个提案作为决议，拒绝掉后续的提案即可。

缺陷也是容易发生单点故障，包括接收者故障或首个提案者节点故障。以上两种情形其实类似主从模式，虽然不那么可靠，但因为原理简单而被广泛采用。当提案者和接收者都推广到多个的情形，会出现一些挑战。

### 多个提案者+多个接收者

既然限定单提案者或单接收者都会出现故障，那么就得允许出现多个提案者和多个接收者。问题一下子变得复杂了。

一种情况是同一时间片段（如一个提案周期）内只有一个提案者，这时可以退化到单提案者的情形。需要设计一种机制来保障提案者的正确产生，例如按照时间、序列、或者大家猜拳（出一个数字来比较）之类。考虑到分布式系统要处理的工作量很大，这个过程要尽量高效，满足这一条件的机制非常难设计。

另一种情况是允许同一时间片段内可以出现多个提案者。那同一个节点可能收到多份提案，怎么对他们进行区分呢？这个时候采用只接受第一个提案而拒绝后续提案的方法也不适用。很自然的，提案需要带上不同的序号。节点需要根据提案序号来判断接受哪个。比如接受其中序号较大（往往意味着是接受新提出的，因为旧提案者故障概率更大）的提案。

如何为提案分配序号呢？一种可能方案是每个节点的提案数字区间彼此隔离开，互相不冲突。为了满足递增的需求可以配合用时间戳作为前缀字段。

此外，提案者即便收到了多数接收者的投票，也不敢说就一定通过。因为在此过程中系统可能还有其它的提案。

## 两阶段的提交

提案者发出提案之后，收到一些反馈。一种结果是自己的提案被大多数接受了，另一种结果是没被接受。没被接受的话好说，过会再试试。

即便收到来自大多数的接受反馈，也不能认为就最终确认了。因为这些接收者自己并不知道自己刚反馈的提案就恰好是全局的绝大多数。

很自然的，引入了新的一个阶段，即提案者在前一阶段拿到所有的反馈后，判断这个提案是可能被大多数接受的提案，需要对其进行最终确认。

Paxos 里面对这两个阶段分别命名为准备（prepare）和提交（commit）。准备阶段解决大家对哪个提案进行投票的问题，提交阶段解决确认最终值的问题。

准备阶段：

提案者发送自己计划提交的提案的编号到多个接受者，试探是否可以锁定多数接受者的支持。

接受者时刻保留收到过提案的最大编号和接受的最大提案。如果收到的提案号比目前保留的最大提案号还大，则返回自己已接受的提案值（如果还未接受过任何提案，则为空）给提案者，更新当前最大提案号，并说明不再接受小于最大提案号的提案。

提交阶段：

提案者如果收到大多数的回复（表示大部分人听到它的请求），则可准备发出带有刚才提案号的接受消息。如果收到的回复中不带有新的提案，说明锁定成功，则使用自己的提案内容；如果返回中有提案内容，则替换提案值为返回中编号最大的提案值。如果没收到足够多的回复，则需要再次发出请求。

接受者收到接受消息后，如果发现提案号不小于已接受的最大提案号，则接受该提案，并更新接受的最大提案。

一旦多数接受了共同的提案值，则形成决议，成为最终确认的提案。

## Raft

Raft 算法是 Paxos 算法的一种简化实现。

包括三种角色：leader、candidate 和 follower，其基本过程为：

Leader 选举：每个 candidate 随机经过一定时间都会提出选举方案，最近阶段中得票最多者被选为 leader；

同步 log：leader 会找到系统中 log 最新的记录，并强制所有的 follower 来刷新到这个记录；

注：此处 log 并非是指日志消息，而是各种事件的发生记录。



## 拜占庭容错算法

拜占庭问题更为广泛，讨论的是允许存在少数节点作恶（消息可能被伪造）场景下的一致性达成问题。拜占庭算法讨论的是最坏情况下的保障。

### 中国将军问题

拜占庭将军问题之前，就已经存在中国将军问题：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，这个问题无解。

### 拜占庭问题

又叫拜占庭将军（Byzantine Generals Problem）问题，是 Leslie Lamport 1982 年提出用来解释一致性问题的一个虚构模型。拜占庭是古代东罗马帝国的首都，由于地域宽广，守卫边境的多个将军（系统中的多个节点）需要通过信使来传递消息，达成某些一致的决定。但由于将军中可能存在叛徒（系统中节点出错），这些叛徒将努力向不同的将军发送不同的消息，试图会干扰一致性的达成。

拜占庭问题即为在此情况下，如何让忠诚的将军们能达成行动的一致。

对于拜占庭问题来说，假如节点总数为  $N$ ，叛变将军数为  $F$ ，则当  $N > 2F$  时，问题才有解，即 Byzantine Fault Tolerant (BFT) 算法。

例如， $N=3, F=1$  时。

提案人不是叛变者，提案人发送一个提案出来，叛变者可以宣称收到的是相反的命令。则对于第三个人（忠诚者）收到两个相反的消息，无法判断谁是叛变者，则系统无法达到一致。

提案人是叛变者，发送两个相反的提案分别给另外两人，另外两人都收到两个相反的消息，无法判断究竟谁是叛变者，则系统无法达到一致。

更一般的，当提案人不是叛变者，提案人提出提案信息 1，则对于合作者来看，系统中会有  $N-F$  份确定的信息 1，和  $F$  份不确定的信息（可能为 1 或 0，假设叛变者会尽量干扰一致的达成）， $N-F > F$ ，即  $N > 2F$  情况下才能达成一致。

当提案人是叛变者，会尽量发送相反的提案给  $N-F$  个合作者，从收到 1 的合作者看来，系统中会存在 1 个信息 1，以及 0 个信息 0；从收到 0 的合作者看来，系统中会存在 0 个信息 0，以及 1 个信息 1；

另外存在  $F-1$  个不确定的信息。合作者要想达成一致，必须进一步的对所获得的消息进行判定，询问其他人某个被怀疑对象的消息值，并通过取多数来作为被怀疑者的信息值。这个过程可以进一步递归下去。

Leslie Lamport 证明，当叛变者不超过  $F$  时，存在有效的算法，不论叛变者如何折腾，忠诚的将军们总能达成一致的结果。如果叛变者过多，则无法保证一定能达到一致性。

多于  $F$  的叛变者时有没有可能有解决方案呢？设想  $f$  个叛变者和  $g$  个忠诚者，叛变者故意使坏，可以给出错误的结果，也可以不响应。某个时候  $f$  个叛变者都不响应，则  $g$  个忠诚者取多数既能得到正确结果。当  $f$  个叛变者都给出一个恶意的提案，并且  $g$  个忠诚者中有  $f$  个离线时，剩下的  $g-f$  个忠诚者此时无法分别是否混入了叛变者，仍然要确保取多数能得到正确结果，因此， $g-f > f$ ，即  $g > 2f$ ，所以系统整体规模要大于  $3f$ 。

能确保达成一致的拜占庭系统节点数至少为 4，允许出现 1 个坏的节点。

### Byzantine Fault Tolerant 算法

面向拜占庭问题的容错算法，解决的是网络通信可靠，但节点可能故障情况下的一致性达成。

最早由 Castro 和 Liskov 在 1999 年提出的 Practical Byzantine Fault Tolerant (PBFT) 是第一个得到广泛应用的 BFT 算法。只要系统中有  $2f+1$  的节点是正常工作的，则可以保证一致性。

PBFT 算法包括三个阶段来达成共识：Pre-Prepare、Prepare 和 Commit。

## 新的解决思路

拜占庭问题之所以难解，在于任何时候系统中都可能存在多个提案（因为提案成本很低），并且要完成最终的一致性确认过程十分困难，容易受干扰。但是一旦确认，即为最终确认。

比特币的区块链网络在设计时提出了创新的 PoW ( Proof of Work) 算法思路。一个是限制一段时间内整个网络中出现提案的个数（增加提案成本），另外一个是放宽对最终一致性确认的需求，约定好大家都确认并沿着已知最长的链进行拓宽。系统的最终确认是概率意义上的存在。这样，即便有人试图恶意破坏，也会付出很大的经济代价（付出超过系统一半的算力）。

后来的各种 PoX 系列算法，也都是沿着这个思路进行改进，采用经济上的惩罚来制约破坏者。

## Bitcoin

项目	内容
链名称	Bitcoin
链类型	公链
匿名性	匿名
共识机制	Pow: Proof of Work
开发语言	CPP
合约语言	只是利用合约实现了业务逻辑
官方网站	<a href="http://bitcoin.org/">http://bitcoin.org/</a>
白皮书	<a href="https://bitcoin.org/bitcoin.pdf">https://bitcoin.org/bitcoin.pdf</a>
项目地址	<a href="https://github.com/bitcoin/bitcoin">https://github.com/bitcoin/bitcoin</a>

## 项目特点

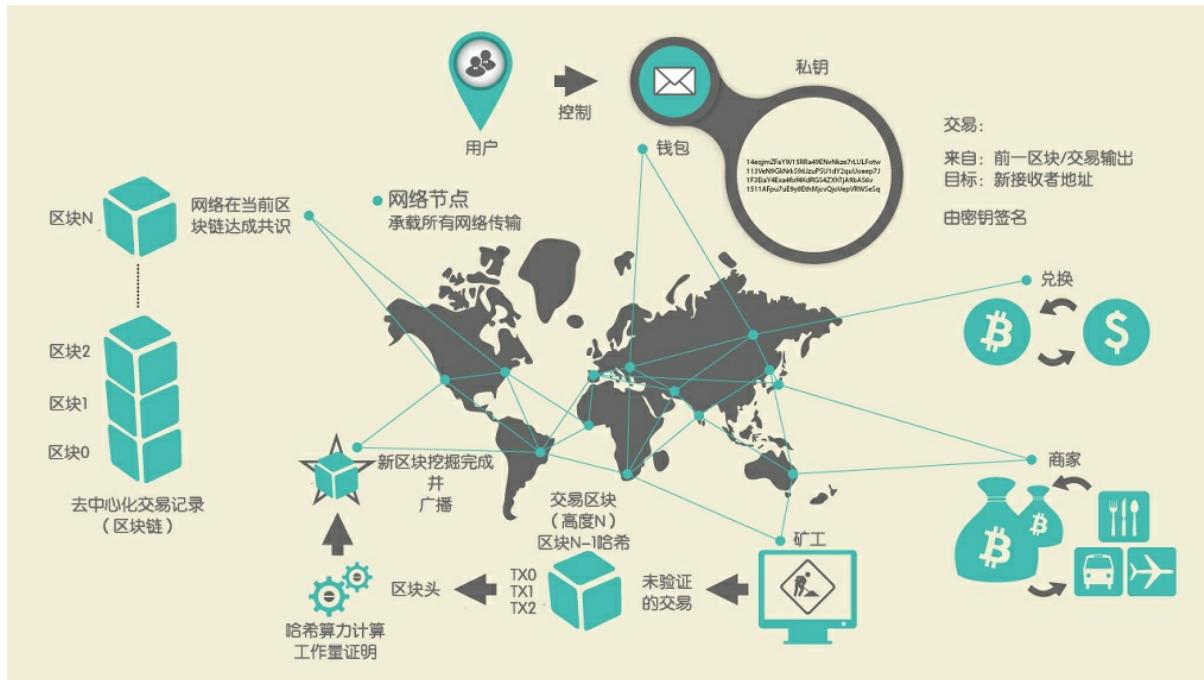
一个去中心化的点对点网络(比特币协议)：通过P2P技术解决了中心化问题。一个公共的交易账簿(区块链)：比特币通过PoW的共识机制，通过CPU算力作为投票权，解决了所谓的拜占庭问题，成为了所有区块链项目的鼻祖。一个去中心化的数学的和确定性的货币发行(分布式挖矿)：比特币不是第一个数字货币系统，甚至不是第一个成功的数字货币系统，但它是第一个被广泛接受的去中心化数字货币系统。一个去中心化的交易验证系统(交易脚本)：同时比特币其实已经有了一个非图灵完整的虚拟机，但由于其自身定位问题，只用于内部使用，并没有将其开放给普通用户，一定程度上限制了其走向更加繁荣。

# 整体架构

## 比特币架构



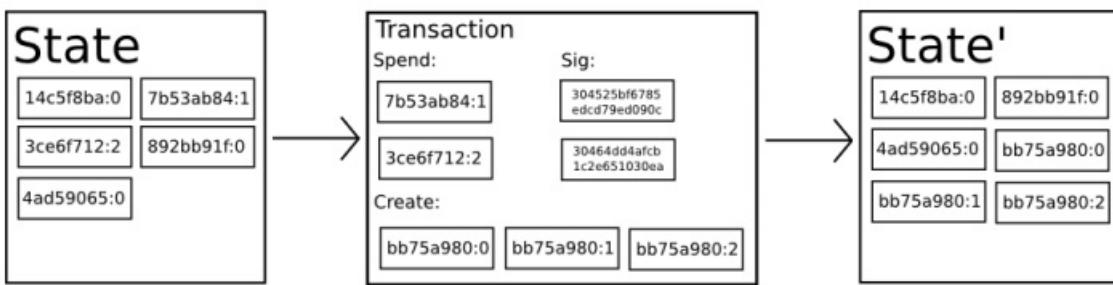
### 比特币架构总览



## 状态转换

### 状态转换

#### 作为状态转换系统的比特币



从技术角度讲，比特币账本可以被认为是一个状态转换系统，该系统包括所有现存的比特币所有权状态和“状态转换函数”。状态转换函数以当前状态和交易为输入，输出新的状态。

比特币系统的“状态”是所有已经被挖出的、没有花费的比特币(技术上称为“未花费的交易输出，unspent transaction outputs 或 UTXO”)的集合。每个 UTXO 都有一个面值和所有者(由 20 个字节的本质上是密码学公钥的地址所定义[1])。一笔交易包括一个或多个输入和一个或多个输出。每个输入包含一个对现有 UTXO 的引用和由与所有者地址相对应的私钥创建的密码学签名。每个输出包含一个新的加入到状态中的 UTXO。

例如，在标准的银行系统中，状态就是一个资产负债表，一个从 A 账户向 B 账户转账 X 美元的请求是一笔交易，状态转换函数将从 A 账户中减去 X 美元，向 B 账户增加 X 美元。如果 A 账户的余额小于 X 美元，状态转换函数就会返回错误提示。所以我们可以如下定义状态转换函数：

APPLY(S,TX) > S' or ERROR

在上面提到的银行系统中，状态转换函数如下：

APPLY({ Alice: \$50, Bob: \$50 }, "send \$20 from Alice to Bob") = { Alice:\$30,Bob: \$70 }

但是：

APPLY({ Alice: \$50, Bob: \$50 }, "send \$70 from Alice to Bob") = ERROR

比特币系统的“状态”是所有已经被挖出的、没有花费的比特币(技术上称为“未花费的交易输出，unspent transaction outputs 或 UTXO”)的集合。每个 UTXO 都有一个面值和所有者(由 20 个字节的本质上是密码学公钥的地址所定义[1])。一笔交易包括一个或多个输入和一个或多个输出。每个输入包含一个对现有 UTXO 的引用和由与所有者地址相对应的私钥创建的密码学签名。每个输出包含一个新的加入到状态中的 UTXO。

在比特币系统中，状态转换函数 APPLY(S,TX)->S' 大体上可以如下定义：

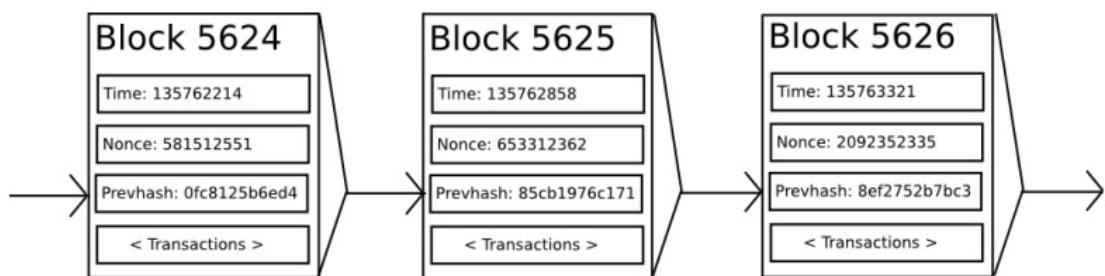
1. 交易的每个输入：
  - 如果引用的 UTXO 不存在于现在的状态中 (S)，返回错误提示
  - 如果签名与 UTXO 所有者的签名不一致，返回错误提示
2. 如果所有的 UTXO 输入面值总额小于所有的 UTXO 输出面值总额，返回错误提示
3. 返回新状态 S'，新状态 S 中移除了所有的输入 UTXO，增加了所有的输出 UTXO。

第一步的第一部分防止交易的发送者花费不存在的比特币，第二部分防止交易的发送者花费其他人的比特币。第二步确保价值守恒。比特币的支付协议如下。假设 Alice 想给 Bob 发送 11.7BTC。事实上，Alice 不可能正好有 11.7BTC。假设，她能得到的最小数额比特币的方式是：6+4+2=12。所以，她可以创建一笔有 3 个输入，2 个输出的交易。第一个输出的面值是

11.7BTC, 所有者是 Bob( Bob的比特币地址), 第二个输出的面值是 0.3BTC, 所有者是 Alice 自己, 也就是找零。

## 区块确认

### 区块确认



每个区块包含一个时间戳、一个随机数、一个对上一个区块的引用(即 哈希)和上一区块生成以来发生的所有交易列表。这样随着时间流逝就创建出了一个持续增长的区块链，它不断地更新，从而能够代表比特币账本的最新状态。

依照这个范式，检查一个区块是否有效的算法如下：

1. 检查区块引用的上一个区块是否存在且有效。
2. 检查区块的时间戳是否晚于以前的区块的时间戳，而且早于未来 2 小时[2]。
3. 检查区块的工作量证明是否有效。
4. 将上一个区块的最终状态赋于  $S[0]$ 。
5. 假设  $TX$  是区块的交易列表，包含  $n$  笔交易。对于属于  $0 \dots n-1$  的所有  $i$ ，进行状态转换  $S[i+1] = \text{APPLY}(S[i], TX[i])$ 。如果任何一笔交易  $i$  在状态转换中出错，退出程序，返回错误。
6. 返回正确，状态  $S[n]$  是这一区块的最终状态。

## 挖矿

本质上，区块中的每笔交易必须提供一个正确的状态转换，要注意的是，“状态”并不是编码到区块的。它纯粹只是被校验节点记住的抽象概念，对于任意区块都可以从创世状态开始，按顺序加上每一个区块的每一笔交易，(妥妥地)计算出当前的状态。另外，需要注意矿工将交易收录进区块的顺序。如果一个区块中有A、B 两笔交易，B 花费的是 A 创建的 UTXO，如果 A 在 B 以前，这个区块是有效的，否则，这个区块是无效的。

区块验证算法的有趣部分是“工作量证明”概念：对每个区块进行 SHA256 哈希处理，将得到的哈希视为长度为 256 比特的数值，该数值必须小于不断动态调整的目标数值，本书写作时目标数值大约是  $2^{190}$ 。工作量证明的目的是使区块的创建变得困难，从而阻止女巫攻击者恶意重新生成区块链。因为 SHA256 是完全不可预测的伪随机函数，创建有效区块的唯一方法就是简单地不断试错，不断地增加随机数的数值，查看新的哈希数值是否小于目标数值。如果当前的目标数值是  $2^{192}$ ，就意味着平均需要尝试  $2^{64}$  次才能生成有效的区块。一般而言，比特币网络每隔 2016 个区块重新设定目标数值，保证平均每十分钟生成一个区块。为了对矿工的计算工作进行奖励，每一个成功生成区块的矿工有权在区块中包含一笔凭空发给他们自己 25BTC 的交易。另外，如果交易的输入大于输出，差额部分就作为“交易费用”付给矿工。顺便提一下，对矿工的奖励是比特币发行的唯一机制，创世状态中并没有比特币。

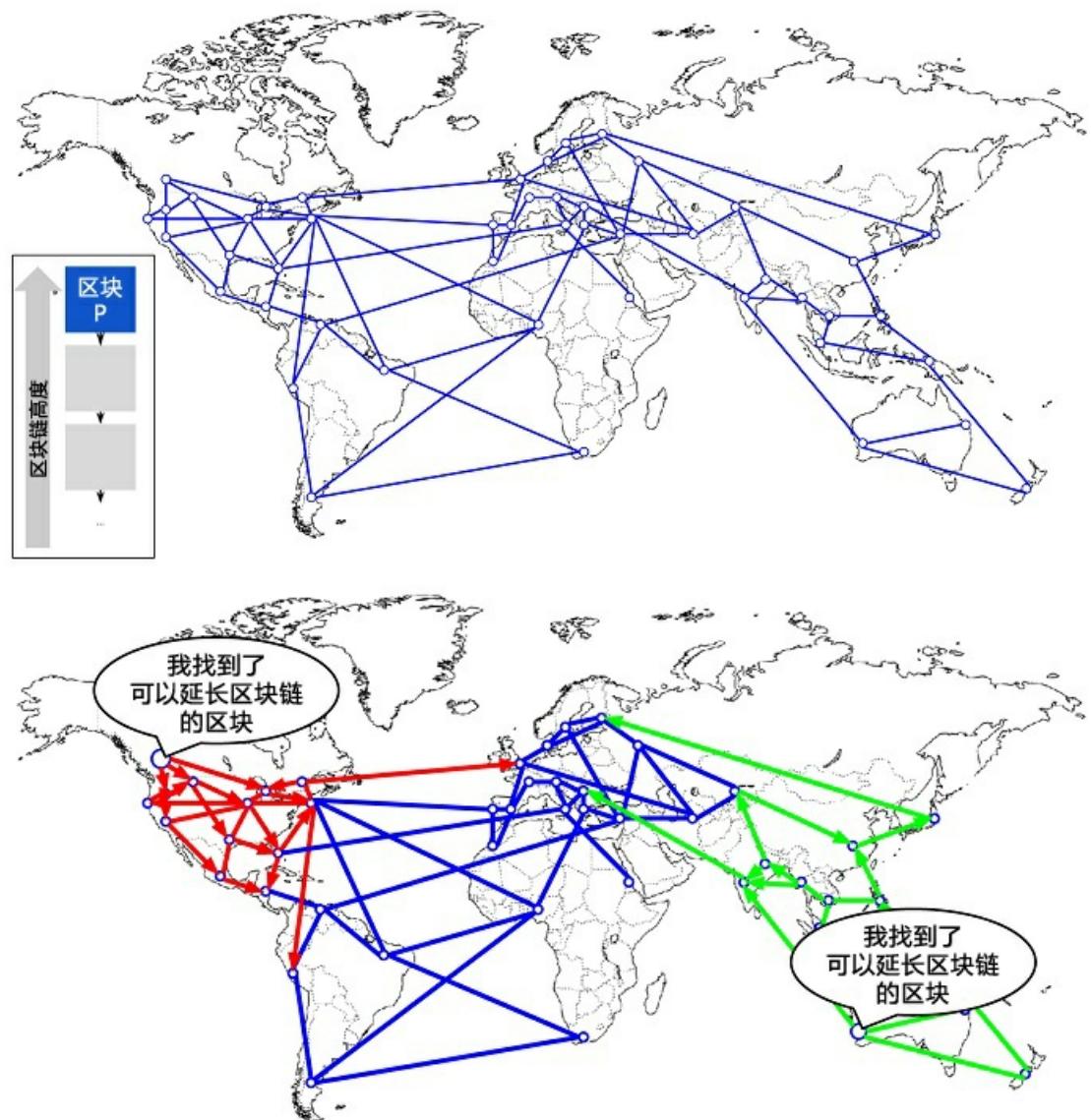
为了更好地理解挖矿的目的，让我们分析比特币网络出现恶意攻击者时会发生什么。因为比特币的密码学基础是非常安全的，所以攻击者会选择攻击没有被密码学直接保护的部分：交易顺序。攻击者的策略非常简单：

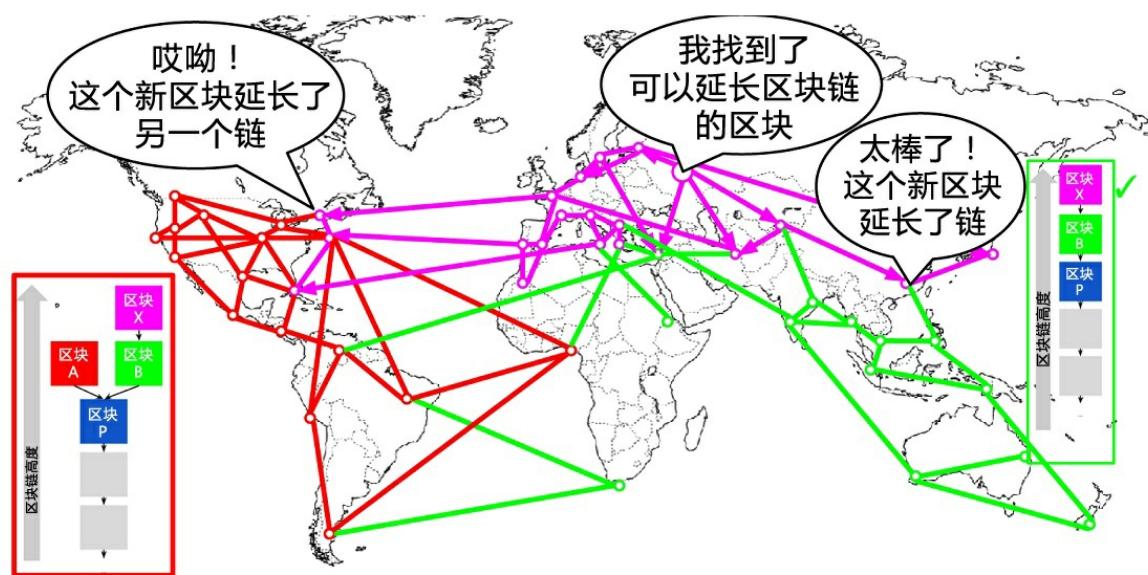
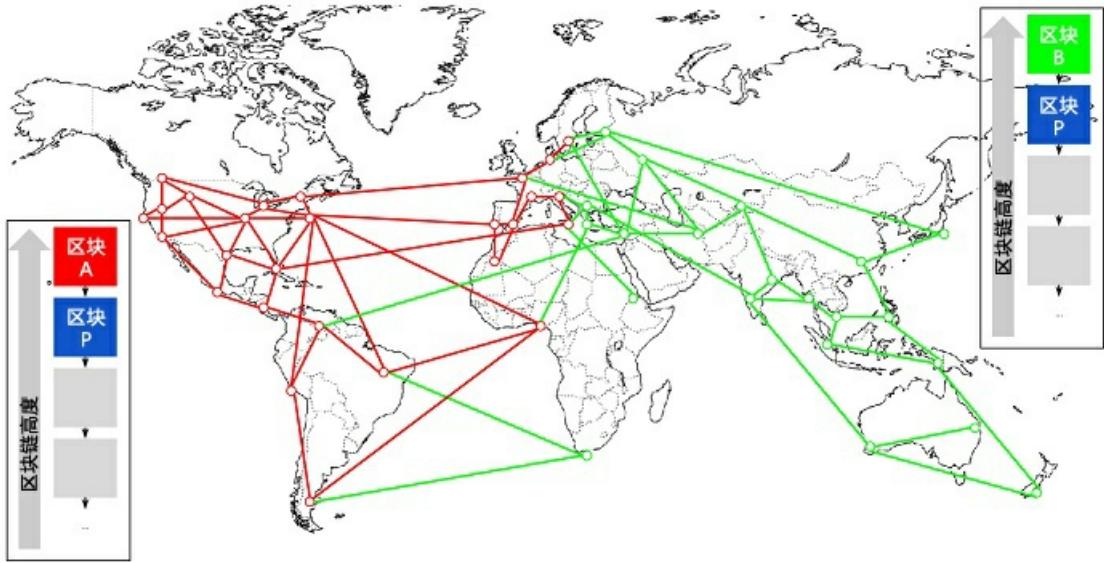
1. 向卖家发送 100BTC 购买商品(尤其是无需邮寄的电子商品)。
2. 等待直至商品发出。
3. 创建另一笔交易，将相同的 100BTC 发送给自己的账户。

#### 4. 使比特币网络相信发送给自己账户的交易是最先发出的。

一旦步骤(1)发生，几分钟后矿工将把这笔交易打包到区块，假设是第270000个区块。大约一个小时以后，在此区块后面将会有五个区块，每个区块间接地指向这笔交易，从而确认这笔交易。这时卖家收到货款，并向买家发货。因为我们假设这是数字商品，攻击者可以即时收到货。现在，攻击者创建另一笔交易，将相同的100BTC发送到自己的账户。如果攻击者只是向全网广播这一消息，这一笔交易不会被处理。矿工会运行状态转换函数 `APPLY(S,TX)`，发现这笔交易将花费已经不在状态中的UTXO。所以，攻击者会对区块链进行分叉，将第269999个区块作为父区块重新生成第270000个区块，在此区块中用新的交易取代旧的交易。因为区块数据是不同的，这要求重新进行工作量证明。另外，因为攻击者生成的新的第270000个区块有不同的哈希，所以原来的第270001到第270005的区块不指向它，因此原有的区块链和攻击者的新区块是完全分离的。在发生区块链分叉时，区块链长的分支被认为是诚实的区块链，合法的矿工将会沿着原有的第270005区块后挖矿，只有攻击者一人在新的第270000区块后挖矿。攻击者为了使得他的区块链最长，他需要拥有比除了他以外的全网更多的算力来追赶(即51%攻击)。

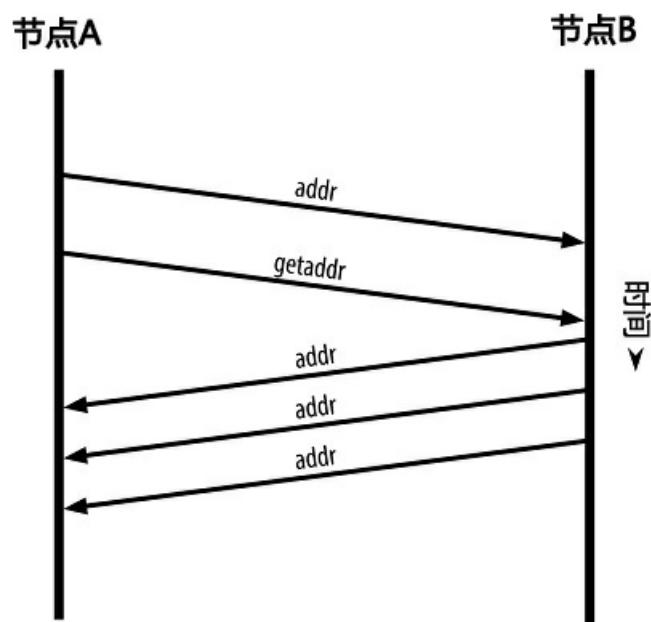
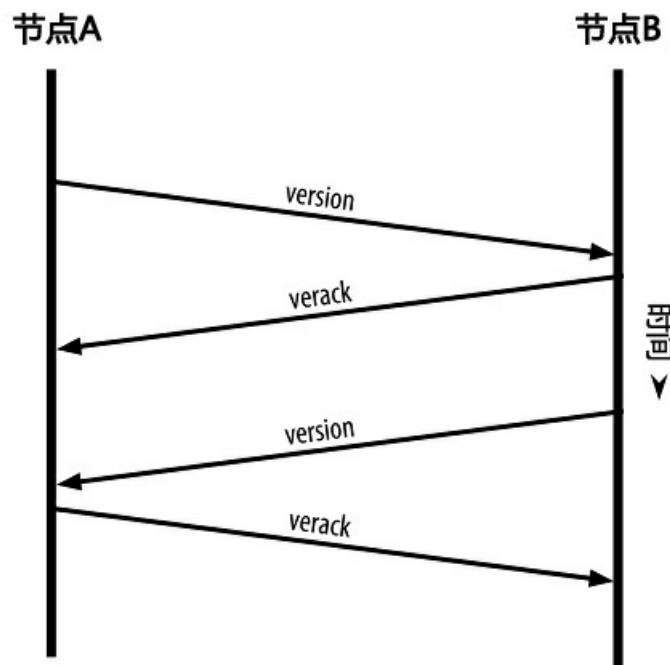
## 分叉处理

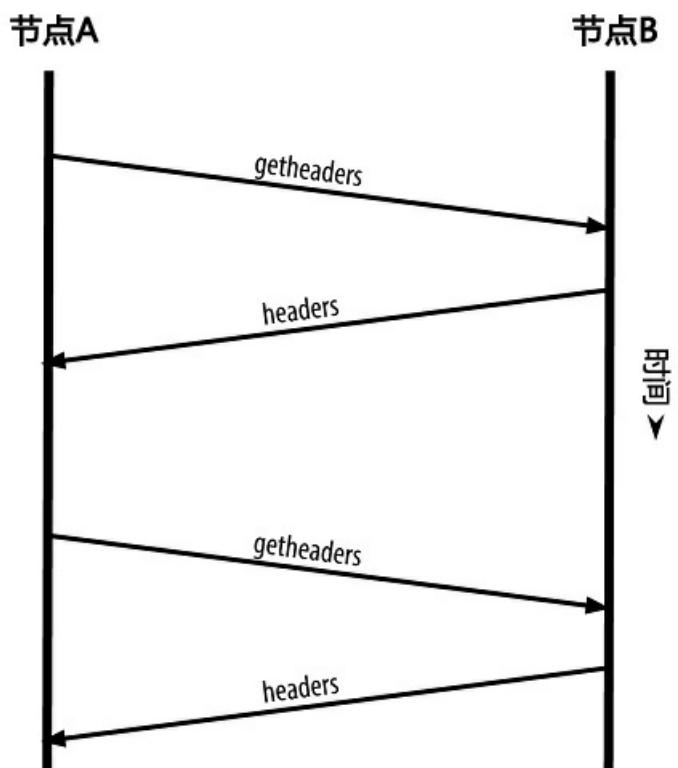
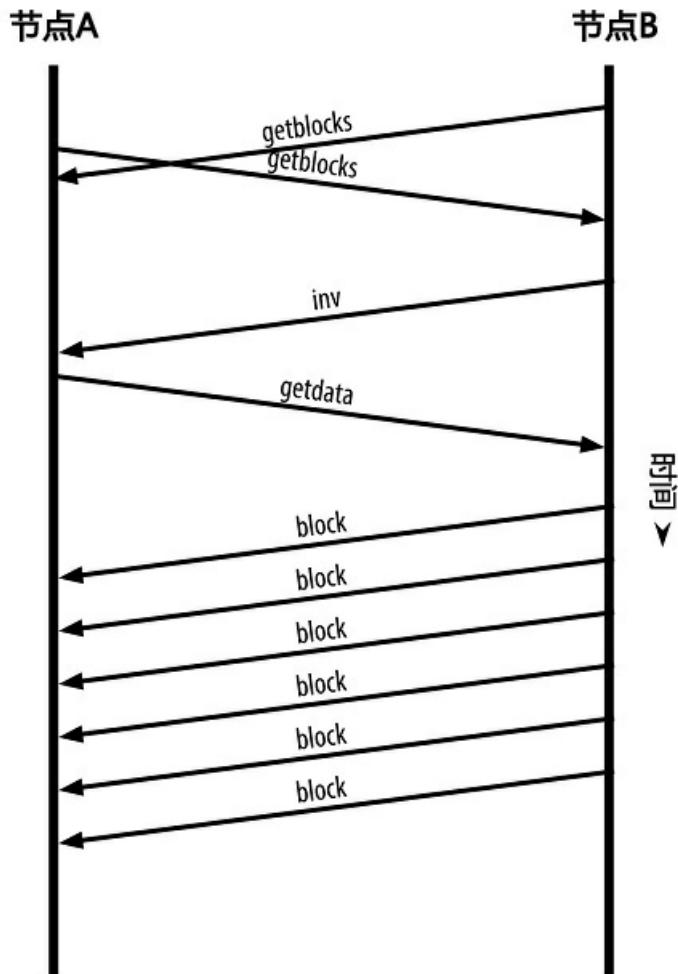






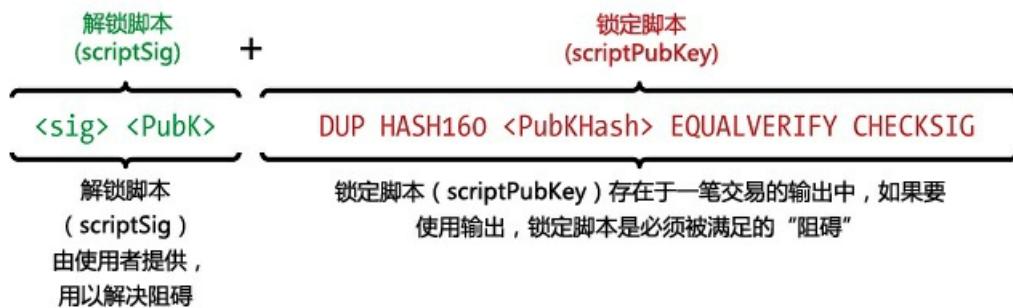
## P2P

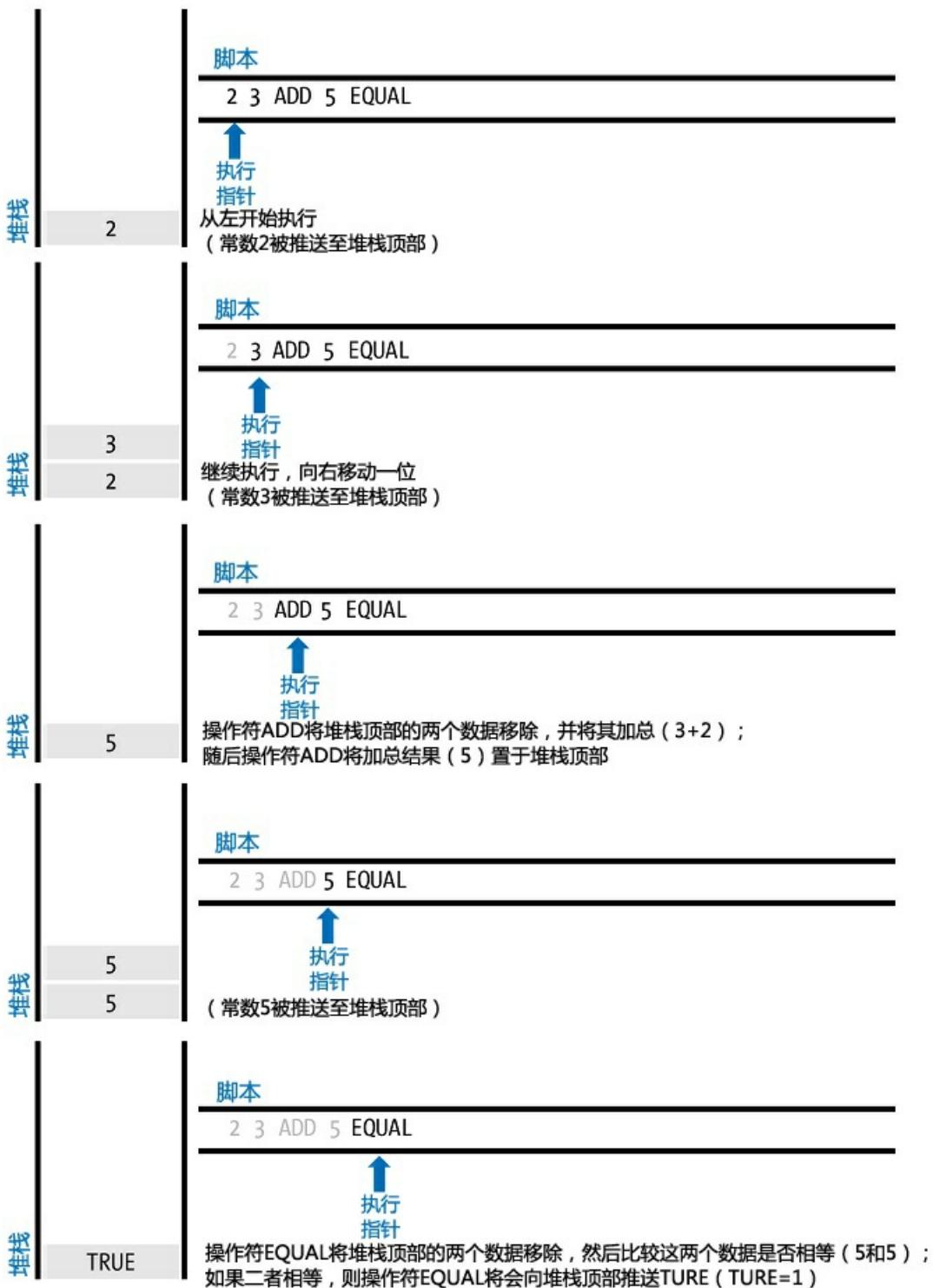


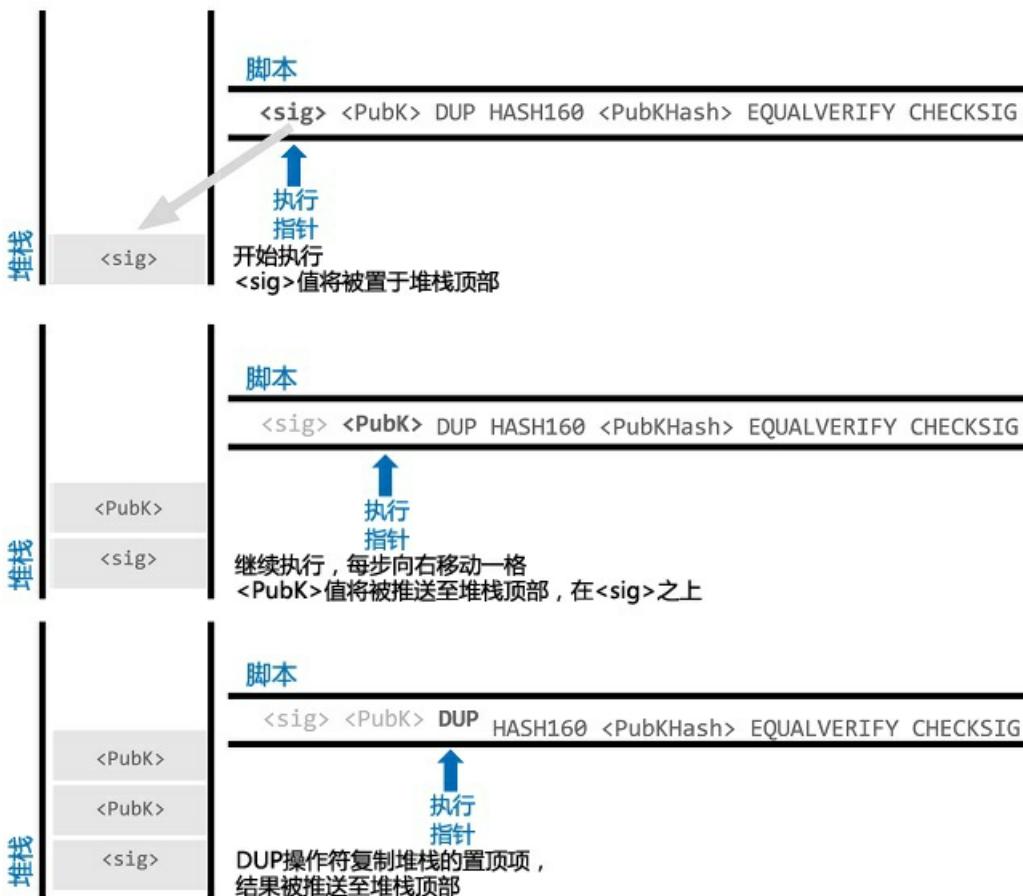




## 脚本

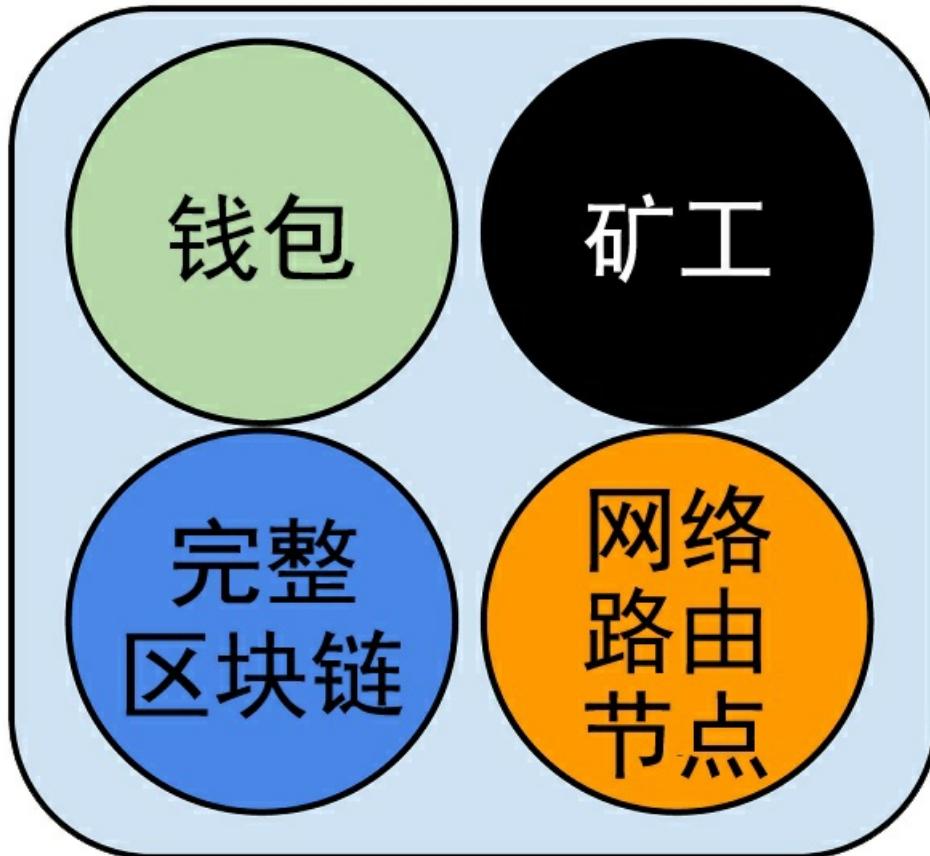






	<p><b>脚本</b></p> <hr/> <pre>&lt;sig&gt; &lt;PubK&gt; DUP HASH160 &lt;PubKHash&gt; EQUALVERIFY CHECKSIG</pre>  <p>HASH160 操作符对堆栈置顶项进行基于 RIPEMD160 (SHA256(PubK)) 的重新表述，&lt;PubKHash&gt;的值被推送至堆栈顶部</p>
	<p><b>脚本</b></p> <hr/> <pre>&lt;sig&gt; &lt;PubK&gt; DUP HASH160 &lt;PubKHash&gt; EQUALVERIFY CHECKSIG</pre>  <p>脚本中&lt;PubKHash&gt;的值将被推送至之前基于 HASH160 所估算出的 PubKHash 值的上方</p>
	<p><b>脚本</b></p> <hr/> <pre>&lt;sig&gt; &lt;PubK&gt; DUP HASH160 &lt;PubKHash&gt; EQUALVERIFY CHECKSIG</pre>  <p>EQUALVERIFY 操作符将 PubKHash 和用户的 PubKHash 对比，如果一致，则都被移除，然后继续执行</p>
	<p><b>脚本</b></p> <hr/> <pre>&lt;sig&gt; &lt;PubK&gt; DUP HASH160 &lt;PubKHash&gt; EQUALVERIFY CHECKSIG</pre>  <p>CHECKSIG 操作符核查签名&lt;sig&gt;是否与公钥的&lt;sig&gt;匹配，如果匹配，则会在顶部显示 TRUE</p>

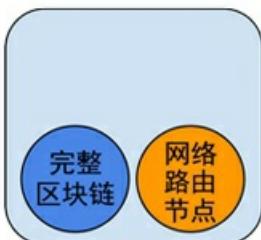
钱包





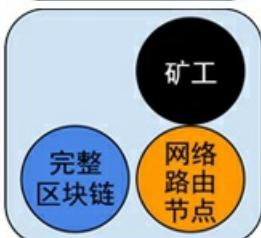
## 核心客户端 (Bitcoin Core)

在比特币P2P网络中，包含钱包、矿工、完整区块链数据库、网络路由节点。



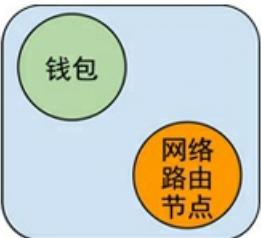
## 完整区块链节点

在比特币P2P网络中，包含完整区块链以及网络路由节点。



## 独立矿工

包含具有完整区块链副本的挖矿功能、以及比特币P2P网络路由节点。



## 轻量(SPV)钱包

包含不具有区块链的钱包以及比特币P2P网络节点。



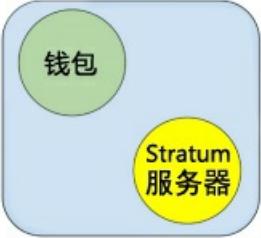
## 矿池协议服务器

将运行其他协议的节点(例如矿池挖矿节点、Stratum节点)，连接至P2P网络的网关路由器。



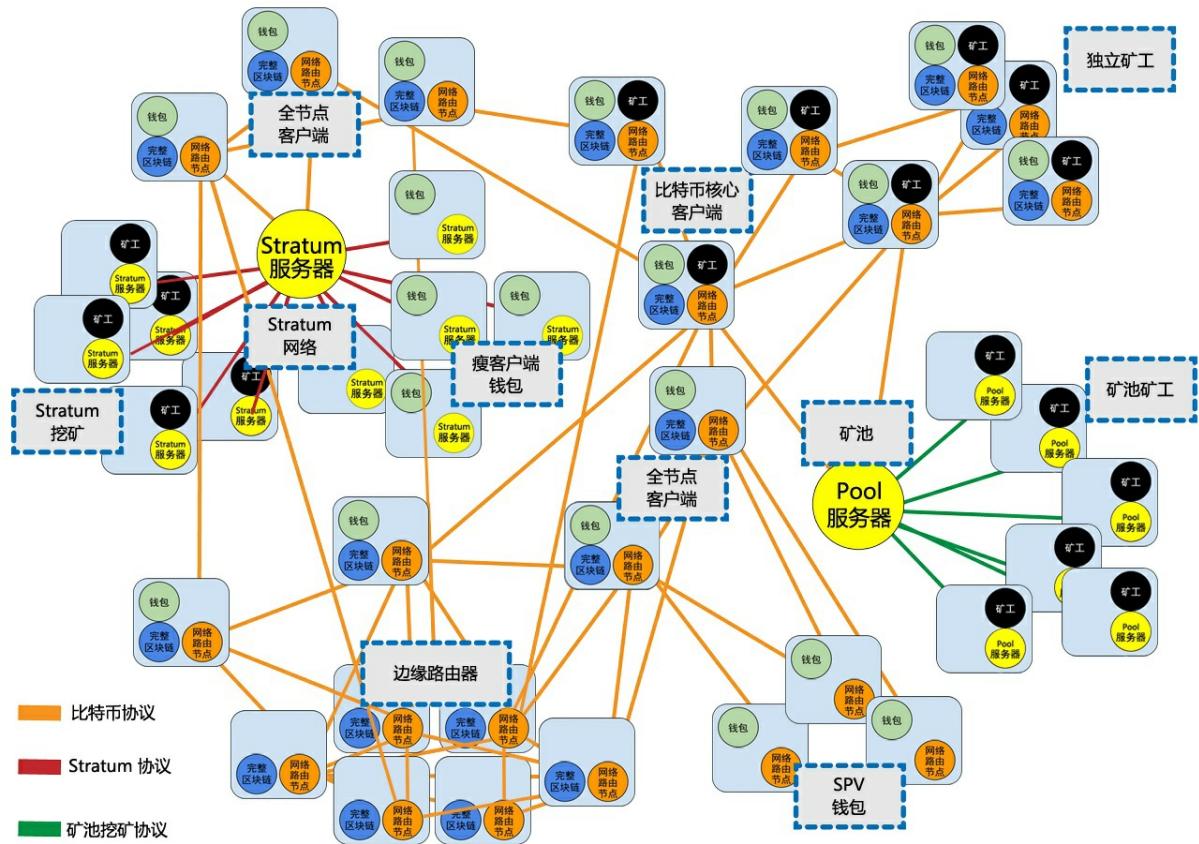
## 挖矿节点

包含不具有区块链、但具备Stratum协议节点 ( S ) 或其他矿池挖矿协议节点 ( P ) 的挖矿功能。



## 轻量(SPV) Stratum 钱包

包含不具有区块链的钱包、运行 Stratum 协议的网络节点。



## 使用

比特币项目的常用命令行。

## 编译(Ubuntu)

### 安装所需环境

```
#构建工具
sudo apt-get install build-essential libtool autotools-dev automake pkg-config bsdmainutils python3

#libssl
#libevent
sudo apt-get install libssl-dev libevent-dev

#Boost
sudo apt-get install libboost-system-dev libboost-filesystem-dev libboost-chrono-dev libboost-program-options-
-dev libboost-test-dev libboost-thread-dev

#Berkeley DB
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:bitcoin/bitcoin
sudo apt-get update
sudo apt-get install libdb4.8-dev libdb4.8++-dev

#GUI/QT
#Payments in GUI
sudo apt-get install libqt5gui5 libqt5core5a libqt5dbus5 qttools5-dev qttools5-dev-tools libprotobuf-dev protobuf-
compiler

#UPnP
#ZMQ notification
#QR codes in GUI
sudo apt-get install libminiupnpc-dev libzmq3-dev libqrencode-dev
```

### 下载源码

```
git clone
```

### 编译

```
./autogen.sh
./configure
make
make install
```

# 私链搭建(Linux)

## 程序下载

```
https://bitcoin.org/en/download
```

## 配置环境

```
#解压  
tar -zxf bitcoin-0.16.0-x86_64-linux-gnu.tar.gz  
  
#创建连个文件夹1和2  
cd bitcoin-0.16.0/bin  
mkdir 1  
mkdir 2  
  
#下载默认配置文https://github.com/bitcoin/bitcoin/blob/master/contrib/debian/examples/bitcoin.conf  
#分别修改对应内容,然后放到1、2文件夹下面
```

```
#修改以下内容  
regtest=1  
server=1  
rpcuser=admin1  
rpcpassword=123  
rpcallowip=0.0.0.0/0  
rpcport=8001  
dnsseed=0  
upnp=0  
port=8000
```

```
#修改以下内容  
regtest=1  
connect=172.16.172.90:8000  
listen=0  
server=1  
rpcuser=admin2  
rpcpassword=123  
rpcallowip=0.0.0.0/0  
rpcport=9001  
dnsseed=0  
upnp=0  
port=9000
```

## 启动服务

```
./bitcoind -daemon -datadir=1  
./bitcoind -daemon -datadir=2
```

## 查询服务

```

./bitcoin-cli -datadir=1 getwalletinfo
{
  "walletname": "wallet.dat",
  "walletversion": 159900,
  "balance": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 0,
  "keypoololdest": 1523884293,
  "keypoolsize": 1000,
  "keypoolsize_hd_internal": 1000,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "3667dcf4ebe66ee13f38ab76d1e87e32580d188c"
}

./bitcoin-cli -datadir=1 getnetworkinfo
{
  "version": 160000,
  "subversion": "/Satoshi:0.16.0/",
  "protocolversion": 70015,
  "localservices": "00000000000040d",
  "localrelay": true,
  "timeoffset": 0,
  "networkactive": true,
  "connections": 1,
  "networks": [
    {
      "name": "ipv4",
      "limited": false,
      "reachable": true,
      "proxy": "",
      "proxy_randomize_credentials": false
    },
    {
      "name": "ipv6",
      "limited": false,
      "reachable": true,
      "proxy": "",
      "proxy_randomize_credentials": false
    },
    {
      "name": "onion",
      "limited": true,
      "reachable": false,
      "proxy": "",
      "proxy_randomize_credentials": false
    }
  ],
  "relayfee": 0.00001000,
  "incrementalfee": 0.00001000,
  "localaddresses": [
  ],
  "warnings": ""
}

./bitcoin-cli -datadir=1 getblockchaininfo
{
  "chain": "regtest",
  "blocks": 0,
  "headers": 0,

```





```

        }
    },
    {
        "id": "bip66",
        "version": 3,
        "reject": {
            "status": false
        }
    },
    {
        "id": "bip65",
        "version": 4,
        "reject": {
            "status": false
        }
    }
],
"bip9_softforks": {
    "csv": {
        "status": "defined",
        "startTime": 0,
        "timeout": 9223372036854775807,
        "since": 0
    },
    "segwit": {
        "status": "active",
        "startTime": -1,
        "timeout": 9223372036854775807,
        "since": 0
    }
},
"warnings": ""
}

```

## 产生200个区块

```

./bitcoin-cli -datadir=1 generate 200
[
    "10f2e86ef528fae6f21f57f3136e3d6b8c07c86a3c044376c613855d8a080dd8",
    "5621bbd0a9748a11998dc2c490b616331464ce61af3de7e7d45f6ecf4b0a5bc8",
]

#节点1里面有5000个币了
./bitcoin-cli --datadir=1 getwalletinfo
{
    "walletname": "wallet.dat",
    "walletversion": 159900,
    "balance": 5000.00000000,
    "unconfirmed_balance": 0.00000000,
    "immature_balance": 3725.00000000,
    "txcount": 200,
    "keypoololdest": 1523884293,
    "keypoolsiz": 999,
    "keypoolsiz_hd_internal": 1000,
    "paytxfee": 0.00000000,
    "hdmasterkeyid": "3667dcf4ebe66ee13f38ab76d1e87e32580d188c"
}

```

## 获取钱包地址

```
./bitcoin-cli --datadir=1 getaccountaddress "2NFZnKjdTEBnkT5LS74eucKVATP76xR3hVm"
./bitcoin-cli --datadir=2 getaccountaddress "2MxcGwmqT73NrA6NM6S43RcZPPYbkBLdyCt"
```

转账

```

"pruned": false,
"softforks": [
  {
    "id": "bip34",
    "version": 2,
    "reject": {
      "status": false
    }
  },
  {
    "id": "bip66",
    "version": 3,
    "reject": {
      "status": false
    }
  },
  {
    "id": "bip65",
    "version": 4,
    "reject": {
      "status": false
    }
  }
],
"bip9_softforks": {
  "csv": {
    "status": "started",
    "bit": 0,
    "startTime": 0,
    "timeout": 9223372036854775807,
    "since": 144,
    "statistics": {
      "period": 144,
      "threshold": 108,
      "elapsed": 67,
      "count": 67,
      "possible": true
    }
  },
  "segwit": {
    "status": "active",
    "startTime": -1,
    "timeout": 9223372036854775807,
    "since": 0
  }
},
"warnings": ""
}

```

## 私链搭建(Docker)

### 下载镜像并运行

```
#下载
docker pull freewil/bitcoin-testnet-box

#运行容器
sudo docker run --name mybitcoin --user=root -t -i -p 19001:19001
-p 19011:19011 freewil/bitcoin-testnet-box

#查看日志
docker logs -f mybitcoin
```

### 开启两个节点

```
bitcoind -daemon -datadir=1

bitcoin-cli -datadir=1 getinfo
{
    "version": 130200,
    "protocolversion": 70015,
    "walletversion": 130000,
    "balance": 0.00000000,
    "blocks": 0,
    "timeoffset": 0,
    "connections": 1,
    "proxy": "",
    "difficulty": 4.656542373906925e-10,
    "testnet": false,
    "keypoololdest": 1520921618,
    "keypoolsize": 100,
    "paytxfee": 0.00000000,
    "relayfee": 0.00001000,
    "errors": ""
}

bitcoind -daemon -datadir=2
bitcoin-cli -datadir=2 getinfo
{
    "version": 130200,
    "protocolversion": 70015,
    "walletversion": 130000,
    "balance": 0.00000000,
    "blocks": 0,
    "timeoffset": 0,
    "connections": 1,
    "proxy": "",
    "difficulty": 4.656542373906925e-10,
    "testnet": false,
    "keypoololdest": 1520921618,
    "keypoolsize": 100,
    "paytxfee": 0.00000000,
    "relayfee": 0.00001000,
    "errors": ""
}
```

```
#根据balance可以看到，两方的比特币都是0
```

## 模拟挖矿

```
#产生200个区块
bitcoin-cli -datadir=1 generate 200
[
  "0df24dddfb187b55a08f0e4470e3b51a5994699da7456a4061050c29cf99670f",
  "788ea259946064cef2d5d2fa48fd3285868f4e64bd96daa2dc0b9207a22be3b3",
  "054e4e77e8e05cba985e5006bf4fe75b22759716dd586a1f541b892a278f49fa",
  ...
]

#查看信息
bitcoin-cli -datadir=1 getinfo
{
  "version": 130200,
  "protocolversion": 70015,
  "walletversion": 130000,
  "balance": 5000.00000000,
  "blocks": 200,
  "timeoffset": 0,
  "connections": 1,
  "proxy": "",
  "difficulty": 4.656542373906925e-10,
  "testnet": false,
  "keypoololdest": 1520921618,
  "keypoolsiz": 100,
  "paytxfee": 0.00000000,
  "relayfee": 0.00001000,
  "errors": ""
}

#查看账户余额
bitcoin-cli --datadir=1 listaccounts
{
  "": 5000.00000000
}

#查看账户地址
bitcoin-cli --datadir=1 getaddressesbyaccount ""
[
  "mwfkSxRsPjAvxWhTh8dzKtWy7z8UiKMwzd"
]

#查看账户余额
bitcoin-cli --datadir=2 listaccounts
{
  "": 0.00000000
}

#查看账户地址
bitcoin-cli --datadir=2 getaddressesbyaccount ""
[
  "muwmDp87VSSoE7Sy4WryYtgQ8ZUbh7Lret"
]
```

## 转账10个比特币

```
#转账10个比特币
bitcoin-cli -datadir=1 sendtoaddress "muwmDp87VSSoE7Sy4WryYtgQ8ZUbh7Lret"" 10

#生成10个区块
bitcoin-cli -datadir=1 generate 10
```

## 查看转账结果

```
bitcoin-cli -datadir=2 getinfo
{
    "version": 130200,
    "protocolversion": 70015,
    "walletversion": 130000,
    "balance": 10.00000000,
    "blocks": 210,
    "timeoffset": 0,
    "connections": 1,
    "proxy": "",
    "difficulty": 4.656542373906925e-10,
    "testnet": false,
    "keypoololdest": 1520921618,
    "keypoolsize": 100,
    "paytxfee": 0.00000000,
    "relayfee": 0.00001000,
    "errors": ""
}
```

## 利用curl进行操作

```
#用curl发起getinfo请求
curl --data-binary '{"jsonrpc":"1.0","id":"curltext","method":"getinfo","params":[]}' -H 'content-type:text/plain;' http://admin1:123@127.0.0.1:19000

#查看结果
{
    "result": {
        "version": 130200,
        "protocolversion": 70015,
        "walletversion": 130000,
        "balance": 5979.99991660,
        "blocks": 220,
        "timeoffset": 0,
        "connections": 1,
        "proxy": "",
        "difficulty": 4.656542373906925e-10,
        "testnet": false,
        "keypoololdest": 1520926587,
        "keypoolsize": 100,
        "paytxfee": 0.00000000,
        "relayfee": 0.00001000,
        "errors": ""
    },
    "error": null,
    "id": "curltext"
}
```



## Ethereum

项目	内容
链名称	Ethereum
链类型	公链/私链
匿名性	匿名或私有
共识机制	PoW: Proof of Work
开发语言	Go/CPP/Python
合约语言	Solidity/Serpent/LLL
官方网站	<a href="http://ethereum.org/">http://ethereum.org/</a>
白皮书	<a href="https://github.com/ethereum/wiki/wiki/White-Paper">https://github.com/ethereum/wiki/wiki/White-Paper</a>
项目地址	<a href="https://github.com/ethereum/go-ethereum">https://github.com/ethereum/go-ethereum</a>

## 项目特点

以太坊的创始人看到了比特币的不足之处，同时区块链世界正发生分裂，出现了各种各样基于比特币的项目。因此他们创建了以太坊，用来解决这些问题。

以太坊最有特色的地方，是它引进了以太坊虚拟机EVM以及智能合约（还有Gas），并提供了友善的开发环境。通过图灵完备的EVM，以太坊向他的用户提供了无限可能，换句话说，以太坊可以算是一个真正的世界计算机。

以太坊采用了内存需求较高的哈希函数，并启用了叔块激励机制，进一步增加了安全性及公平性。以太坊有自己的账号系统，而不仅仅是UTXO。

同时，以太坊积极推进了以自己为主链的生态圈，通过以太币、发行代币、众筹、DAPP、DAO等导致了区块链技术的第二次大爆发。

## 整体架构

### 以太坊架构



图2-14 以太坊总体架构

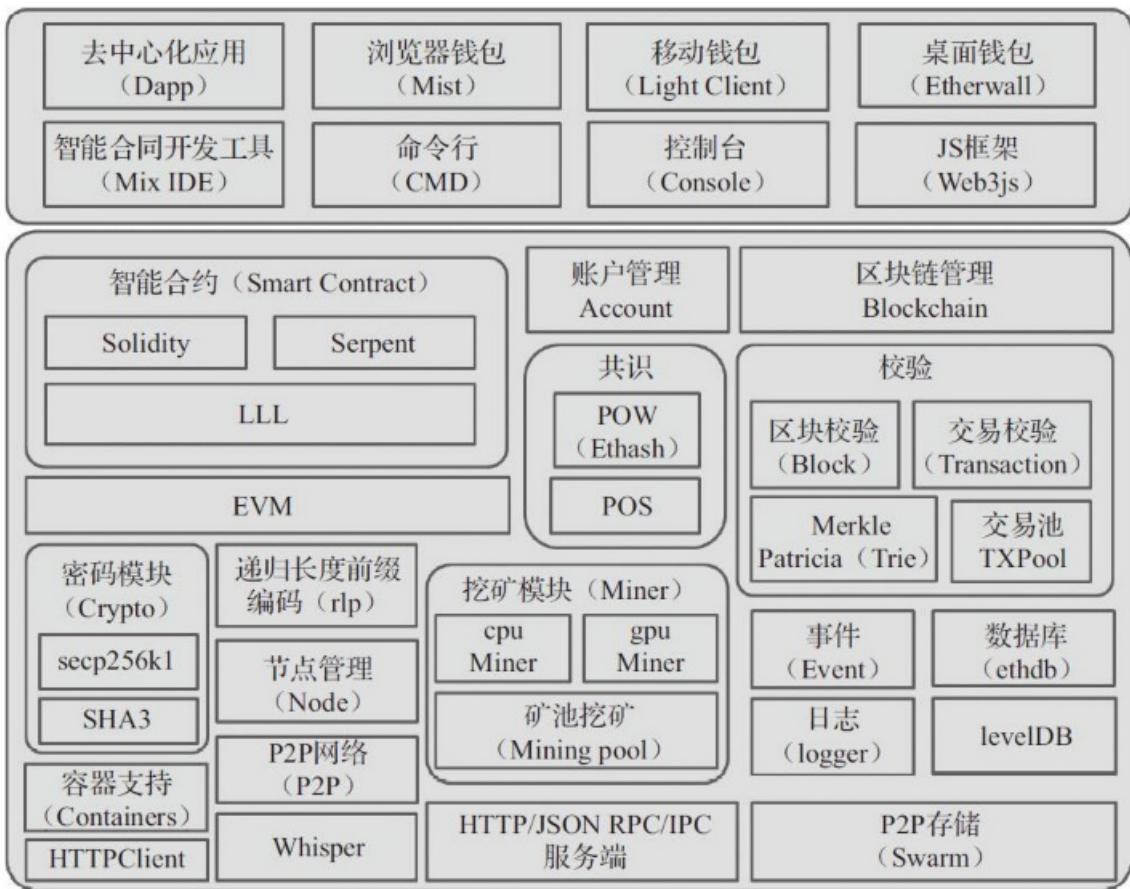


图3-9 以太坊架构

## 状态转换

### 账户

在以太坊系统中，状态是由被称为“账户”（每个账户由一个 20 字节的地址）的对象和在两个账户之间转移价值和信息的状态转换构成的。以太坊的账户包含四个部分：

随机数，用于确定每笔交易只能被处理一次的计数器

账户目前的以太币余额

账户的合约代码，如果有的话

账户的存储（默认为空）

以太坊账户分为两种类型：合约账户（Contracts Accounts）和外部账户（Externally Owned Accounts，或 EOA）。

合约账户：存储执行的智能合约代码，只能被外部账户来调用激活；

外部账户：以太币拥有者账户，对应到某公钥。账户包括 nonce、balance、storageRoot、codeHash 等字段，由个人来控制。

当合约账户被调用时，存储其中的智能合约会在矿工处的虚拟机中自动执行，并消耗一定的燃料。燃料通过外部账户中的以太币进行购买。

## 消息和交易

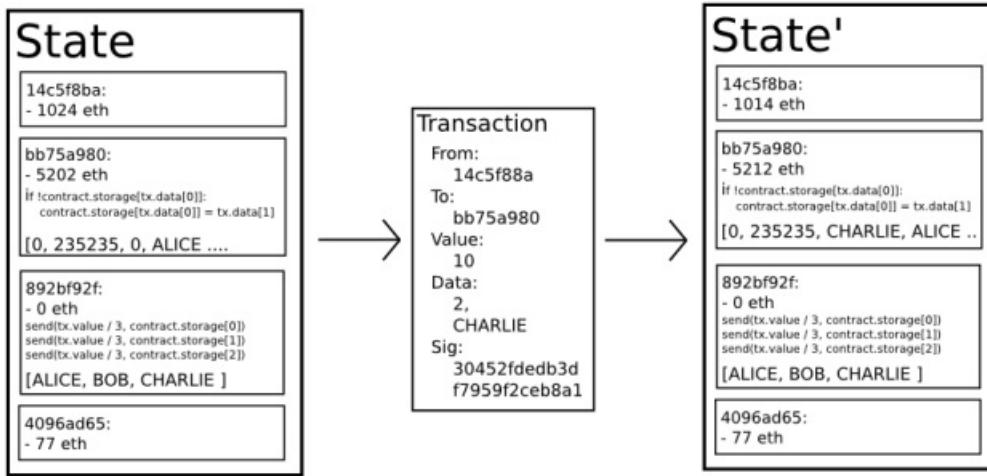
以太坊的消息在某种程度上类似于比特币的交易，但是两者之间存在三点重要的不同。第一，以太坊的消息可以由外部实体或者合约创建，然而比特币的交易只能从外部创建。第二，以太坊消息可以选择包含数据。第三，如果以太坊消息的接受者是合约账户，可以选择进行回应，这意味着以太坊消息也包含函数概念。

以太坊中“交易”是指存储从外部账户发出的消息的签名数据包。交易包含消息的接收者、用于确认发送者的签名、以太币账户余额、要发送的数据和两个被称为 STARTGAS 和 GASPRICE 的数值。为了防止代码的指数型爆炸和无限循环，每笔交易需要对执行代码所引发的计算步骤-包括初始消息和所有执行中引发的消息-做出限制。STARTGAS 就是限制，

GASPRICE 是每一计算步骤需要支付矿工的费用。如果执行交易的过程中，“用完了瓦斯”，所有的状态改变恢复原状态，但是已经支付的交易费用不可收回了。如果执行交易中止时还剩余瓦斯，那么这些瓦斯将退还给发送者。创建合约有单独的交易类型和相应的消息类型；合约的地址是基于账号随机数和交易数据的哈希计算出来的。

消息机制的一个重要后果是以太坊的“头等公民”财产-合约与外部账户拥有同样权利，包括发送消息和创建其它合约的权利。这使得合约可以同时充当多个不同的角色，例如，用户可以使去中心化组织（一个合约）的一个成员成为一个中介账户（另一个合约），为一个偏执的使用定制的基于量子证明的兰波特签名（第三个合约）的个人和一个自身使用由五个私钥保证安全的账户（第四个合约）的共同签名实体提供居间服务。以太坊平台的强大之处在于去中心化的组织和代理合约不需要关心合约的每一参与方是什么类型的账户。

### 状态转换



以太坊的状态转换函数:  $\text{APPLY}(S, TX) \rightarrow S'$ , 可以定义如下:

1. 检查交易的格式是否正确(即有正确数值)、签名是否有效和随机数是否与发送者账户的随机数匹配。如否, 返回错误。
2. 计算交易费用:  $\text{fee} = \text{STARTGAS} * \text{GASPRICE}$ , 并从签名中确定发送者的地址。从发送者的账户中减去交易费用和增加发送者的随机数。如果账户余额不足, 返回错误。
3. 设定初值  $\text{GAS} = \text{STARTGAS}$ , 并根据交易中的字节数减去一定量的瓦斯值。
4. 从发送者的账户转移价值到接收者账户。如果接收账户还不存在, 创建此账户。如果接收账户是一个合约, 运行合约的代码, 直到代码运行结束或者瓦斯用完。
5. 如果因为发送者账户没有足够的钱或者代码执行耗尽瓦斯导致价值转移失败, 恢复原来的状态, 但是还需要支付交易费用, 交易费用加至矿工账户。
6. 否则, 将所有剩余的瓦斯归还给发送者, 消耗掉的瓦斯作为交易费用发送给矿工。例如, 假设合约的代码如下:

```
if !self.storage[calldataload(0)]:self.storage[calldataload(0)] = calldataload(32)
```

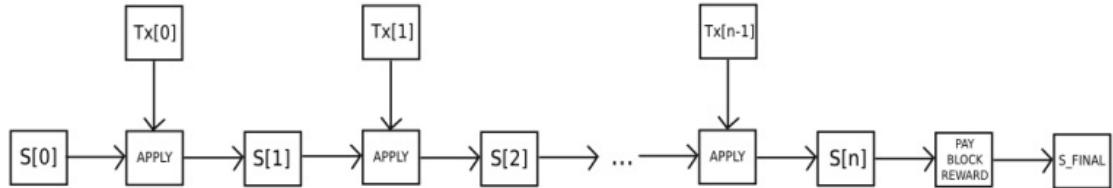
需要注意的是, 在现实中合约代码是用底层以太坊虚拟机(EVM)代码写成的。上面的合约是用我们的高级语言 Serpent 语言写成的, 它可以被编译成 EVM 代码。假设合约存储器开始时是空的, 一个值为 10 以太, 瓦斯为 2000, 瓦斯价格为 0.001 以太并且 64 字节数据, 第一个三十二字节的快代表号码 2 和第二个代表词 CHARLIE。的交易发送后, 状态转换函数的处理过程如下:

1. 检查交易是否有效、格式是否正确。
2. 检查交易发送者至少有  $2000 * 0.001 = 2$  个以太币。如果有, 从发送者账户中减去 2 个以太币。
3. 初始设定  $\text{gas}=2000$ , 假设交易长为 170 字节, 每字节的费用是 5, 减去 850, 所以还剩 1150。
4. 从发送者账户减去 10 个以太币, 为合约账户增加 10 个以太币。
5. 运行代码。在这个合约中, 运行代码很简单: 它检查合约存储器索引为 2 处是否已使用, 注意到它未被使用, 然后将其值置为 CHARLIE。假设这消耗了 187 单位的瓦斯, 于是剩余的瓦斯为  $1150 - 187 = 963$ 。
6. 向发送者的账户增加  $963 * 0.001 = 0.963$  个以太币, 返回最终状态。如果没有合约接收交易, 那么所有的交易费用就等于  $\text{GASPRICE}$  乘以交易的字节长度, 交易的数据就与交易费用无关了。另外, 需要注意的是, 合约发起的消息可以对它们产生的计算分配瓦斯限额, 如果子计算的瓦斯用完了, 它只恢复到消息发出时的状态。因此, 就像交易一样, 合约也可以通过对它产生的子计算设置严格的限制, 保护它们的计算资源。



## 区块确认

### 区块确认



虽然有一些不同，但以太坊的区块链在很多方面类似于比特币区块链。它们的区块链架构的不同在于，以太坊区块不仅包含交易记录和最近的状态，还包含区块序号和难度值。以太坊中的区块确认算法如下：

1. 检查区块引用的上一个区块是否存在和有效。
2. 检查区块的时间戳是否比引用的上一个区块大，而且小于 15 分钟。
3. 检查区块序号、难度值、交易根、叔根和瓦斯限额(许多以太坊特有的底层概念)是否有效。
4. 检查区块的工作量证明是否有效。
5. 将  $S[0]$  赋值为上一个区块的 STATE\_ROOT。
6. 将 TX 赋值为区块的交易列表，一共有 n 笔交易。对于属于  $0 \dots n-1$  的 i，进行状态转换  $S[i+1] = \text{APPLY}(S[i], TX[i])$ 。如果任何一个转换发生错误，或者程序执行到此处所花费的瓦斯(gas)超过了 GASLIMIT，返回错误。
7. 用  $S[n]$  给  $S_{\text{FINAL}}$  赋值，向矿工支付区块奖励。
- 8 检查  $S_{\text{FINAL}}$  是否与 STATE\_ROOT 相同。如果相同，区块是有效的。否则，区块是无效的。

# 使用

## 单位

不要被贫穷限制你的想象力ether=1e18 wei。

wei	1 wei
Kwei (babbage)	1e3 wei
Mwei (lovelace)	1e6 wei
Gwei (shannon)	1e9 wei
microether (szabo)	1e12 wei
milliether (finney)	1e15 wei
ether	1e18 wei
Kether	1e21 wei
Mether	1e24 wei
Gether	1e27 wei
Tether	1e30 wei

## 端口列表

端口描述	端口号
服务端口	30303
Bootnode	30301
RPC端口	8545
WS-RPC端口	8546
PPROF端口	6060

## Remix IDE环境搭建(Ubuntu16)

### 编译

```
#建议用root运行, 里面部分代码没有使用sudo权限  
npm install remix-ide -g  
  
#如果安装时出现root没有权限的问题, 请用以下代码运行  
npm install remix-ide -g --unsafe-perm
```

### 运行

```
remix-ide
```

如果出现以下错误

```
Cannot find module './build/Release/scrypt'
```

需要修改一行代码

```
#修改此文件  
vi /usr/lib/node_modules/remix-ide/node_modules/scrypt/index.js  
#找到这一行  
require("./build/Release/scrypt")  
#修改为这一行  
require("scrypt")
```

### 修改监听地址

```
#默认127.0.0.1:8080  
#如果需要改变地址, 需要修改此文件  
vi /usr/bin/remix-ide  
#找到这一行  
server.listen(8080, '127.0.0.1', function () {})  
#修改为这一行  
server.listen(8080, '0.0.0.0', function () {})  
#重启remix-ide
```

# 智能合约编写01

## Hello.sol

```
pragma solidity ^0.4.22;

contract Hello {
    /* Define variable owner of the type address */
    address public owner;

    /* Define variable greeting of the type string */
    string public greeting;

    /* Function to recover the funds on the contract */
    function kill() public {
        if (msg.sender == owner) selfdestruct(owner);
    }

    /* This runs when the contract is executed */
    constructor(string _greeting) public {
        owner = msg.sender;
        greeting = _greeting;
    }

    /* Main function */
    function greet() public constant returns (string) {
        return greeting;
    }
}
```

## 编译合约

```
#编译方法有两种，一种是命令行编译(当前使用)，一种是用Remid IDE编译(后面有介绍)
#从github上面下载solc编译工具
#用下面的命令行，生成abi和bin文件
..../solc -o . --bin --abi Hello.sol

helloAbi01 = JSON.parse('ABI文件的全部内容');
helloBin01 = '0x'+'BIN文件的全部内容';
```

## 开启私链或者测试链

测试链开启，请查看前面的章节

私链开启，请查看后面的章节

本例用了私链

## 部署并使用合约

```
#在BC01开启geth
./geth --datadir=~/eth001 --networkid 1 --bootnodes=enode://f2576f4b3334979404226043bb412e9a1f3c33d2ce9f9894e
1478f9dfde5493a3f3db189393db841b308cbf35dfa4a6dafa63796968eae8721b285c12ff40bae@[172.16.172.81]:30301 console
```



```
1 waiting to be mined...
undefined

#等待合约被挖坑

#在BC02调用合约

var helloAbi01 = JSON.parse('[{"constant":false,"inputs":[],"name":"kill","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"constant":true,"inputs":[],"name":"owner","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability":"view","type":"function"}, {"constant":true,"inputs":[],"name":"greet","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"}, {"constant":true,"inputs":[],"name":"greeting","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"}, {"constant":true,"inputs":[{"name":"_greeting","type":"string"}],"name":"greet","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"constructor"}]');

var helloFactory01 = eth.contract(helloAbi01);
#这个地址就是部署合约时, 返回的合约地址
var myHello01 = helloFactory01.at('0xBA67BbD4cF60C7Ab8277954B9b6445fF9589c3a6')
myHello01.greet();
```

## 智能合约编写02

### MyTransfer.sol

```
pragma solidity ^0.4.22;

contract MyTransfer {
    mapping (address => uint) public balances;

    constructor() public {
        balances[msg.sender] = 1000000;
    }

    function transfer(address _to, uint _amount) public {
        require(balances[msg.sender] < _amount, "you do not have enough money");
        balances[msg.sender] -= _amount;
        balances[_to] += _amount;
    }
}
```

### 编译合约

```
#编译方法有两种, 一种是命令行编译(当前使用), 一种是用Remix IDE编译(后面有介绍)
#从github上面下载solc编译工具
#用下面的命令行, 生成abi和bin文件
./solc -o . --bin --abi MyTransfer.sol

transAbi01 = JSON.parse('ABI文件的全部内容');
transBin01 = '0x'+BIN文件的全部内容';
```

### 开启私链或测试链

测试链开启, 请查看前面的章节

私链开启, 请查看后面的章节

本例用了私链

### 部署并使用合约

```
#在BC01开启geth
./geth --datadir=~/eth001 --networkid 1 --bootnodes=enode://f2576f4b3334979404226043bb412e9a1f3c33d2ce9f9894e
1478f9dfde5493a3f3db189393db841b308cbf35dfa4a6dafa63796968eae8721b285c12ff40bae@[172.16.172.81]:30301 console

#在BC01解锁账号
personal.unlockAccount('c230847d2669091b77053df0cbc7f533a0a4cf1c','patient1',3600)

#在BC01部署合约
var transAbi01 = JSON.parse('[{"constant":true,"inputs":[{"name":"","type":"address"}],"name":"balances","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"}, {"constant":false,"inputs":[{"name":"_to","type":"address"}, {"name":"_amount","type":"uint256"}],"name":"transfer","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"}, {"inputs":[],"payable":false,"stateMutability":"nonpayable","type":"constructor"}]');
```



```
tability":"nonpayable","type":"constructor"}]);  
var transFactory01 = eth.contract(transAbi01);  
#这个地址就是部署合约时, 返回的合约地址  
var mytrans01= transFactory01.at('0x952704a96A017ca3EB292947B91830dB25C48185')  
mytrans01.balances("0xc230847d2669091b77053df0cbc7f533a0a4cf1c")  
mytrans01.balances("0xc9fec0bd5aab52ba08a826af70bdecd2963bf5c9")
```

# 智能合约编写03

## MyToken.sol

```
pragma solidity ^0.4.22;

contract MyToken {

    //pragma solidity

    /*在合约中使用public关键字定义所有能被别的合约访问的变量*/
    string public name;
    string public symbol;
    uint8 public decimals;

    /*建立一个数组存储账户的余额*/
    mapping (address => uint256) public balanceOf;

    /*建立一个公共事件通知*/
    event Transfer( address indexed from, address indexed to, uint256 value);

    /*构造函数*/
    constructor( uint256 _supply, string _name, string _symbol, uint8 _decimals) public {

        /*默认将股权分为10000份, 即股权的最小单位是0.01%*/
        if (_supply == 0) _supply = 10000;

        /*可自定义股权数量和最小单位*/
        balanceOf[ msg.sender ] = _supply;

        /*定义股权名称*/
        name = _name;

        /*设定股权所使用的单位符号, 例如%*/
        symbol = _symbol;

        /*设定小数位数*/
        decimals = _decimals;
    }

    /*创建股权转移函数*/
    function transfer( address _to, uint256 _value) public {
        /*检验是否有足够的股权*/
        require(balanceOf[ msg.sender ] < _value, "not enough share");
        /*不能转移负数的股权*/
        require(balanceOf[_to] + _value < balanceOf[_to], "can not transfer negtive share");

        /*更新股权转让信息*/
        balanceOf[ msg. sender ] -= _value;
        balanceOf[_to] += _value;

        /*通知用户股权转让成功*/
        emit Transfer( msg.sender, _to, _value);
    }
}
```

## 编译合约

```
#编译方法有两种，一种是命令行编译(当前使用)，一种是用Remid IDE编译(后面有介绍)
#从github上面下载solc编译工具
#用下面的命令行，生成abi和bin文件
..../solc -o . --bin --abi MyToken.sol

myTokenAbi01 = JSON.parse('ABI文件的全部内容');
myTokenBin01 = '0x'+BIN文件的全部内容';
```

## 开启私链或测试链

测试链开启, 请查看前面的章节

私链开启, 请查看后面的章节

本例用了私链

部署并使用合约



```

#查看合约内容
mytoken01.name()
mytoken01.symbol()
mytoken01.decimals()
mytoken01.balanceOf(eth.accounts[0])

#在BC01进行转账
mytoken01.transfer.sendTransaction("0xc9fec0bd5aab52ba08a826af70bdecd2963bf5c9", 100, {from: eth.accounts[0]})

#转账结果
INFO [05-01|22:30:45] Submitted transaction fullhash=0x1ae4dc29a945d13ee3823daab0cb4dea60e
74512eb495f9705a370cbd07a3236 recipient=0xa4f0e30a6A6dE938d36779c74921F2b0AF031dA4
"0x1ae4dc29a945d13ee3823daab0cb4dea60e74512eb495f9705a370cbd07a3236"

#等待挖坑

#在BC02查看结果
var myTokenAbi01 = JSON.parse('[
  {"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},
  {"constant":true,"inputs":[],"name":"decimals","outputs":[{"name":"","type":"uint8"}],"payable":false,"stateMutability":"view","type":"function"},
  {"constant":true,"inputs":[{"name":"","type":"address"}],"name":"balanceOf","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},
  {"constant":true,"inputs":[],"name":"symbol","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"},
  {"constant":false,"inputs":[{"name":"_to","type":"address"}, {"name":"_value","type":"uint256"}],"name":"transfer","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},
  {"inputs":[{"name":"_supply","type":"uint256"}],"name":"_name","outputs":[{"name":"_symbol","type":"string"}],"payable":false,"stateMutability":"nonpayable","type":"constructor"},
  {"anonymous":false,"inputs":[{"indexed":true,"name":"to","type":"address"}, {"indexed":false,"name":"value","type":"uint256"}],"name":"Transfer","type":"event"]
]');

var myTokenFactory01 = eth.contract(myTokenAbi01);
#这个地址就是部署合约时, 返回的合约地址
var mytoken01 = myTokenFactory01.at('0xa4f0e30a6A6dE938d36779c74921F2b0AF031dA4')
mytoken01.balanceOf(eth.accounts[0])
mytoken01.balanceOf("0xc230847d2669091b77053df0cbc7f533a0a4cf1c")
mytoken01.balanceOf("0xc9fec0bd5aab52ba08a826af70bdecd2963bf5c9")

```

## 智能合约编写04

MyRating.sol

```
pragma solidity ^0.4.22;

contract MyRating {

    /*存储特定商品的分数*/
    mapping (bytes32 => uint256) public ratings;

    function setRating( bytes32 _key, uint256 _value) public {
        /*为特定编号的商品打分*/
        ratings[_key] = _value;
    }
}
```

编译合约

编译方法有两种，一种是命令行编译(签名有介绍)，一种是用Remid IDE编译(当前使用)

访问<http://localhost:8080>或者官方地址<http://remix.ethereum.org/>

将MyRating.sol中内容拷贝到编辑器

点击Compile

编译成功后，点击details，拷贝web3部署一节，并按自己的需要进行修改。

部署合约



## 编译(MacOS)

### 安装go等必须的软件

```
brew install go
```

### 设置gopath环境变量

```
#新建gopath
mkdir PAHT_TO_GOPATH

#导出环境变量
export gopath=PAHT_TO_GOPATH

#然后在gopath下创建下面的目录
PAHT_TO_GOPATH/src/github.com/ethereum/

#设置必要的代理信息
export http_proxy=xxxx:xx
export https_proxy=xxxx:xx
export no_proxy="localhost, 127.0.0.1, 192.168.99.100, 根据你自己的实际情况进行处理"
```

### 下载源码

```
cd PAHT_TO_GOPATH/src/github.com/ethereum/
git clone https://github.com/ethereum/go-ethereum.git
```

### 编译

```
cd go-ethereum
make geth
```

### 完结

# 私链搭建

## 环境搭建

### 环境说明

虚拟机	作用
BC01	Bootnode, Wallet
BC02	Wallet
BC03	Wallet, Miner

### 下载文件

```
# 下载geth-alltools-linux-amd64-1.8.4-2423ae01.tar.gz  
# 拷贝到各节点  
# 解压
```

### 开启Bootnode

```
#BC01  
. /bootnode --genkey=boot.key  
. /bootnode --nodekey=boot.key 1>bootstrap.log 2>&1 &  
# 记录下面地址，并修改为正确IP  
UDP listener up self=enode://f2576f4b3334979404226043bb412e9a1f3c33d2ce9f9894e1478f9dfde5493a3f3db189393db841  
b308cbf35dfa4a6dafa63796968eae8721b285c12ff40bae@[::]:30301
```

### 新建账号

```
./geth --datadir=~/eth001 --networkid 1 --bootnodes=enode://f2576f4b3334979404226043bb412e9a1f3c33d2ce9f9894e  
1478f9dfde5493a3f3db189393db841b308cbf35dfa4a6dafa63796968eae8721b285c12ff40bae@[172.16.172.81]:30301 account  
new  
  
#BC01  
Password: patient1  
Address: {c230847d2669091b77053df0cbc7f533a0a4cf1c}  
  
#BC02  
Password: patient2  
Address: {c9fec0bd5aab52ba08a826af70bdecd2963bf5c9}  
  
#BC03  
Password: patient3  
Address: {328399ffc312cca5e6955d0dc3986313b68236c8}
```

### 初始文件准备

```
//genesis.json  
// 拷贝到每台机器  
// 修改chainId及nonce确保只有自己认证的节点可以加入  
// 修改alloc给各用户预分配ETH
```

节点初始化

```
#每个节点  
.geth --datadir=~/eth001 init genesis.json
```

基本操作

## 基本查询操作

```
#BC01
./geth --datadir=~/eth001 --networkid 1 --bootnodes=enode://f2576f4b3334979404226043bb412e9a1f3c33d2ce9f9894e
1478f9fdfde5493a3f3db189393db841b308cbf35dfa4a6dafa63796968eae8721b285c12ff40bae@[172.16.172.81]:30301 console

#获取区块链高度
eth.blockNumber

#获取区块内容
eth.getBlock(0)

#换算单位
web3.toWei(1, "ether")
```

转账操作

```
#BC01
./geth --datadir=~/eth001 --networkid 1 --bootnodes=enode://f2576f4b3334979404226043bb412e9a1f3c33d2ce9f9894e
1478f9dfde5493a3f3db189393db841b308cbf35dfa4a6dafa63796968eae8721b285c12ff40bae@[172.16.172.81]:30301 console

#查看BC01账号余额
web3.fromWei(eth.getBalance(eth.coinbase))
web3.fromWei(eth.getBalance('c230847d2669091b77053df0cbc7f533a0a4cf1c'))
1000
```

```
#查看BC02账号余额
web3.fromWei(eth.getBalance('c9fec0bd5aab52ba08a826af70bdecd2963bf5c9'))
1000

#解锁BC01账号
personal.unlockAccount('c230847d2669091b77053df0cbc7f533a0a4cf1c','patient1',3600)

#向账号BC02发起转账
eth.sendTransaction({from: 'c230847d2669091b77053df0cbc7f533a0a4cf1c', to: 'c9fec0bd5aab52ba08a826af70bdecd2963bf5c9', value: web3.toWei(1)})

INFO [04-23|16:00:57] Submitted transaction fullhash=0xb14ee153be478e1ba208a30eae4a4d3cc34
a8b21116d1b17e627cb53ff6a82e6 recipient=0xC9fEC0bd5AAB52bA08a826Af70BdeCD2963BF5c9
```

## 挖矿确认交易

## 查询交易结果

```
#BC01或BC02
./geth --datadir=~/eth001 --networkid 1 --bootnodes=enode://f2576f4b3334979404226043bb412e9a1f3c33d2ce9f9894e
1478f9dfde5493a3f3db189393db841b308cbf35dfa4a6dafa63796968eae8721b285c12ff40bae@[172.16.172.81]:30301 console

web3.fromWei(eth.getBalance('c230847d2669091b77053df0cbc7f533a0a4cf1c'))
998.999622
#虽然只转账了1ETH, 但是交易需要燃烧gas, 所以余额小于999

web3.fromWei(eth.getBalance('c9fec0bd5aab52ba08a826af70bdecd2963bf5c9'))
1001
#收到了转账
```

## EOS

项目	内容
链名称	EOS
链类型	公链
匿名性	匿名
共识机制	BFT-DPOS: Byzantine Fault Tolerance - Delegated Proof of Stake
开发语言	CPP
合约语言	CPP/Web Assembly
官方网站	<a href="https://eos.io/">https://eos.io/</a>
白皮书	<a href="https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md">https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md</a>
项目地址	<a href="https://github.com/EOSIO/eos">https://github.com/EOSIO/eos</a>

### 项目特点：

EOS通过全球选出21个矿工来达成共识，这个机制大大提高了共识达成的速度，但也在被人诟病其过于集中化。

EOS引入了角色和权限的概念；

EOS有一定的密钥恢复机制；

EOS后续会有存储相关的模块，类似于Filecoin，可以接入多种存储，但还没有放出。

EOS与以太坊之间的定位有很多重叠的地方，当前看下来，EOS在共识机制方面，是优于以太坊1.0的，但合约开发环境等支持等相对于以太坊来说还有较大差距。

## 整体架构

**EOS**架构没找到合适的图。。。

## 使用

EOS常用命令行等资料

## 智能合约使用(单机单节点单钱包)

### 账号列表

账号ID	账号含义	钱包
eosio	矿工1	-
token1	交易所1	default
hospital1	医院1	default
patient1	患者1	default

### 开启挖坑

```
nodeos -e -p eosio --plugin eosio::wallet_api_plugin --plugin eosio::chain_api_plugin --plugin eosio::account_history_api_plugin
```

### 初始化默认钱包

```
cleos wallet create  
#记住这个密钥, 用来处理钱包操作  
"PW5JAj9qw2tQsuVajLVXsiKUD7ULQnZ3qxpab5YVFcs6qbhQ84J1"
```

### 创建账号

```
#创建密钥  
cleos create key  
Private key: 5JJhx7BsRMKNnVcm4gSg7TCkK3B5Dwn2iqjUS6NoSRkJQyRPsxn  
Public key: EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms  
  
#导入密钥  
cleos wallet import 5JJhx7BsRMKNnVcm4gSg7TCkK3B5Dwn2iqjUS6NoSRkJQyRPsxn  
  
#创建三个账号hospital1, patient1, token1  
cleos create account eosio hospital1 EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms  
cleos create account eosio patient1 EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms  
cleos create account eosio token1 EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms  
  
#查看账号列表  
cleos get accounts EOS5jXAL8pkuAb7gdkLT3dmqudCRXhMhAxfEkiFqsaM2yKK9CSzms
```

### 加载并执行智能合约

```
#加载智能合约  
cleos set contract token1 ..../contracts/eosio.token -p token1  
  
#发YR币, 最多10000个  
cleos push action token1 create '{"issuer":"token1", "maximum_supply":"10000.0000 YR", "can_freeze":0, "can_r
```

```
ecall":0, "can_whitelist":0}' -p token1

#将200个币发给用户patient1
cleos push action token1 issue '["patient1", "100.0000 YR", "memo"]' -p token1
cleos push action token1 issue '["patient1", "100.0000 YR", "memo"]' -p token1 -d -j

#patient1用户向hospital1用户转账50
cleos push action token1 transfer '["patient1", "hospital1", "50.0000 YR", "One USB disk with CT Images" ]' -
p patient1
```

## 查看交易信息

```
#查看用户交易列表
cleos get transactions patient1

#查看某次交易的细节
cleos get transaction f217ea4d293a026ed010c527b84d2133c7c335f43234bb58a5c87c911c933b9e
```

## 查看智能合约

```
#查看合约id
cleos get code token1

#获取合约描述
cleos get code -a token1.abi token1
```

## 智能合约使用(单机单节点多钱包)

本节延续上一节内容

### 账号列表

账号ID	账号含义	钱包
eosio	矿工1	-
token1	交易所1	default
hospital1	医院1	default
patient1	患者1	default
hospital2	医院2	hospital2
doctor2	医生2	hospital2

### 创建钱包及用户

```
#创建钱包
cleos wallet create -n hospital1
"PW5JxnbBTTbTQKUHHNvYojU1xrcjxQduzzwU6gfMEgiZjwT5S6xJ6"

#查看钱包列表
cleos wallet list

#创建并导入key
cleos create key
Private key: 5KZS8Q9MVniCbLY2RxjSWruM5YvfDPteP3qQDM23YP5ngwvqHMt
Public key: EOS8K9wqRtD8cCuHvbzdcquEhYHnSaFvyQSwm8kVxYFHavraNeue2

cleos wallet import -n hospital2 5KZS8Q9MVniCbLY2RxjSWruM5YvfDPteP3qQDM23YP5ngwvqHMt

#创建用户
cleos create account eosio hospital1 EOS8K9wqRtD8cCuHvbzdcquEhYHnSaFvyQSwm8kVxYFHavraNeue2 EOS8K9wqRtD8cCuHvbzdcquEhYHnSaFvyQSwm8kVxYFHavraNeue2
cleos create account eosio doctor2 EOS8K9wqRtD8cCuHvbzdcquEhYHnSaFvyQSwm8kVxYFHavraNeue2 EOS8K9wqRtD8cCuHvbzdcquEhYHnSaFvyQSwm8kVxYFHavraNeue2

#查看用户列表
cleos get accounts EOS8K9wqRtD8cCuHvbzdcquEhYHnSaFvyQSwm8kVxYFHavraNeue2

#查看eosio下级用户
cleos get servants eosio
```

### 跨钱包转账

```
cleos push action token1 transfer '[{"patient1", "hospital2", "50.0000 YR", "Remote Diagnose"}]' -p patient1
```

钱包锁定后，不能花钱，只能收钱

```
#锁定patient1所在钱包
```

```
cleos wallet lock -n default

#转账会失败
cleos push action token1 transfer '["patient1", "hospital2", "50.0000 YR", "Remote Diagnose" ]' -p patient1

#解锁patient1所在钱包, 锁定hospital2所在钱包
cleos wallet unlock -n default
cleos wallet lock -n hospital2

#转账会成功
cleos push action token1 transfer '["patient1", "hospital2", "50.0000 YR", "Remote Diagnose" ]' -p patient1
```

# 编写简单的智能合约

## 基本步骤

EOS的智能合约，是用CPP语言写的，通过eosio.cpp工具，分别生成wasm编码（执行），和abi文件（描述）

然后通过cleos set contract工具加载到区块链，最后通过cleos push action执行

### Hello.cpp

```
#include <eosiolib/eosio.hpp>
#include <eosiolib/print.hpp>

using namespace eosio;

class nhello : public eosio::contract {
public:
    using contract::contract;

    /// @abi action
    void hi( account_name user ) {
        //限定输入的user必须是执行合约的用户
        require_auth( user );
        print( "Hello, ", name{user} );
    }
};

EOSIO_ABI( nhello, (hi) )
```

## 编译

```
 eosiocpp -o neo.hello.wast neo.hello.cpp
 eosiocpp -g neo.hello.abi neo.hello.cpp
```

## 加载及运行

```
#开启挖坑
nodeos -e -p eosio --plugin eosio::wallet_api_plugin --plugin eosio::chain_api_plugin --plugin eosio::account_history_api_plugin

#打开并解锁钱包
cleos wallet open
cleos wallet unlock
"PW5JAj9qw2tQsuVajLVXsiKUD7ULQnZ83qxpab5YVFcs6qbhQ84J1"

cleos wallet open -n hospital2
cleos wallet unlock -n hospital2
"PW5JxnBTTbTQKUHHNvYojU1xrcjxQduz2wU6gfMEgiZjwT5S6xJ6"

cleos wallet list

#创建运行合约的账号
cleos create account eosio hello1 EOS8K9wqRtD8cCuHvbzdzcquEhYHnSaFvyQSWh8kVxYFHavraNeue2 EOS8K9wqRtD8cCuHvbzdzc
```

```
quEhYHnSaFvyQSWM8kVxYFHavraNeue2

#加载合约
cleos set contract hello1 neo.hello -p hello1

#运行合约
cleos push action hello1 hi '["hello1"]' -p hello1
```

# 编写有数据存储的智能合约

## 如何存储数据

EOS通过table的概念来存储智能合约运行中所需的数据。

数据储存在区块链上。

部署到区块链后，如果有数据存储，修改wasm编码后，会产生原始储存数据不可读的问题

## 我们看下官方例子tic\_tac\_toe

其中

hpp+cpp文件，主要实现了数据结构定义、业务逻辑、权限控制、数据存储

abi文件，定义了合约的types、structs、actions、tables和ricardian\_clauses

### hpp文件

```
#include <eosiolib/eosio.hpp>

<**
 * 圆叉旗：
 * 在一个3*3的棋盘里，一人用x(先手host)，一人用o(后手challenger)，先连成三点一线的获胜。
 * 棋盘如下所示：
 *
 * (0,0)坐标表示左上角
 * (2,2)坐标表示右下角
 *
 * (0,2)
 * (0,0) - | o | x      - = 空
 *           - | x | -
 * (2,0) x | o | o      x = 先手
 * o = 后手
 *
 * 为了让存储变得简单，在本例中我们用一个9个长度的数组，表示上面3*3的矩阵
 * 比如，上面的棋盘存储后就是，[0, 2, 1, 0, 1, 0, 1, 2, 2]
 *
 * 另外，为了让智能合约变的简单，我们对每两对用户A和B，只允许开启两个旗局，一个是A为先手，一个是B为先手
 * 数据存储在先手账户中，并用后手作为key
 *
 * create创建游戏
 * restart重开游戏
 * close关闭游戏
 * move下棋，先手后手交替进行
 */

namespace neo_tictactoe {
    //游戏开启账户，先手
    static const account_name games_account = N(games);
    //合约账户
    static const account_name code_account = N(tic.tac.toe);

    /**
     * @brief 用于保存棋盘信息的结构
     */
    static const uint32_t board_len = 9;
```

```

struct game {
    game() { initialize_board(); }

    game(account_name challenger, account_name host):challenger(challenger), host(host), turn(host) {
        initialize_board();
    }

    account_name challenger; //后手
    account_name host; //先手
    account_name turn; // 谁的回合
    account_name winner = N(none); // 谁赢了 none/ draw/ host/ challenger
    uint8_t board[board_len]; // 棋盘信息

    // 初始化游戏
    void initialize_board() {
        for (uint8_t i = 0; i < board_len ; i++) {
            board[i] = 0;
        }
    }

    // 重置游戏
    void reset_game() {
        initialize_board();
        turn = host;
        winner = N(none);
    }

    auto primary_key() const { return challenger; }

    EOSLIB_SERIALIZE( game, (challenger)(host)(turn)(winner)(board) )
};

struct create {
    account_name challenger;
    account_name host;

    EOSLIB_SERIALIZE( create, (challenger)(host) )
};

struct restart {
    account_name challenger;
    account_name host;
    account_name by; // the account who wants to restart the game

    EOSLIB_SERIALIZE( restart, (challenger)(host)(by) )
};

struct close {
    account_name challenger;
    account_name host;

    EOSLIB_SERIALIZE( close, (challenger)(host) )
};

```

```

};

/**
 * @brief 下棋坐标
 */
struct movement {
    uint32_t    row;
    uint32_t    column;

    EOSLIB_SERIALIZE( movement, (row)(column) )
};

/**
 * @brief 下棋
 */
struct move {
    account_name challenger;
    account_name host;
    account_name by; // the account who wants to make the move
    movement      mvt;

    EOSLIB_SERIALIZE( move, (challenger)(host)(by)(mvt) )
};

/**
 * @brief 新建表, 保存账号和游戏信息
 */
typedef eosio::multi_index< games_account, game> games;
}

```

## cpp文件

```

#include "neo_tictactoe.hpp"

using namespace eosio;
namespace neo_tictactoe {
struct impl {
    /**
     * @brief Check if cell is empty
     * @param cell - value of the cell (should be either 0, 1, or 2)
     * @return true if cell is empty
     */
    bool is_empty_cell(const uint8_t& cell) {
        return cell == 0;
    }

    /**
     * @brief Check for valid movement
     * @detail Movement is considered valid if it is inside the board and done on empty cell
     * @param movement - the movement made by the player
     * @param game - the game on which the movement is being made
     * @return true if movement is valid
     */
    bool is_valid_movement(const movement& mvt, const game& game_for_movement) {
        uint32_t movement_location = mvt.row * 3 + mvt.column;
        bool is_valid = movement_location < board_len && is_empty_cell(game_for_movement.board[movement_location]);
        return is_valid;
    }
}

```

```

/**
 * @brief Get winner of the game
 * @detail Winner of the game is the first player who made three consecutive aligned movement
 * @param game - the game which we want to determine the winner of
 * @return winner of the game (can be either none/ draw/ account name of host/ account name of challenger)
 */
account_name get_winner(const game& current_game) {
    if((current_game.board[0] == current_game.board[4] && current_game.board[4] == current_game.board[8]) ||
    (current_game.board[1] == current_game.board[4] && current_game.board[4] == current_game.board[7]) ||
    (current_game.board[2] == current_game.board[4] && current_game.board[4] == current_game.board[6]) ||
    (current_game.board[3] == current_game.board[4] && current_game.board[4] == current_game.board[5]))
    {
        // - | - | x     x | - | -     - | - | -     - | x | -
        // - | x | -     - | x | -     x | x | x     - | x | -
        // x | - | -     - | - | x     - | - | -     - | x | -
        if (current_game.board[4] == 1) {
            return current_game.host;
        } else if (current_game.board[4] == 2) {
            return current_game.challenger;
        }
    } else if ((current_game.board[0] == current_game.board[1] && current_game.board[1] == current_game.board[2]) ||
               (current_game.board[0] == current_game.board[3] && current_game.board[3] == current_game.board[6])) {
        // x | x | x     x | - | -
        // - | - | -     x | - | -
        // - | - | -     x | - | -
        if (current_game.board[0] == 1) {
            return current_game.host;
        } else if (current_game.board[0] == 2) {
            return current_game.challenger;
        }
    } else if ((current_game.board[2] == current_game.board[5] && current_game.board[5] == current_game.board[8]) ||
               (current_game.board[6] == current_game.board[7] && current_game.board[7] == current_game.board[8])) {
        // - | - | -     - | - | x
        // - | - | -     - | - | x
        // x | x | x     - | - | x
        if (current_game.board[8] == 1) {
            return current_game.host;
        } else if (current_game.board[8] == 2) {
            return current_game.challenger;
        }
    } else {
        bool is_board_full = true;
        for (uint8_t i = 0; i < board_len; i++) {
            if (is_empty_cell(current_game.board[i])) {
                is_board_full = false;
                break;
            }
        }
        if (is_board_full) {
            return N(draw);
        }
    }
}

```

```

        return N(none);
    }

    /**
     * @brief Apply create action
     * @param create - action to be applied
     */
    void on(const create& c) {
        require_auth(c.host);
        eosio_assert(c.challenger != c.host, "challenger shouldn't be the same as host");

        // Check if game already exists
        games existing_host_games(code_account, c.host);
        auto itr = existing_host_games.find( c.challenger );
        eosio_assert(itr == existing_host_games.end(), "game already exists");

        existing_host_games.emplace(c.host, [&]( auto& g ) {
            g.challenger = c.challenger;
            g.host = c.host;
            g.turn = c.host;
        });
    }

    /**
     * @brief Apply restart action
     * @param restart - action to be applied
     */
    void on(const restart& r) {
        require_auth(r.by);

        // Check if game exists
        games existing_host_games(code_account, r.host);
        auto itr = existing_host_games.find( r.challenger );
        eosio_assert(itr != existing_host_games.end(), "game doesn't exists");

        // Check if this game belongs to the action sender
        eosio_assert(r.by == itr->host || r.by == itr->challenger, "this is not your game!");

        // Reset game
        existing_host_games.modify(itr, itr->host, [&]( auto& g ) {
            g.reset_game();
        });
    }

    /**
     * @brief Apply close action
     * @param close - action to be applied
     */
    void on(const close& c) {
        require_auth(c.host);

        // Check if game exists
        games existing_host_games(code_account, c.host);
        auto itr = existing_host_games.find( c.challenger );
        eosio_assert(itr != existing_host_games.end(), "game doesn't exists");

        // Remove game
        existing_host_games.erase(itr);
    }
}

```

```

    * @brief Apply move action
    * @param move - action to be applied
    */
void on(const move& m) {
    require_auth(m.by);

    // Check if game exists
    games existing_host_games(code_account, m.host);
    auto itr = existing_host_games.find( m.challenger );
    eosio_assert(itr != existing_host_games.end(), "game doesn't exists");

    // Check if this game hasn't ended yet
    eosio_assert(itr->winner == N(none), "the game has ended!");
    // Check if this game belongs to the action sender
    eosio_assert(m.by == itr->host || m.by == itr->challenger, "this is not your game!");
    // Check if this is the action sender's turn
    eosio_assert(m.by == itr->turn, "it's not your turn yet!");

    // Check if user makes a valid movement
    eosio_assert(is_valid_movement(m.mvt, *itr), "not a valid movement!");

    // Fill the cell, 1 for host, 2 for challenger
    const auto cell_value = itr->turn == itr->host ? 1 : 2;
    const auto turn = itr->turn == itr->host ? itr->challenger : itr->host;
    existing_host_games.modify(itr, itr->host, [&]( auto& g ) {
        g.board[m.mvt.row * 3 + m.mvt.column] = cell_value;
        g.turn = turn;

        //check to see if we have a winner
        g.winner = get_winner(g);
    });
}

/// The apply method implements the dispatch of events to this contract
void apply( uint64_t receiver, uint64_t code, uint64_t action ) {

    if (code == code_account) {
        if (action == N(create)) {
            impl::on(eosio::unpack_action_data<tic_tac_toe::create>());
        } else if (action == N(restart)) {
            impl::on(eosio::unpack_action_data<tic_tac_toe::restart>());
        } else if (action == N(close)) {
            impl::on(eosio::unpack_action_data<tic_tac_toe::close>());
        } else if (action == N(move)) {
            impl::on(eosio::unpack_action_data<tic_tac_toe::move>());
        }
    }
};

/* 
 * The apply() methods must have C calling convention so that the blockchain can lookup and
 * call these methods.
*/
extern "C" {

    using namespace neo_tictactoe;
    /// The apply method implements the dispatch of events to this contract
}

```

```

    void apply( uint64_t receiver, uint64_t code, uint64_t action ) {
        impl().apply(receiver, code, action);
    }

} // extern "C"

```

## abi文件

```
{
    "types": [],
    "structs": [
        {
            "name": "game",
            "base": "",
            "fields": [
                {"name": "challenger", "type": "account_name"},
                {"name": "host", "type": "account_name"},
                {"name": "turn", "type": "account_name"},
                {"name": "winner", "type": "account_name"},
                {"name": "board", "type": "uint8[]"}
            ]
        },
        {
            "name": "create",
            "base": "",
            "fields": [
                {"name": "challenger", "type": "account_name"},
                {"name": "host", "type": "account_name"}
            ]
        },
        {
            "name": "restart",
            "base": "",
            "fields": [
                {"name": "challenger", "type": "account_name"},
                {"name": "host", "type": "account_name"},
                {"name": "by", "type": "account_name"}
            ]
        },
        {
            "name": "close",
            "base": "",
            "fields": [
                {"name": "challenger", "type": "account_name"},
                {"name": "host", "type": "account_name"}
            ]
        },
        {
            "name": "movement",
            "base": "",
            "fields": [
                {"name": "row", "type": "uint32"},
                {"name": "column", "type": "uint32"}
            ]
        },
        {
            "name": "move",
            "base": "",
            "fields": [
                {"name": "challenger", "type": "account_name"},
                {"name": "host", "type": "account_name"},
                {"name": "by", "type": "account_name"},
                {"name": "mvt", "type": "movement"}
            ]
        }
    ]
}
```

```

        ],
        "actions": [
            {
                "name": "create",
                "type": "create",
                "ricardian_contract": ""
            },
            {
                "name": "restart",
                "type": "restart",
                "ricardian_contract": ""
            },
            {
                "name": "close",
                "type": "close",
                "ricardian_contract": ""
            },
            {
                "name": "move",
                "type": "move",
                "ricardian_contract": ""
            }
        ],
        "tables": [
            {
                "name": "games",
                "type": "game",
                "index_type": "i64",
                "key_names" : ["challenger"],
                "key_types" : ["account_name"]
            }
        ],
        "ricardian_clauses": []
    }
}

```

## 编译并加载合约

```

#编译
eosiocpp -o neo.tictactoe.wast neo.tictactoe.cpp
eosiocpp -g neo.tictactoe.abi neo.tictactoe.hpp

```

```

#开启区块
nodeos -e -p eosio --plugin eosio::wallet_api_plugin --plugin eosio::chain_api_plugin --plugin eosio::account_history_api_plugin

#加载钱包

cleos wallet open
cleos wallet unlock
"PW5JAj9qw2tQsuVajLVXsiKUD7ULQnZ83qxpaB5YVFcs6qbhQ84J1"

cleos wallet open -n hospital2
cleos wallet unlock -n hospital2
"PW5JxnBTB7QKUHHNvYojU1xrcjxQduz2wU6gfMEgiZjwT5S6xJ6"

cleos wallet list

#创建用户
cleos create account eosio tictactoe1 EOS8K9wqRtD8cCuHvbdcquEhYHnSaFvyQSWh8kVxYFHavraNeue2 EOS8K9wqRtD8cCuHvbdcquEhYHnSaFvyQSWh8kVxYFHavraNeue2

#加载合约
cleos set contract tictactoe1 neo.tictactoe -p tictactoe1

```

## 执行合约

```
cleos push action tictactoe1 create '{"challenger":"patient2", "host":"patient1"}' -p patient1

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient1", "mvt":{"row":0, "column":0}}' -p patient1

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient2", "mvt":{"row":0, "column":1}}' -p patient2

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient1", "mvt":{"row":1, "column":1}}' -p patient1

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient2", "mvt":{"row":2, "column":2}}' -p patient2

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient1", "mvt":{"row":0, "column":2}}' -p patient1

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient2", "mvt":{"row":1, "column":0}}' -p patient2

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient1", "mvt":{"row":2, "column":0}}' -p patient1

cleos push action tictactoe1 move '{"challenger":"patient2", "host":"patient1", "by":"patient2", "mvt":{"row":2, "column":0}}' -p patient2
```

## 查看数据

```
cleos get table tictactoe1 patient1 games
{
  "rows": [
    {
      "challenger": "patient2",
      "host": "patient1",
      "turn": "patient2",
      "winner": "patient1",
      "board": [
        1,
        2,
        1,
        2,
        1,
        0,
        1,
        0,
        2
      ]
    }
  ],
  "more": false
}
```



## 编译(Linux)

### 下载源码

```
git clone https://github.com/EOSIO/eos --recursive
```

### 编译

```
cd eos  
./eosio_build.sh
```

### 安装

```
#脚本运行  
cd build  
make install  
  
#或者拷贝下面的几个文件到path中  
cleos eosio-abigen eosio-launcher keosd nodeos
```

### 完结

## 私链搭建(单机多节点)

最终创建两个生产节点的私链

### 开启钱包服务节点

```
#建议只在本地监听, 因为密钥传输是不加密的
keosd --http-server-address 127.0.0.1:8899
#直接输出
2042116ms thread-0  wallet_plugin.cpp:41      plugin_initialize    ] initializing wallet plugin
2042120ms thread-0   http_plugin.cpp:141        plugin_initialize    ] host: 127.0.0.1 port: 8899
2042121ms thread-0   http_plugin.cpp:144        plugin_initialize    ] configured http to listen on 127.0.
.1:8899
2042122ms thread-0   http_plugin.cpp:213        plugin_startup       ] start listening for http requests
2042123ms thread-0   wallet_api_plugin.cpp:70   plugin_startup       ] starting wallet_api_plugin
2042127ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/create
2042127ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/get_public_
keys
2042127ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/import_key
2042127ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/list_keys
2042127ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/list_wallet_
s
2042127ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/lock
2042127ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/lock_all
2042128ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/open
2042128ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/set_timeout
2042128ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/sign_transa_
ction
2042128ms thread-0   http_plugin.cpp:242        add_handler          ] add api url: /v1/wallet/unlock\
#后输出的
2167622ms thread-0   wallet.cpp:182           save_wallet_file    ] saving wallet to file /home/hiup/eo
sio-wallet./default.wallet
3194752ms thread-0   wallet.cpp:182           save_wallet_file    ] saving wallet to file /home/hiup/eo
sio-wallet./default.wallet
```

### 创建一个默认钱包

```
cleos --wallet-port 8899  wallet create
#记住输出的私钥
"PW5JMmtnJxaUiEwwSLVSrA9UTxSzypnxVkhRtKA1QeSAQm25LU8Dx"
```

### 创建第一个生产节点

```
#第一个节点叫eosio
nodeos --enable-stale-production --producer-name eosio --plugin eosio::chain_api_plugin --plugin eosio::net_a
pi_plugin
#不停输出区块信息
eosio generated block 36cd7280... #7030 @ 2018-04-16T03:36:09.000 with 0 trxs, lib: 7029
eosio generated block a135dfcc... #7031 @ 2018-04-16T03:36:09.500 with 0 trxs, lib: 7030
eosio generated block 6b6e8f9d... #7032 @ 2018-04-16T03:36:10.000 with 0 trxs, lib: 7031
eosio generated block 26f1a273... #7033 @ 2018-04-16T03:36:10.500 with 0 trxs, lib: 7032
eosio generated block 797bdc13... #7034 @ 2018-04-16T03:36:11.000 with 0 trxs, lib: 7033
```

## 创建第二个生产节点

```
#加载智能合约
cleos --wallet-port 8899 set contract eosio ..eos/build/contracts/eosio.bios
Reading WAST/WASM from ..eos/build/contracts/eosio.bios/eosio.bios.wast...
Assembling WASM...
Publishing contract...
executed transaction: 684d83f376333d381841d79931b0c820698a3f2cadb11080169bc5504d1eff8b 3280 bytes 2200576 cycles
#           eosio <= eosio::setcode {"account":"eosio","vmtype":0,"vmversion":0,"code":"0061736d010000001581060037f7e7f0060057f7e7e7e7e...
#           eosio <= eosio::setabi {"account":"eosio","abi":{"types":[],"structs":[{"name":"set_account_limits","base":"","fields":[{"n...
#创建密钥对
cleos create key
Private key: 5JaN9LSpwuqY89KbtijHQDdngQztDyB9XZhZ2KC6QxP7xn9NUeN
Public key: EOS5BTsWP7hC7wAeUuoo6Q3LBTjiPLoEAM6hY51ujDEzsq7UNfex2

#导入密钥对到钱包
cleos --wallet-port 8899 wallet import 5JaN9LSpwuqY89KbtijHQDdngQztDyB9XZhZ2KC6QxP7xn9NUeN

#创建新用户test2
./cleos --wallet-port 8899 create account eosio test2 EOS5BTsWP7hC7wAeUuoo6Q3LBTjiPLoEAM6hY51ujDEzsq7UNfex2 EOS5BTsWP7hC7wAeUuoo6Q3LBTjiPLoEAM6hY51ujDEzsq7UNfex2
executed transaction: 431d872ad07764b912c1eca32e50e01bd0dac26113f6545054b453a1b5330daf 352 bytes 102400 cycles
#           eosio <= eosio::newaccount {"creator":"eosio","name":"test2","owner":{"threshold":1,"key_s": [{"key": "EOS5BTsWP7hC7wAeUuoo6Q3LBTj...
#开启用户test2的生产
#避免了端口冲突
./nodeos --producer-name test2 --plugin eosio::chain_api_plugin --plugin eosio::net_api_plugin --http-server-address 127.0.0.1:8889 --p2p-listen-endpoint 127.0.0.1:9877 --p2p-peer-address 127.0.0.1:9876 --config-dir node2 --data-dir node2 --private-key ["\\"EOS5BTsWP7hC7wAeUuoo6Q3LBTjiPLoEAM6hY51ujDEzsq7UNfex2\\",\\"5JaN9LSpwuqY89KbtijHQDdngQztDyB9XZhZ2KC6QxP7xn9NUeN\\"]"]
test2 generated block b6fe0ecd... #7978 @ 2018-04-16T03:44:03.000 with 0 trxs, lib: 7977
test2 generated block 85f9b073... #7979 @ 2018-04-16T03:44:03.500 with 0 trxs, lib: 7978
test2 generated block 36066d05... #7980 @ 2018-04-16T03:44:04.000 with 0 trxs, lib: 7979
test2 generated block 1d5314fa... #7981 @ 2018-04-16T03:44:04.500 with 0 trxs, lib: 7980
test2 generated block b5410853... #7982 @ 2018-04-16T03:44:05.000 with 0 trxs, lib: 7981
```

## 激活test2用户

```
./cleos --wallet-port 8899 push action eosio setprods "{\"version\": 1, \"producers\": [{\"producer_name\": \"test2\", \"block_signing_key\": \"EOS5BTsWP7hC7wAeUuoo6Q3LBTjiPLoEAM6hY51ujDEzsq7UNfex2\"]}]\" -p eosio@active
executed transaction: 8644f6707157ecceae7397000234e1725d2f18a39a53de27e1b375e6b5b85d1f 272 bytes 105472 cycles
#           eosio <= eosio::setprods {"version":1,"producers":[{"producer_name":"test2","block_signing_key":"EOS5BTsWP7hC7wAeUuoo6Q3LBTji...

```

## 状态检查

```
#节点1
```

```
./cleos get info
{
  "server_version": "74a4c7a0",
  "head_block_num": 7715,
  "last_irreversible_block_num": 7714,
  "head_block_id": "00001e23e240c450f2bf1b747ea64eba97f8ffd784f6a8e63daae279a386ab9d",
  "head_block_time": "2018-04-16T03:41:51",
  "head_block_producer": "test2"
}

#节点2
./cleos --port 8889 get info
{
  "server_version": "74a4c7a0",
  "head_block_num": 7770,
  "last_irreversible_block_num": 7769,
  "head_block_id": "00001e5a78fd1fcfa6b21c823e51b50f0d7984d1ff5eb0e297a48d2bd1b610b52",
  "head_block_time": "2018-04-16T03:42:19",
  "head_block_producer": "test2"
```

## 完 结

## 私链搭建(多机多节点)

### 最终创建三个虚拟机，三个生产节点的私链

各节点之间通过区块服务nodeos联通，区块服务是对公网的钱包服务keosd传输密钥时使用明文，所以只对本机提供服务  
cleos通过四个参数，设置连接哪个keosd和哪个nodeos  
cleos也可以直接使用nodeos提供的本地钱包服务操作钱包，但nodeos要加载钱包插件

## 开启服务

```
#BC01
./nodeos --http-server-address 0.0.0.0:8888 --p2p-listen-endpoint 0.0.0.0:9876 --enable-stale-production --producer-name eosio --plugin eosio::chain_api_plugin --plugin eosio::net_api_plugin --plugin eosio::wallet_api_plugin --plugin eosio::account_history_api_plugin 1>nodeos.log 2>&1 &
./keosd --http-server-address 127.0.0.1:8899 1>keosd.log 2>&1 &

#BC02, --p2p-peer-address指向BC01
./nodeos --http-server-address 0.0.0.0:8888 --p2p-listen-endpoint 0.0.0.0:9876 --p2p-peer-address 172.16.172.81:9876 --enable-stale-production --producer-name eosio --plugin eosio::chain_api_plugin --plugin eosio::net_api_plugin --plugin eosio::wallet_api_plugin --plugin eosio::account_history_api_plugin 1>nodeos.log 2>&1 &
./keosd --http-server-address 127.0.0.1:8899 1>keosd.log 2>&1 &

#BC03, --p2p-peer-address指向BC01
./nodeos --http-server-address 0.0.0.0:8888 --p2p-listen-endpoint 0.0.0.0:9876 --p2p-peer-address 172.16.172.81:9876 --enable-stale-production --producer-name eosio --plugin eosio::chain_api_plugin --plugin eosio::net_api_plugin --plugin eosio::wallet_api_plugin --plugin eosio::account_history_api_plugin 1>nodeos.log 2>&1 &
./keosd --http-server-address 127.0.0.1:8899 1>keosd.log 2>&1 &
```

## 创建钱包与账号

```
#BC01
#创建钱包、生成并导入密钥，创建账号
./cleos --wallet-port 8899 wallet create
"PW5KDZZWxtGroTQG8d1v1fiRm2FvNfE8QZzvnfZLEV8BGRPw8Tx8X"

./cleos create key
Private key: 5KPRMHBNnXLybgHFhFb7wVF9hLkRKu4aLb7s7xNaZND8GtunT8
Public key: EOS5wfaugnegAjXPSTgaquJL51taDmqftbGFcWmvNoDKMsYzrnydX
./cleos --wallet-port 8899 wallet import -n default 5KPRMHBNnXLybgHFhFb7wVF9hLkRKu4aLb7s7xNaZND8GtunT8

./cleos --wallet-port 8899 create account eosio token1 EOS5wfaugnegAjXPSTgaquJL51taDmqftbGFcWmvNoDKMsYzrnydX
EOS5wfaugnegAjXPSTgaquJL51taDmqftbGFcWmvNoDKMsYzrnydX
./cleos --wallet-port 8899 create account eosio patient1 EOS5wfaugnegAjXPSTgaquJL51taDmqftbGFcWmvNoDKMsYzrnydX
X EOS5wfaugnegAjXPSTgaquJL51taDmqftbGFcWmvNoDKMsYzrnydX

#BC02
#创建钱包、生成并导入密钥，创建账号
./cleos --wallet-port 8899 wallet create
"PW5JJv6uxu8FcxB3zRbzVuxDquKFeeGQj6f2kguXqfApSx9Lyg5k"

./cleos create key
Private key: 5K2G75aBEqBWRZdJZcFcb7gCjezxfc8hFMUk4XVPmGj8JmqSCA
```

```

Public key: EOS5baefwj8ALKjjo9fZsjtDagYecd3VVVjwaMuVN55e44ViJQXve
./cleos --wallet-port 8899 wallet import -n default 5K2G75aBEqBWRzdJZcFcb7gCjezcxfc8hFMUk4XVPmGj8JmqSCA

./cleos --wallet-port 8899 create account eosio patient2 EOS5baefwj8ALKjjo9fZsjtDagYecd3VVVjwaMuVN55e44ViJQXve
e EOS5baefwj8ALKjjo9fZsjtDagYecd3VVVjwaMuVN55e44ViJQXve

#BC03
#创建钱包、生成并导入密钥, 创建账号
./cleos --wallet-port 8899 wallet create
"PW5JWH5MHmrZhAgbmmMMNEHj4hdG2HxueCX8amKaNxX5o1kyclkmf"

./cleos create key
Private key: 5JM4jmDJoBdjTksz5KcBkCqrZxB3hi11GCPohbMg67r2pZu1P1E
Public key: EOS5MwWYV4dDVyrY3gpiZrfPpvX2ADWA2C8n9TT4D1Qho7BHHSLK
./cleos --wallet-port 8899 wallet import -n default 5JM4jmDJoBdjTksz5KcBkCqrZxB3hi11GCPohbMg67r2pZu1P1E

./cleos --wallet-port 8899 create account eosio patient3 EOS5MwWYV4dDVyrY3gpiZrfPpvX2ADWA2C8n9TT4D1Qho7BHHSLK
h EOS5MwWYV4dDVyrY3gpiZrfPpvX2ADWA2C8n9TT4D1Qho7BHHSLK

```

## 加载智能合约, 发币并转账

```

#BC01
./cleos --wallet-port 8899 set contract token1 ..//contracts/eosio.token -p token1
./cleos --wallet-port 8899 push action token1 create '{"issuer":"token1", "maximum_supply":"10000.00 CT", "can_freeze":0, "can_recall":0, "can_whitelist":0}' -p token1
./cleos --wallet-port 8899 push action token1 issue '[["patient1", "100.00 CT", "memo"]]' -p token1
./cleos --wallet-port 8899 push action token1 transfer '[["patient1", "patient2", "5.00 CT", "Just a test"]]' -p patient1

#BC02
./cleos --wallet-port 8899 push action token1 transfer '[["patient2", "patient3", "2.00 CT", "Just a test"]]' -p patient2

#BC03
./cleos --wallet-port 8899 push action token1 transfer '[["patient3", "patient1", "1.00 CT", "Just a test"]]' -p patient3

```

## 查询结果

```

#任意节点
./cleos --wallet-port 8899 get table token1 token1 accounts
{
  "rows": [
    {
      "balance": "0.00 CT",
      "frozen": 0,
      "whitelist": 1
    }
  ],
  "more": false
}

./cleos --wallet-port 8899 get table token1 patient1 accounts
{
  "rows": [
    {
      "balance": "96.00 CT",
      "frozen": 0,
      "whitelist": 1
    }
  ]
}

```

```
        },
    ],
    "more": false
}

./cleos --wallet-port 8899 get table token1 patient2 accounts
{
    "rows": [
        {
            "balance": "3.00 CT",
            "frozen": 0,
            "whitelist": 1
        }
    ],
    "more": false
}

./cleos --wallet-port 8899 get table token1 patient3 accounts
{
    "rows": [
        {
            "balance": "1.00 CT",
            "frozen": 0,
            "whitelist": 1
        }
    ],
    "more": false
}
```

## Fabric

项目	内容
链名称	Fabric
链类型	联盟链或私链
匿名性	共有或认证
共识机制	No-op/PBFT
开发语言	Go/Java
合约语言	Go/JS/Chaincode
官方网站	<a href="https://www.hyperledger.org">https://www.hyperledger.org</a>
白皮书	<a href="https://docs.google.com/document/d/1Z4M_qwILLRehPbVRUsJ3OF8lir-gqS-ZYe7W-LE9gnE/pub">https://docs.google.com/document/d/1Z4M_qwILLRehPbVRUsJ3OF8lir-gqS-ZYe7W-LE9gnE/pub</a>
项目地址	<a href="https://github.com/hyperledger/fabric/">https://github.com/hyperledger/fabric/</a>

## 项目特点

Fabric是大厂支持的开源区块链项目，相关开发资料十分完善，是当下很火的区块链项目。

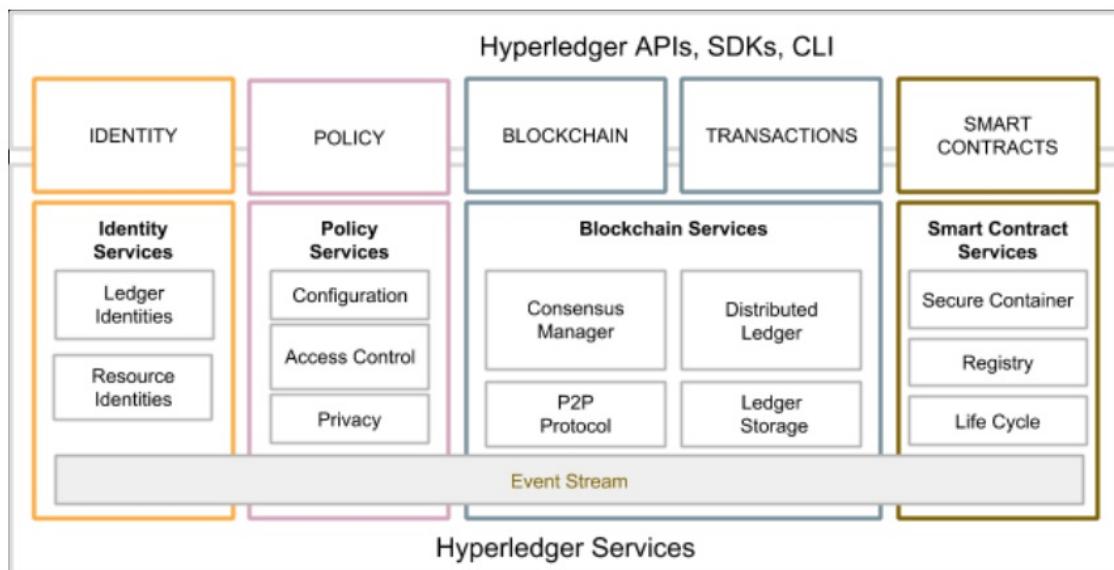
Fabric的目标主要是做联盟链，提供了MSP、Organizations、CA、Identity、Policy、Peer、Orderer、Transaction等传统安全相关概念与Block、Ledger、Blockchain、Chaincode、SmartContract等区块链概念相结合，给出了一套企业级的解决方案。

Fabric提供了可插拔的共识模块，为其后续扩展提供了很多的可能。

其最特别的贡献，在我看来，是提出了Channel的概念，在受控环境下可以大大提供共识速度，但这也是很多人诟病Fabric的地方之一。

## 整体架构

### Hyperledger架构



### Fabric架构

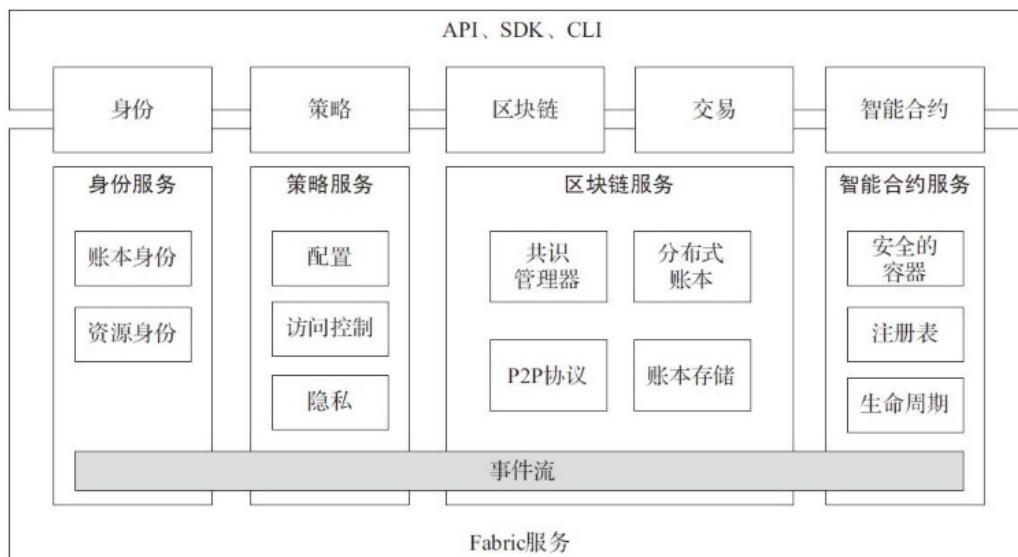


图8-1 Fabric项目的架构

### Sawtooth架构

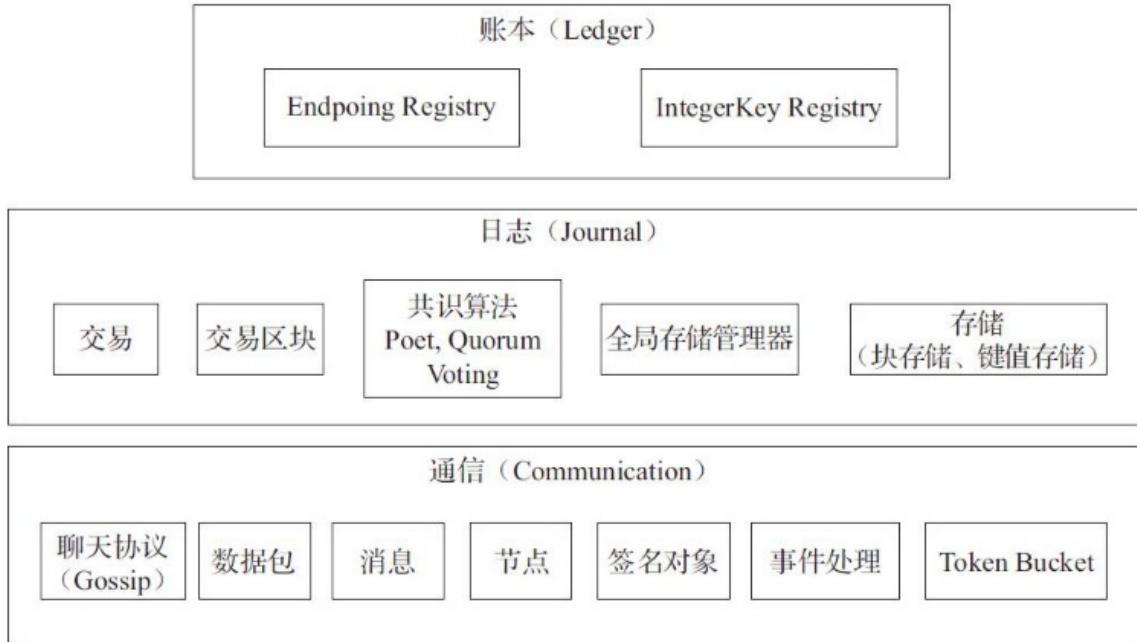
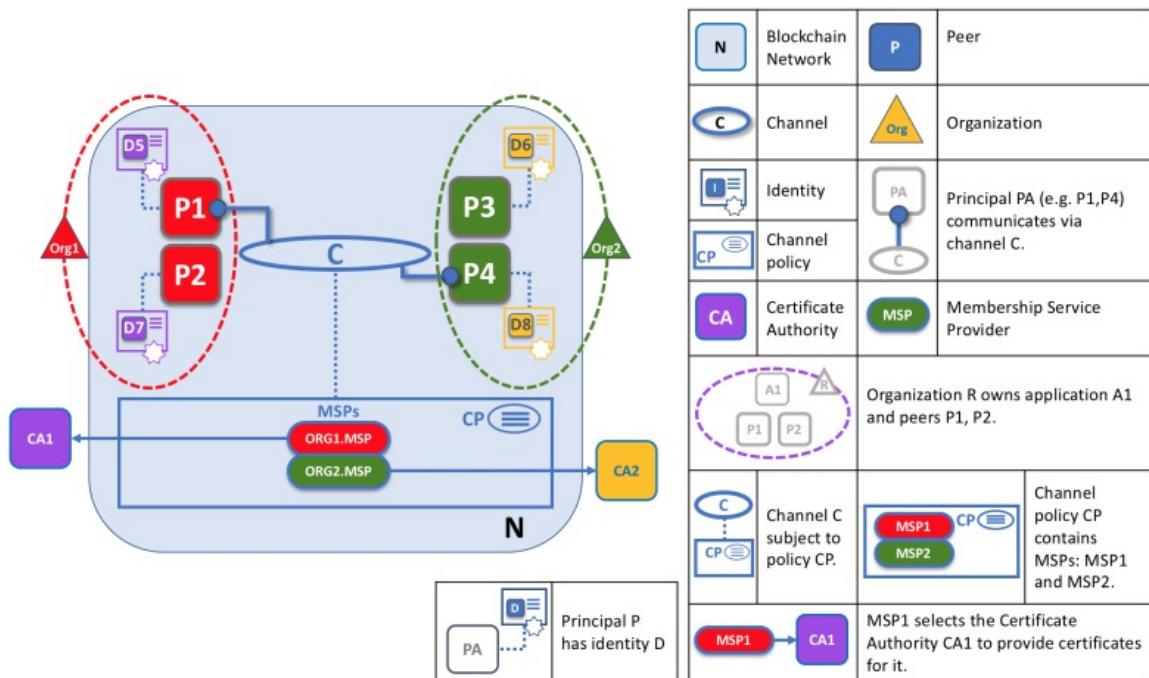
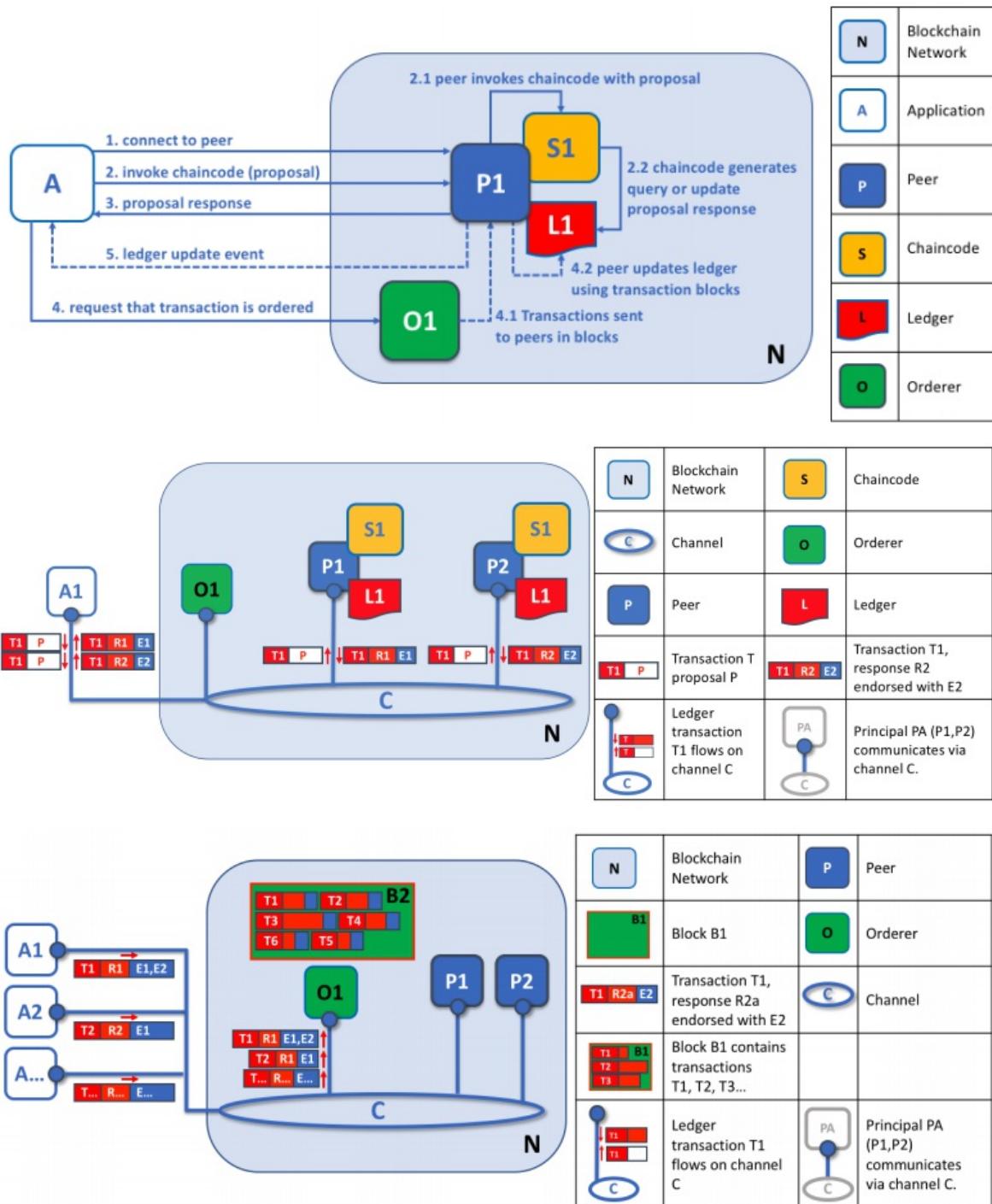


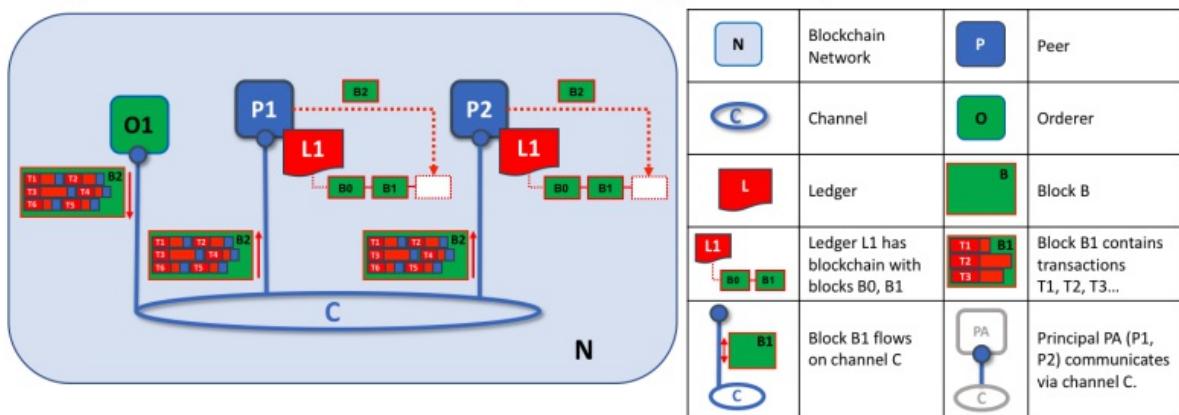
图8-5 锯齿湖项目架构

## 身份管理



## 共识达成





## 使用

Fabric的常用命令行等资料

## 智能合约介绍

fabric的智能合约叫做chaincode, 其分为两种, System Chaincode 和 普通Chaincode。

### System Chaincode

```
#System Chaincode是用来控制整个系统行为的, 其部署是通过peer代码的更新来完成的, 其使用, 是通过API调用完成的。  
System Chaincode当前分为以下几种  
1. LSCC Lifecycle system chaincode  
2. CSCC Configuration system chaincode  
3. QSCC Query system chaincode  
4. ESCC Endorsement system chaincode  
5. VSCC Validation system chaincode  
  
#此外, System Chaincode是支持插件的
```

### 普通Chaincode

```
普通ChainCode其实就是智能合约。  
但由于fabric其本身有强大的权限控制功能, 会导致整个过程比以太坊等公链中, 部署合约要复杂的多。  
好在fabric开发团队提供了各种Docker环境, 减少了我们的痛苦。
```

# 智能合约使用01

## 下载示例

```
git clone -b master https://github.com/hyperledger/fabric-samples.git
cd fabric-samples
git checkout v1.1.0
```

## 安装依赖包

```
cd fabric-samples/fabcar
./npm install
```

## 查看合约内容

智能合约文件在：

```
fabric-samples/chaincode/fabcar/go/fabcar.go
```

```
package main

// 后两个引用是智能合约相关的
import (
    "bytes"
    "encoding/json"
    "fmt"
    "strconv"
    "github.com/hyperledger/fabric/core/chaincode/shim"
    sc "github.com/hyperledger/fabric/protos/peer"
)

// 合约结构
type SmartContract struct {
}

// Car结构
type Car struct {
    Make   string `json:"make"`
    Model  string `json:"model"`
    Colour string `json:"colour"`
    Owner   string `json:"owner"`
}

// 合约初始化
func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {
    return shim.Success(nil)
}

//消息分发函数
//通过这个函数响应对应的消息，调用后面的功能函数
func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {
    // Retrieve the requested Smart Contract function and arguments
}
```

```

function, args := APIstub.GetFunctionAndParameters()
// Route to the appropriate handler function to interact with the ledger appropriately
if function == "queryCar" {
    return s.queryCar(APIstub, args)
} else if function == "initLedger" {
    return s.initLedger(APIstub)
} else if function == "createCar" {
    return s.createCar(APIstub, args)
} else if function == "queryAllCars" {
    return s.queryAllCars(APIstub)
} else if function == "changeCarOwner" {
    return s.changeCarOwner(APIstub, args)
}

return shim.Error("Invalid Smart Contract function name.")
}

//根据条件查询车
func (s *SmartContract) queryCar(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }

    carAsBytes, _ := APIstub.GetState(args[0])
    return shim.Success(carAsBytes)
}

//初始化合约数据
func (s *SmartContract) initLedger(APIstub shim.ChaincodeStubInterface) sc.Response {
    cars := []Car{
        Car{Make: "Toyota", Model: "Prius", Colour: "blue", Owner: "Tomoko"},
        Car{Make: "Ford", Model: "Mustang", Colour: "red", Owner: "Brad"},
        Car{Make: "Hyundai", Model: "Tucson", Colour: "green", Owner: "Jin Soo"},
        Car{Make: "Volkswagen", Model: "Passat", Colour: "yellow", Owner: "Max"},
        Car{Make: "Tesla", Model: "S", Colour: "black", Owner: "Adriana"},
        Car{Make: "Peugeot", Model: "205", Colour: "purple", Owner: "Michel"},
        Car{Make: "Chery", Model: "S22L", Colour: "white", Owner: "Aarav"},
        Car{Make: "Fiat", Model: "Punto", Colour: "violet", Owner: "Pari"},
        Car{Make: "Tata", Model: "Nano", Colour: "indigo", Owner: "Valeria"},
        Car{Make: "Holden", Model: "Barina", Colour: "brown", Owner: "Shotaro"},
    }

    i := 0
    for i < len(cars) {
        fmt.Println("i is ", i)
        carAsBytes, _ := json.Marshal(cars[i])
        APIstub.PutState("CAR"+strconv.Itoa(i), carAsBytes)
        fmt.Println("Added", cars[i])
        i = i + 1
    }

    return shim.Success(nil)
}

//新建一辆车
func (s *SmartContract) createCar(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
    if len(args) != 5 {
        return shim.Error("Incorrect number of arguments. Expecting 5")
    }
}

```

```

var car = Car{Make: args[1], Model: args[2], Colour: args[3], Owner: args[4]}

carAsBytes, _ := json.Marshal(car)
APIstub.PutState(args[0], carAsBytes)

return shim.Success(nil)
}

//查询全部的车
func (s *SmartContract) queryAllCars(APIstub shim.ChaincodeStubInterface) sc.Response {

startKey := "CAR0"
endKey := "CAR999"

resultsIterator, err := APIstub.GetStateByRange(startKey, endKey)
if err != nil {
    return shim.Error(err.Error())
}
defer resultsIterator.Close()

// buffer is a JSON array containing QueryResults
var buffer bytes.Buffer
buffer.WriteString("[")  

bArrayMemberAlreadyWritten := false
for resultsIterator.HasNext() {
    queryResponse, err := resultsIterator.Next()
    if err != nil {
        return shim.Error(err.Error())
    }
    // Add a comma before array members, suppress it for the first array member
    if bArrayMemberAlreadyWritten == true {
        buffer.WriteString(",")
    }
    buffer.WriteString("{\"Key\":")
    buffer.WriteString("\"")
    buffer.WriteString(queryResponse.Key)
    buffer.WriteString("\"")

    buffer.WriteString(", \"Record\":")
    // Record is a JSON object, so we write as-is
    buffer.WriteString(string(queryResponse.Value))
    buffer.WriteString("}")
    bArrayMemberAlreadyWritten = true
}
buffer.WriteString("]")

fmt.Printf("- queryAllCars:\n%s\n", buffer.String())

return shim.Success(buffer.Bytes())
}

//修改车主
func (s *SmartContract) changeCarOwner(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

if len(args) != 2 {
    return shim.Error("Incorrect number of arguments. Expecting 2")
}

carAsBytes, _ := APIstub.GetState(args[0])

```

```

car := Car{}

json.Unmarshal(carAsBytes, &car)
car.Owner = args[1]

carAsBytes, _ = json.Marshal(car)
APIstub.PutState(args[0], carAsBytes)

return shim.Success(nil)
}

//测试代码
func main() {

    // Create a new Smart Contract
    err := shim.Start(new(SmartContract))
    if err != nil {
        fmt.Printf("Error creating new Smart Contract: %s", err)
    }
}

```

## 部署合约

```

./startFabric.sh

#清理工作
sudo docker-compose -f docker-compose.yml down
Removing network net_basic
WARNING: Network net_basic not found.

#开启虚机准备网络
sudo docker-compose -f docker-compose.yml up -d ca.example.com orderer.example.com peer0.org1.example.com couchdb
Creating network "net_basic" with the default driver
Creating orderer.example.com ...
Creating couchdb ...
Creating ca.example.com ...
Creating couchdb
Creating orderer.example.com
Creating couchdb ... done
Creating peer0.org1.example.com ...
Creating peer0.org1.example.com ... done

#等待启动完毕
# wait for Hyperledger Fabric to start

#开启管道
#节点加入管道
# Create the channel
sudo docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp" peer0.org1.example.com peer channel create -o orderer.example.com:7050 -c mychannel -f /etc/hyperledger/configtx/channel.tx
# Join peer0.org1.example.com to the channel.
sudo docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp" peer0.org1.example.com peer channel join -b mychannel.block

#开启cli
Creating cli ...
Creating cli ... done

```

```
#部署合约  
  
#完成  
Total setup execution time : 37 secs ...
```

## 创建用户

```
node enrollAdmin.js  
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store  
Successfully enrolled admin user "admin"  
Assigned the admin user to the fabric client ::{"name":"admin","mspid":"Org1MSP","roles":null,"affiliation":"","enrollmentSecret":"","enrollment":{"signingIdentity":"b07a6f50d049f48f9956aca0681e9008011580571539e1ec5f77277749748468","identity":{"certificate":"-----BEGIN CERTIFICATE-----\nMIICATCCAigAwIBAgIUEFYVmdzPesMH3QeISPzK/4sPBRQwCgYIKoZIzj0EAwIw\\nczELMAkGA1UEBhMCVVMxExARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNh\\nbIBGcmFuY2lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAABgNVBAMT\\nE2NhLm9yZzEuZXhhbXBsZS5jb20wHhcNMTgwNTA2MTEzMDAwWhcNMTkwNTA2MTEz\\nNTAwWjAhMQ8wDQYDVQQLEwZjbG1lbnQxDjAMBgNVBAMTBWFkbWluMFkwEwYHKoZI\\nj0CAQYIKoZIzj0DAQcDQgAEeCQVU3sN0vEuWjjMZZk/Sazmtw+inpeJhB8n43hf\\nSLEa6nyPcXs1xlp89ja6TDB0SV0RT7n7PMLu5u3EMCn0AaNsMGowDgYDVR0PAQH/\\nBAQDAgeAMAwGA1UDewEB/wQCMAAwHQYDVVR0OBByEFE4/06IYrSMbUgTuc8I197Sx\\nBLfAMCsGA1UdIwQkMCKAIEI5qg3NdtruuLoM2nAYUdFFBNMarRst3dusalc2Xk18\\nMaOGCCqGSM49BAMCA0cAMEQCIEFTsT089nYM0FLfWaT/gC+kT7B7BW8knqQ1TGTQ\\nn8oxAiA4sVmzhvkCUvZ6f723pqKe70t6HK/Bxn0+Yc28M+uJUw==\\n-----END CERTIFICATE-----\\n"}}  
  
node registerUser.js  
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store  
Successfully loaded admin from persistence  
Successfully registered user1 - secret:NlaDuZJaUyMp  
Successfully enrolled member user "user1"  
User1 was successfully registered and enrolled and is ready to interact with the fabric network
```

## 查询全部数据

```
node query.js  
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store  
Successfully loaded user1 from persistence  
Query has completed, checking results  
Response is [{"Key":"CAR0", "Record":{"colour":"blue","make":"Toyota","model":"Prius","owner":"Tomoko"}}, {"Key":"CAR1", "Record":{"colour":"red","make":"Ford","model":"Mustang","owner":"Brad"}}, {"Key":"CAR2", "Record":{"colour":"green","make":"Hyundai","model":"Tucson","owner":"Jin Soo"}}, {"Key":"CAR3", "Record":{"colour":"yellow","make":"Volkswagen","model":"Passat","owner":"Max"}}, {"Key":"CAR4", "Record":{"colour":"black","make":"Tesla","model":"S","owner":"Adriana"}}, {"Key":"CAR5", "Record":{"colour":"purple","make":"Peugeot","model":"205","owner":"Michel"}}, {"Key":"CAR6", "Record":{"colour":"white","make":"Chery","model":"S22L","owner":"Aarav"}}, {"Key":"CAR7", "Record":{"colour":"violet","make":"Fiat","model":"Punto","owner":"Pari"}}, {"Key":"CAR8", "Record":{"colour":"indigo","make":"Tata","model":"Nano","owner":"Valeria"}}, {"Key":"CAR9", "Record":{"colour":"brown","make":"Holden","model":"Barina","owner":"Shotaro"}}]
```

## 带条件的查询

```
#修改query.js的这部分内容  
const request = {  
    chaincodeId: 'fabcar',  
    fcn: 'queryCar',  
    args: ['CAR4']  
};
```

```
node query.js
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  {"colour":"black","make":"Tesla","model":"S","owner":"Adriana"}
```

## 更新数据

```
#修改invoke.js的这部分内容
var request = {
    chaincodeId: 'fabcar',
    fcn: 'changeCarOwner',
    args: ['CAR10', 'Dave'],
    chainId: 'mychannel',
    txId: tx_id
};

#修改所有权
node invoke.js
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store
Successfully loaded user1 from persistence
Assigning transaction_id: a126f84d2ec199abacb59eedfeba29d223534efdf8f01333da66908b97aa94e5
Transaction proposal was good
Successfully sent Proposal and received ProposalResponse: Status - 200, message - "OK"
The transaction has been committed on peer localhost:7053
Send transaction promise and event listener promise have completed
Successfully sent transaction to the orderer.
Successfully committed the change to the ledger by the peer

#查询所有权
node query.js
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  {"colour":"red","make":"Ford","model":"Mustang","owner":"neohope"}
```

## 新增数据

```
#修改invoke.js的这部分内容
var request = {
    chaincodeId: 'fabcar',
    fcn: 'createCar',
    args: ['CAR10', 'Honda', 'Accord', 'Black', 'Tom'],
    chainId: 'mychannel',
    txId: tx_id
};

#修改query.js的这部分内容
const request = {
    chaincodeId: 'fabcar',
    fcn: 'queryCar',
    args: ['CAR10']
};
```

```
node invoke.js
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store
Successfully loaded user1 from persistence
Assigning transaction_id: cecf39954c4badd43691fe2fa0a124cb19cf66736837366e698134d5b91cfe1b
Transaction proposal was good
Successfully sent Proposal and received ProposalResponse: Status - 200, message - "OK"
The transaction has been committed on peer localhost:7053
Send transaction promise and event listener promise have completed
Successfully sent transaction to the orderer.
Successfully committed the change to the ledger by the peer
```

```
node query.js
Store path:/home/neohope/fabric-samples/fabcar/hfc-key-store
Successfully loaded user1 from persistence
Query has completed, checking results
Response is {"colour":"Black","make":"Honda","model":"Accord","owner": "Tom"}
```

## 停止网络

```
cd fabric-samples/basic-network
./stop.sh
```

## 智能合约使用02

### 合约文件sacc.go

```
//代码位置:  
//fabric-samples/chaincode/sacc/sacc.go  
//合约很简单就是kv储存、查询和更新的功能  
  
package main  
  
import (  
    "fmt"  
    "github.com/hyperledger/fabric/core/chaincode/shim"  
    "github.com/hyperledger/fabric/protos/peer"  
)  
  
type SimpleAsset struct {  
}  
  
// 部署及初始化时调用此函数  
// 代码升级时也会调用此函数重置和迁移数据  
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {  
    // Get the args from the transaction proposal  
    args := stub.GetStringArgs()  
    if len(args) != 2 {  
        return shim.Error("Incorrect arguments. Expecting a key and a value")  
    }  
  
    // Set up any variables or assets here by calling stub.PutState()  
  
    // We store the key and the value on the ledger  
    err := stub.PutState(args[0], []byte(args[1]))  
    if err != nil {  
        return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))  
    }  
    return shim.Success(nil)  
}  
  
// 消息路由函数  
// 调用 set 或 get方法  
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {  
    // Extract the function and args from the transaction proposal  
    fn, args := stub.GetFunctionAndParameters()  
  
    var result string  
    var err error  
    if fn == "set" {  
        result, err = set(stub, args)  
    } else { // assume 'get' even if fn is nil  
        result, err = get(stub, args)  
    }  
    if err != nil {  
        return shim.Error(err.Error())  
    }  
  
    // Return the result as success payload  
    return shim.Success([]byte(result))  
}
```

```

}

// Set
func set(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 2 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key and a value")
    }

    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return "", fmt.Errorf("Failed to set asset: %s", args[0])
    }
    return args[1], nil
}

// Get
func get(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 1 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key")
    }

    value, err := stub.GetState(args[0])
    if err != nil {
        return "", fmt.Errorf("Failed to get asset: %s with error: %s", args[0], err)
    }
    if value == nil {
        return "", fmt.Errorf("Asset not found: %s", args[0])
    }
    return string(value), nil
}

// main function starts up the chaincode in the container during instantiate
func main() {
    if err := shim.Start(new(SimpleAsset)); err != nil {
        fmt.Printf("Error starting SimpleAsset chaincode: %s", err)
    }
}

```

## 在宿主机下尝试进行编译

```

cd fabric-samples/chaincode/sacc/
go build

```

## 开启开发环境

```

cd fabric-samples/chaincode-docker-devmode

#打开一个ssh
sudo docker-compose -f docker-compose-simple.yaml up

```

## 在虚拟机内进行编译合约

```

sudo docker exec -it chaincode bash

#在docker中运行

```

```
cd sacc  
go build  
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./sacc
```

## 在虚拟机内部署并运行合约

```
sudo docker exec -it cli bash  
  
#在docker中运行  
peer chaincode install -p chaincodedev/chaincode/sacc -n mycc -v 0  
peer chaincode instantiate -n mycc -v 0 -c '{"Args":["a","10"]}' -C myc  
peer chaincode invoke -n mycc -c '{"Args":["set", "a", "20"]}' -C myc  
peer chaincode query -n mycc -c '{"Args":["query","a"]}' -C myc
```

## 打包签名部署命令行

```
#打包  
#-n合约名  
#-p路径  
#-v版本  
#-s签名  
#-S多重签名  
#-i初始化策略  
peer chaincode package -n 名称 -p 路径 -v 0 -s -S -i "AND('OrgA.admin')" 打包名  
  
#签名  
peer chaincode signpackage 打包名 签名后的打包名  
  
#部署  
#-n合约名  
#-v版本  
#-p路径  
peer chaincode install -n 名称 -v 1.0 -p 路径  
  
#初始化  
#-n合约名  
#-v版本  
#-c初始化参数  
#-P策略  
peer chaincode instantiate -n 名称 -v 1.0 -c '{"Args":["john","0"]}' -P "OR ('Org1.member','Org2.member')"  
  
#部署0版本  
peer chaincode install -n 名称 -v 0 -p 路径  
#初始化0版本  
peer chaincode instantiate -n 名称 -v 0 -c '{"Args":["a", "b", "c"]}' -C channel名称  
#部署1版本  
peer chaincode install -n 名称 -v 1 -p 路径  
#升级为1版本  
peer chaincode upgrade -n 名称 -v 1 -c '{"Args":["d", "e", "f"]}' -C channel名称  
#查询  
peer chaincode query -C channel名称 -n 名称 -c '{"Args":["query","e"]}'  
#调用  
peer chaincode invoke -o orderer.example.com:7050 --tls --cafile $ORDERER_CA -C channel名称 -n 名称 -c '{"Args": ["invoke", "a", "b", "10"]}'
```

## 打包时附带lib

```
#初始化  
govendor init  
#增加全部外部包  
govendor add +external  
#增加某个包  
govendor add packagepath  
#然后再编译、打包就好了
```

## 数据加密

```
#fabric是支持合约数据加密的  
#但官方文档上没有找到合适的例子  
#待补充
```

## 编译(MacOS)

### 安装docker、go等必须的软件

```
brew install docker dokcer-machine go  
brew install gnu-tar --with-default-names  
brew install libtool
```

### 开启dokcer deamon环境

```
docker-machine create forfabric  
eval $(docker-machine env forfabric)
```

### 设置gopath环境变量

```
#创建目录  
mkdir PAHT_TO_GOPATH  
  
#导出环境变量  
export gopath=PAHT_TO_GOPATH  
  
#然后在gopath下创建下面的目录  
PAHT_TO_GOPATH/src/github.com/hyperledger/  
  
#设置必要的代理信息  
export http_proxy=xxxx:xx  
export https_proxy=xxxx:xx  
export no_proxy="localhost, 127.0.0.1, 192.168.99.100, 根据你自己的实际情况进行处理"
```

### 下载源码

```
cd PAHT_TO_GOPATH/src/github.com/hyperledger/  
git clone https://github.com/hyperledger/fabric.git
```

### 编译gotools

```
cd fabric  
make gotools
```

### 编译native

```
make native
```

### 完结



## 私链搭建(Ubuntu)

### 安装需要的软件

```
#ubuntu18
sudo apt-get install curl docker docker-compose golang nodejs npm python
```

软件	版本
docker	17.06.2-ce+
docker-compose	1.14.0+
go	1.9.x+
nodejs	8.9.x
bon	5.6.0
python	2.7.x

### 设置环境变量

```
#!/.bashrc
export GOPATH=$HOME/gopath
export PATH=$PATH:$GOPATH/bin
```

### 准备环境

```
#下载脚本
wget https://raw.githubusercontent.com/hyperledger/fabric/master/scripts/bootstrap.sh

#运行脚本, 可以考虑将里面的docker命令前增加su
./bootstrap.sh
```

### 运行私链

```
cd fabric-samples/first-network

#初始化配置
./byfn.sh -m generate

#这个脚本, 做了以下事情:
#创建证书
#生成第一个区块
#生成MSP的锚节点
#启动管道mychannel
#管道包括了两个组织, org1和org2
#org1有两个节点peer0, peer1
#org2有两个节点peer0, peer1
#然后在org1和org2的peer0节点部署了智能合约
#查询合约
#合约转账
#查询合约
```

```

sudo ./byfn.sh -m up
#####
    Generate certificates using cryptogen tool #####
#####
    Generating Orderer Genesis block #####
#####
    Generating anchor peer update for Org1MSP #####
#####
    Generating anchor peer update for Org2MSP #####
#####
        Channel "mychannel" is created successfully =====
#####
        peer0.org1 joined on the channel "mychannel" =====
#####
        peer1.org1 joined on the channel "mychannel" =====
#####
        peer0.org2 joined on the channel "mychannel" =====
#####
        peer1.org2 joined on the channel "mychannel" =====
#####
        Anchor peers for org "Org1MSP" on "mychannel" is updated successfully =====
=====
#####
        Anchor peers for org "Org2MSP" on "mychannel" is updated successfully =====
=====
#####
        Chaincode is installed on peer0.org1 =====
#####
        Chaincode is installed on peer0.org2 =====
#####
        Chaincode Instantiation on peer0.org2 on channel 'mychannel' is successful =====
=====
#####
        Querying on peer0.org1 on channel 'mychannel'... =====
Query Result: 100
#####
        Query on peer0.org1 on channel 'mychannel' is successful =====
#####
        Invoke transaction on peer0.org1 on channel 'mychannel' is successful =====
=====
#####
        Chaincode is installed on peer1.org2 =====
#####
        Querying on peer1.org2 on channel 'mychannel'... =====
Query Result: 90
#####
        Query on peer1.org2 on channel 'mychannel' is successful =====
=====
All GOOD, BYFN execution completed =====

```

## 查看容器列表

COLUMN	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
			NAMES			
92b34725eb84		dev-peer1.org2.example.com-mycc-1.0-26c2ef32838554aac4f7ad6f100aca865e87959c9a126e86d764c	"chaincode -peer.add..."	14 minutes ago	Up 14 minutes	
8d01f8346ab		dev-peer1.org2.example.com-mycc-1.0				
fc09cb911752		dev-peer0.org1.example.com-mycc-1.0-384f11f484b9302df90b453200cfb25174305fce8f53f4e94d45e	"chaincode -peer.add..."	15 minutes ago	Up 15 minutes	
e3b6cab0ce9		dev-peer0.org1.example.com-mycc-1.0				
bed1373228f0		dev-peer0.org2.example.com-mycc-1.0-15b571b3ce849066b7ec74497da3b27e54e0df1345daff3951b94	"chaincode -peer.add..."	15 minutes ago	Up 15 minutes	
245ce09c42b		dev-peer0.org2.example.com-mycc-1.0				
bfa16375cae2		hyperledger/fabric-tools:latest	"/bin/bash"	16 minutes ago	Up 16 minutes	
		cli				
97bbdad5a204		hyperledger/fabric-orderer:latest	"orderer"	16 minutes ago	Up 16 minutes	0.0.0.0:7050->7050/tcp
		orderer.example.com				
9743b8716888		hyperledger/fabric-peer:latest	"peer node start"	16 minutes ago	Up 16 minutes	0.0.0.0:10051->7051/tcp, 0.0.0
.0:10053->7053/tcp		peer1.org2.example.com				
d3ea4a45d1f3		hyperledger/fabric-peer:latest	"peer node start"	16 minutes ago	Up 16 minutes	0.0.0.0:8051->7051/tcp, 0.0.0.
0:8053->7053/tcp		peer1.org1.example.com				
514337d43df4		hyperledger/fabric-peer:latest	"peer node start"	16 minutes ago	Up 16 minutes	0.0.0.0:7051->7051/tcp, 0.0.0.
0:7053->7053/tcp		peer0.org1.example.com				

```
3ced253d6808      hyperledger/fabric-peer:latest
```

## 查看交易细节

```
sudo docker logs dev-peer0.org2.example.com-mycc-1.0
ex02 Init
Aval = 100, Bval = 200

sudo docker logs dev-peer0.org1.example.com-mycc-1.0
ex02 Invoke
Query Response:{"Name":"a","Amount":"100"}
ex02 Invoke
Aval = 90, Bval = 210

sudo docker logs dev-peer1.org2.example.com-mycc-1.0
ex02 Invoke
Query Response:{"Name":"a","Amount":"90"}
```

## 清理工作

```
sudo ./byfn.sh -m down
```

# 在私链中添加一个机构

## 清理并启动网络

```
#清理
cd fabric-samples/first-network
sudo ./eyfn.sh -m down
sudo ./byfn.sh -m down

#初始化配置
sudo ./byfn.sh -m generate -c nchannel

#启动私链
sudo ./byfn.sh -m up -t 30 -d 10 -c nchannel

#新增一个机构org3, 但org3的节点会告诉你无法加入nchannel
#因为org3不在nchannel中
sudo ./eyfn.sh -m up -t 30 -d 10 -c nchannel
```

## 生成org3的配置文件

```
#生成配置文件
cd org3-artifacts
sudo ../../bin/cryptogen generate --config=./org3-crypto.yaml
export FABRIC_CFG_PATH=$PWD
sudo ../../bin/configtxgen -printOrg Org3MSP > ../channel-artifacts/org3.json

#拷贝到指定位置
cd ../
sudo cp -r crypto-config/ordererOrganizations org3-artifacts/crypto-config/
```

## 获取并修改当前配置文件

```
sudo docker exec -it cli bash

#虚拟机内

#安装jq
apt update && apt install -y jq

#配置环境变量
export ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
export CHANNEL_NAME=nchannel
echo $ORDERER_CA && echo $CHANNEL_NAME

#获取最新配置文件
peer channel fetch config config_block.pb -o orderer.example.com:7050 -c $CHANNEL_NAME --tls --cafile $ORDERER_CA

#修改配置文件格式为json
#更新配置文件
#新增org3
#修改配置文件格式为pb
```

```

configtxlator proto_decode --input config_block.pb --type common.Block | jq .data.data[0].payload.data.config
> config.json
jq -s '[.[] * {"channel_group":{"groups":{"Application":{"groups": [{"Org3MSP":.[1]}]}}}' config.json ./channel
1-artifacts/org3.json > modified_config.json
configtxlator proto_encode --input config.json --type common.Config --output config.pb
configtxlator proto_encode --input modified_config.json --type common.Config --output modified_config.pb
configtxlator compute_update --channel_id $CHANNEL_NAME --original config.pb --updated modified_config.pb --o
utput org3_update.pb
configtxlator proto_decode --input org3_update.pb --type common.ConfigUpdate | jq . > org3_update.json
echo '{"payload":{"header":{"channel_header":{"channel_id":"nchannel", "type":2}}, "data":{"config_update":"'$(cat org3_update.json)'}}' | jq . > org3_update_in_envelope.json
configtxlator proto_encode --input org3_update_in_envelope.json --type common.Envelope --output org3_update_i
n_envelope.pb

#用org1进行签名
peer channel signconfigtx -f org3_update_in_envelope.pb

#非常规操作
#用org2进行部署(现在channel中只有org1和org2, 所以org1签名, org2部署就是全部org都认可org3了)
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganization
s/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/or
g2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=peer0.org2.example.com:7051

#但是这个地方一直报错
#我试了10多次, 只有1次成功了, 还是在开会, 之后再没有成功过
peer channel update -f org3_update_in_envelope.pb -c $CHANNEL_NAME -o orderer.example.com:7050 --tls --cafile
$ORDERER_CA

#错误内容
Rejecting broadcast of config message from 172.19.0.7:38982 because of error: error authorizing update: error
validating DeltaSet:
policy for [Value] /Channel/Application/Org3MSP/MSP not satisfied: signature set did not satisfy policy

```

## 配置选举方式

```

#peer0.org3
#peer1.org3
#默认采用了dynamic配置方式, 所以默认可以不用进行配置, 可以跳过这一步

#static
CORE_PEER_GOSSIP_USELEADERELECTION=false
CORE_PEER_GOSSIP_ORGLEADER=true

#dynamic
CORE_PEER_GOSSIP_USELEADERELECTION=true
CORE_PEER_GOSSIP_ORGLEADER=false

```

## peer0.org3节点加入channel

```

#开启Org3cli
sudo docker-compose -f docker-compose-org3.yaml up -d
sudo docker exec -it Org3cli bash

#虚拟机内

```

```
export ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
export CHANNEL_NAME=nchannel
echo $ORDERER_CA && echo $CHANNEL_NAME

#org3获取创世区块
#必须只同步这个区块，才能加入，否则会失败
peer channel fetch 0 nchannel.block -o orderer.example.com:7050 -c $CHANNEL_NAME --tls --cafile $ORDERER_CA

#pee0.org3节点加入channel
peer channel join -b nchannel.block
```

## peer1.org3节点加入channel

```
#还是同一个虚拟机Org3cli
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizationS/org3.example.com/peers/peer1.org3.example.com/tls/ca.crt
export CORE_PEER_ADDRESS=peer1.org3.example.com:7051
peer channel join -b nchannel.block
```

## org3安装新版本合约

```
#还是同一个虚拟机Org3cli
#还是同一个binary
peer chaincode install -n mycc -v 2.0 -p github.com/chaincode/chaincode_example02/go/
```

## org2安装新版本合约

```
#虚拟机cli
#还是同一个binary
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizationS/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=peer0.org2.example.com:7051
peer chaincode install -n mycc -v 2.0 -p github.com/chaincode/chaincode_example02/go/
```

## org1安装新版本合约

```
#虚拟机cli
#还是同一个binary
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizationS/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=peer0.org1.example.com:7051
peer chaincode install -n mycc -v 2.0 -p github.com/chaincode/chaincode_example02/go/
```

## 启用新版本合约

```
#虚拟机Org3cli
#升级合约
peer chaincode upgrade -o orderer.example.com:7050 --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA -C $CHANNEL_NAME -n mycc -v 2.0 -c '{"Args":["init","a","90","b","210"]}' -P "OR ('Org1MSP.peer','Org2MSP.peer','Org3MSP.peer')"

#调用合约
peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","a"]}'
peer chaincode invoke -o orderer.example.com:7050 --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA -C $CHANNEL_NAME -n mycc -c '{"Args":["invoke","a","b","10"]}'
peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","a"]}'
```

## Cosmos

项目	内容
链名称	Cosmos, Cosmos hub
链类型	生态, 公链
匿名性	匿名
共识机制	PoS: Proof of Stack, Tendermint BFT
开发语言	不限, Go
合约语言	不限, 无
官方网站	<a href="https://cosmos.network/">https://cosmos.network/</a>
白皮书	<a href="https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md">https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md</a>
项目地址	<a href="https://github.com/cosmos/cosmos-sdk">https://github.com/cosmos/cosmos-sdk</a>

### 项目特点：

Cosmos引入了Validators节点作为共识节点，其余节点只是接受共识结果。但非Validators节点，可以将自己的Atom委托给某个Validators，从而加强某个Validators的投票权，从而通过委托方式来参与投票。

Cosmos其本身发布的第一条链为 Cosmos hub，用了PoS共识机制，可以投票的的币叫Atom，进行交易的币叫做Photons。

Cosmos引入了Zone的概念，和Fabric的组织有些像。Fabric组织是管理的是Peer，Zone其实是一条链（共有或私有）。

Cosmos引入了Hub的概念，Hub是一个特殊的Zone，和Fabric的Channel有些相像。Fabric的Channel连接的是多个组织，但Hub连接的是多个链。在同一个Hub下，多个链之间可以高效进行交易。

Cosmos引入了IBC(inter-blockchain communication)来实现链的接入，只要该链符合IBC，理论上就可以成为一个Zone，接入到某一个Hub中。

准确来说Cosmos的野心，并不是做一个公链，而是做生态：一个链可以将所有符合IBC接口标准的链，统一起来。

因此，Cosmos本身并不一定会提供合约功能，因为只要共识引擎符合其标准，Cosmos就可以将该链整个接入进来。当新链接入Cosmos时，如用引入了新的代币，会通过Hard spoon这种才做方式，来举行投票，确定是否支持接入，接入时间，以及Photons如何重新分配/通胀等。

用官方的话说，Cosmos本身并不是一个单纯的状态机，而更像是一个数据库。其他的链通过IBC接口方式接入，在Cosmos Hub中达成共识，Cosmos记录这些共识，从而解决了跨链互认的问题。

## 整体架构



## Zone与Hub



# 使用

## 编译源码

```
#需要Go1.1.0+
#设置环境变量
export GOPATH=$HOME/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin

#安装wget
sudo apt-get install wget

#下载cosmos-sdk
mkdir -p $GOPATH/src/github.com/cosmos/cosmos-sdk
git clone https://github.com/cosmos/cosmos-sdk.git
git checkout v0.17.3

#安装dep
#第一次
make get_tools
#升级
make update_tools

#编译并安装gaiad及gaiacli
make get_vendor_deps
make install

#查看版本
gaiad version
gaiacli version
```

## 加入测试网络

```
#下载测试网配置
git clone https://github.com/cosmos/testnets

#拷贝5001配置文件
mkdir -p $HOME/.gaiad/config
cp -a testnets/gaia-5001/. $HOME/.gaiad/config/gentx

#初始化
gaiad unsafe_reset_all
gaiad init --gen-txs -o --chain-id=gaia-5001
{
  "chain_id": "gaia-5001",
  "node_id": "de87d2889c39ff159aad1b8320ca167fc20abcb9",
  "app_message": null
}

#修改配置文件, 起一个自己的名字
vi ~/.gaiad/config/config.toml
moniker = "tnode01"

#接入测试网5001
gaiad start
```

```
#比较漫长的等待, 等待区块同步完成
```

```
#可以通过Cosmos Monitor查看相关内容:
```

```
#http://IP:46657/
```

## 账号申请

```
#另起一个命令行
#查看节点情况
gaiacli status
{
  "node_info": {
    "id": "14a44f8842e8a509a8fa81b5dd6d1e4a81ede76a",
    "listen_addr": "172.16.172.100:46656",
    "network": "gaia-5001",
    "version": "0.19.6",
    "channels": "402022223303800",
    "moniker": "node01",
    "other": [
      "amino_version=0.9.9",
      "p2p_version=0.5.0",
      "consensus_version=v1/0.2.2",
      "rpc_version=0.7.0/3",
      "tx_index=on",
      "rpc_addr=cp://0.0.0.0:46657"
    ],
    "sync_info": {
      "latest_block_hash": "",
      "latest_app_hash": "",
      "latest_block_height": 0,
      "latest_block_time": "1970-01-01T00:00:00Z",
      "syncing": "true"
    },
    "validator_info": {
      "address": "0DF31C99A66A8502DF3EC6119B34CAC8BC966F0A",
      "pub_key": {
        "type": "AC26791624DE60",
        "value": "UTPQHHJHJWeV3qU/iadXhv0d2uRFG7jRchfVYIYq+w="
      },
      "voting_power": 100
    }
  }
}

#新增dc01
gaiacli keys add dc01
NAME: ADDRESS: PUBKEY:
dc01 C34E9EBFF95440DE8C597F30D609EE32807D142C 1624DE62209EE2218C2A639E23F117FC549DDD11900C7CE6EF238
FF9846930B283DF5D1FA4

#查看
gaiacli keys show dc01

#查看key列表
gaiacli keys list
dc01 C34E9EBFF95440DE8C597F30D609EE32807D142C 1624DE62209EE2218C2A639E23F117FC549DDD11900C7CE6EF238
FF9846930B283DF5D1FA4

#查看validator地址
gaiad show_validator
1624DE62205133D01C7247256795DEA53F89A75786F39DDAE4451BB8D17217D5608D58ABEC

#新增dc01
gaiacli keys add dc02
NAME: ADDRESS: PUBKEY:
dc02 E2F9D0D1E965125084F20FD79E17BFE1D40347DE 1624DE6220F50EDF6E1D7BD7493CF4CA7C727AC9D091C8EB7C32C
45C98449EE4C73F6B14B2
```

## 申请代币

可以用Riot chat申请代币：

<https://riot.im/app/#/room/#cosmos:matrix.org> 中询问 @adrian:matrix.org 获得steak代币。

网站申请代币：

<https://faucet.adrianbrink.com>输入地址来领取代币(文档完成时还未开放)。

## 转账测试

```
#查看账号情况
gaiacli account C34E9EBFF95440DE8C597F30D609EE32807D142C
```

```

gaiacli account E2F9D0D1E965125084F20FD79E17BFE1D40347DE

#dc01转账给dc02
gaiacli send --amount=1000fermion --chain-id=gaia-5001 --sequence=1 --name=dc01 --to=E2F9D0D1E965125084F20FD7
9E17BFE1D40347DE

#查看账号情况
gaiacli account C34E9EBFF95440DE8C597F30D609EE32807D142C
gaiacli account E2F9D0D1E965125084F20FD79E17BFE1D40347DE

#查看150000高度时账号情况
gaiacli account C34E9EBFF95440DE8C597F30D609EE32807D142C --block=150000

```

## 成为Validator

```

#查看地址
gaiad show_validator
1624DE6220B83500DC8637F3F4591EE5042B92E7492F7F9C2CB0584E21A60C615EA0134213

#申请成为Validator
gaiacli declare-candidacy --amount=500steak --pubkey=1624DE62209EE2218C2A639E23F117FC549DDD11900C7CE6EF238FF9
846930B283DF5D1FA4 --address-candidate=DA1E912BEE44F4ED30C893C93BC153F3411EA270 --moniker=satoshi --chain-id=
gaia-5001 --sequence=1 --name=<key_name>

#修改信息
gaiacli edit-candidacy --details="my cosmos" --website="https://cosmos.network"

#检查配置信息
gaiacli candidate --address-candidate=DA1E912BEE44F4ED30C893C93BC153F3411EA270 --chain-id=gaia-5001

#确认已经成为candidacy
gaiacli validatorset

```

## 投票权委托

```

#委托
gaiacli delegate --amount=10steak --address-delegator=DA1E912BEE44F4ED30C893C93BC153F3411EA270 --address-cand
idate=<bonded_validator_address> --name=<key_name> --chain-id=gaia-5001 --sequence=1

#取消委托
gaiacli unbond --address-delegator=DA1E912BEE44F4ED30C893C93BC153F3411EA270 --address-candidate=<bonded_vali
dator_address> --shares=MAX --name=<key_name> --chain-id=gaia-5001 --sequence=1

#查看投票权变化
gaiacli account DA1E912BEE44F4ED30C893C93BC153F3411EA270

#查看委托情况
gaiacli delegator-bond --address-delegator=DA1E912BEE44F4ED30C893C93BC153F3411EA270 --address-candidate=<bond
ed_validator_address> --chain-id=gaia-5001

```

## IPFS

项目	内容
链名称	IPFS
链类型	公链
匿名性	匿名
共识机制	PoSt: Proof of Storage (Proof-of-Replication + Proof-of-Spacetime)
开发语言	Go/JS
合约语言	---
官方网站	<a href="https://ipfs.io/">https://ipfs.io/</a>
白皮书	<a href="https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf">https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf</a>
项目地址	<a href="https://github.com/ipfs/go-ipfs">https://github.com/ipfs/go-ipfs</a>

### 项目特点：

IPFS本身其实不是一个区块链项目，只是一个分布式存储。说白了大家可以将它看作是一个有版本控制的BT软件，并提供了文件系统支持。IPFS本身想用来替代HTTP或CDN等，但当前情况下，其性能应该很难达到这个目标。

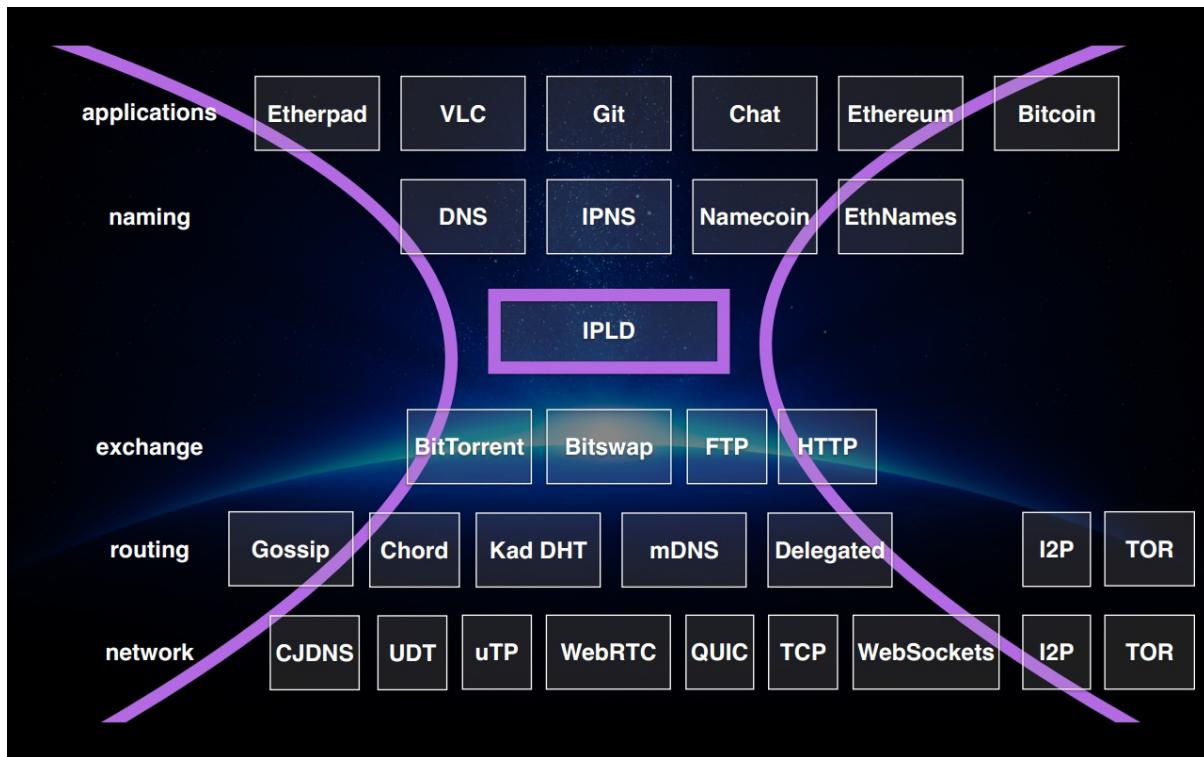
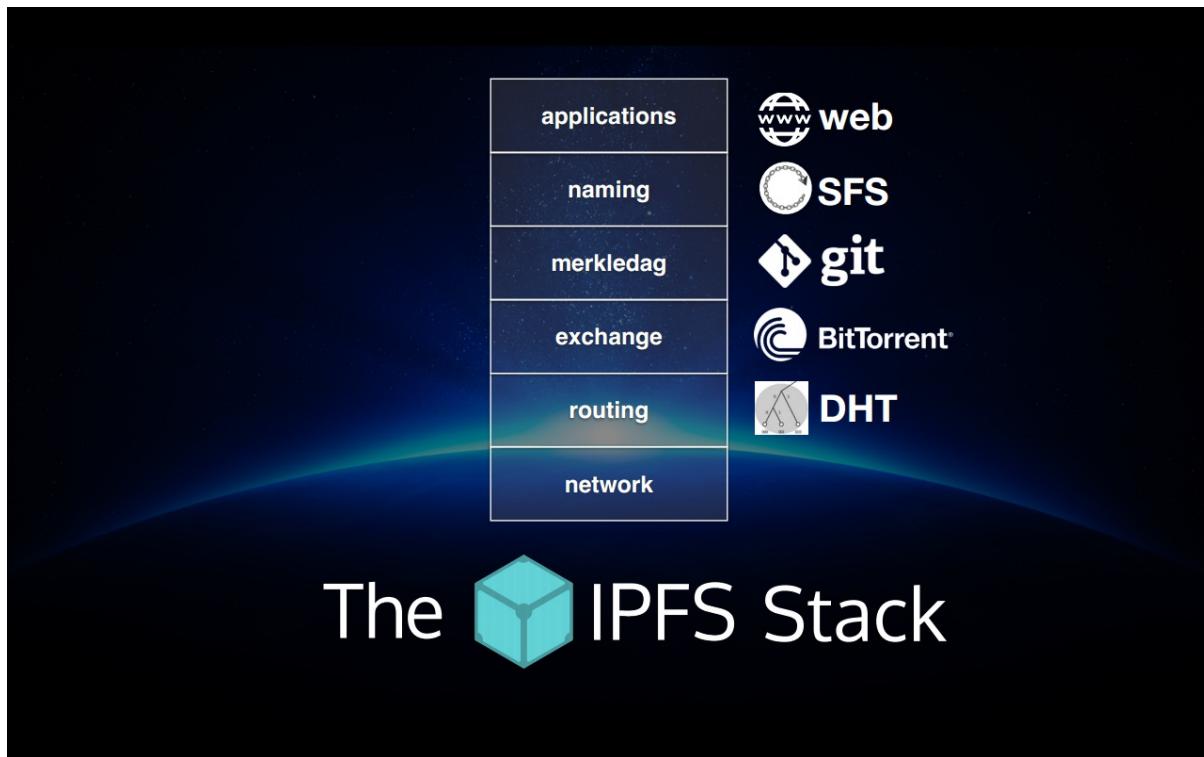
IPFS最牛的地方，也是问题最大的地方，就是数据上传后，在公共IPFS网络中，是没有办法删除的。在提供了很高开放性的同时，也会给自己带来无尽的麻烦。

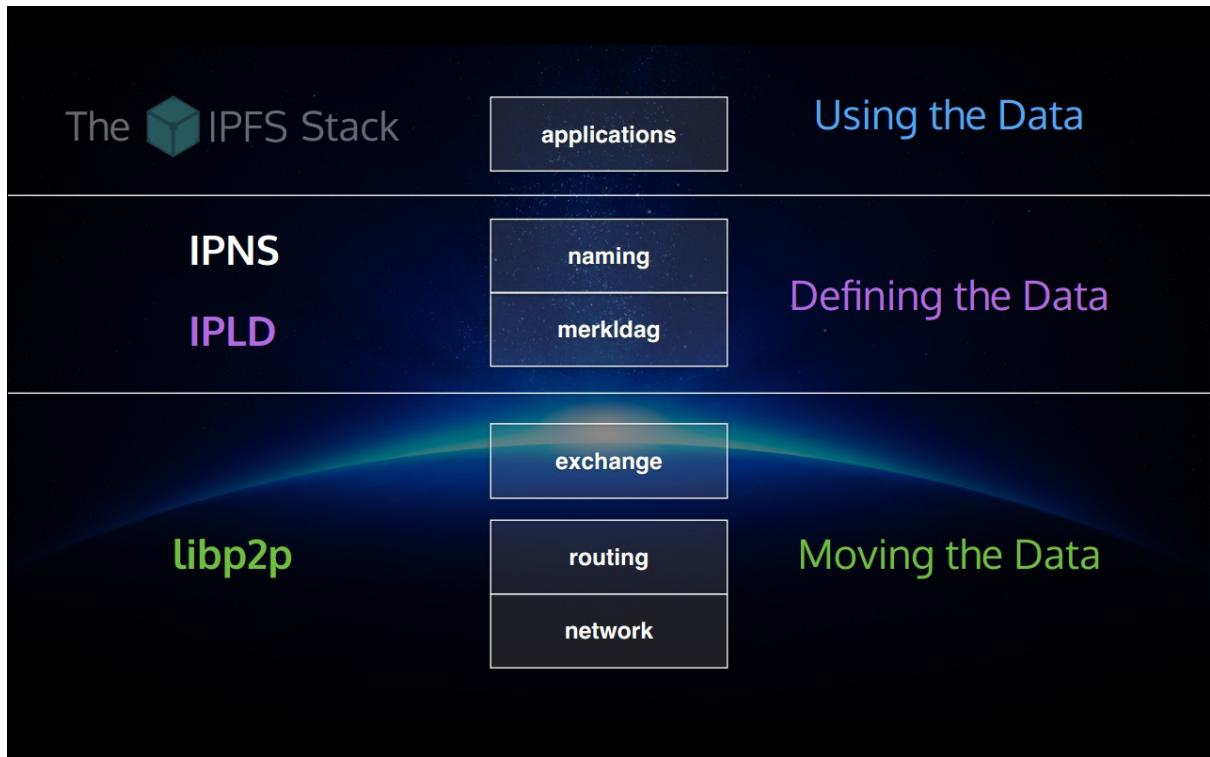
IPFS项目现在还没有完成，比如其文件加密等功能还都没有放出。

IPFS本身也提供了Filecoin这种代币，用来激励各存储节点，同样这个功能也暂时没有开放。

## 整体架构

### IPFS架构

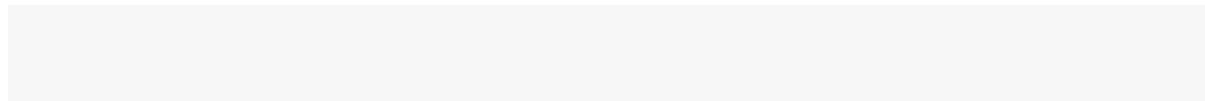




# 使用

## 下载及安装

### 下载到本地



### 解压及安装

```
#解压及安装
tar -zxf go-ipfs_v0.4.14_linux-amd64.tar.gz
cd go-ipfs/
./install.sh

#测试
ipfs help
USAGE
 ipfs - Global p2p merkle-dag filesystem.

 ipfs [--config=<config> | -c] [--debug=<debug> | -D] [--help=<help>] [-h=<h>] [--local=<local> | -L] [--api
=<api>] <command> ...

SUBCOMMANDS
BASIC COMMANDS
  init      Initialize ipfs local configuration
  add <path> Add a file to IPFS
  cat <ref>   Show IPFS object data
  get <ref>   Download IPFS objects
  ls <ref>    List links from an object
  refs <ref>  List hashes of links from an object

DATA STRUCTURE COMMANDS
  block     Interact with raw blocks in the datastore
  object    Interact with raw dag nodes
  files     Interact with objects as if they were a unix filesystem
  dag       Interact with IPLD documents (experimental)

ADVANCED COMMANDS
  daemon    Start a long-running daemon process
  mount     Mount an IPFS read-only mountpoint
  resolve   Resolve any type of name
  name      Publish and resolve IPNS names
  key       Create and list IPNS name keypairs
  dns       Resolve DNS links
  pin       Pin objects to local storage
  repo      Manipulate the IPFS repository
  stats     Various operational stats
  p2p      Libp2p stream mounting
  filestore Manage the filestore (experimental)

NETWORK COMMANDS
  id       Show info about IPFS peers
  bootstrap Add or remove bootstrap peers
  swarm    Manage connections to the p2p network
```

```

dht      Query the DHT for values or peers
ping     Measure the latency of a connection
diag     Print diagnostics

TOOL COMMANDS
config   Manage configuration
version   Show ipfs version information
update    Download and apply go-ipfs updates
commands  List all available commands

Use 'ipfs <command> --help' to learn more about each command.

ipfs uses a repository in the local file system. By default, the repo is
located at ~/.ipfs. To change the repo location, set the $IPFS_PATH
environment variable:

export IPFS_PATH=/path/to/ipfsrepo

EXIT STATUS

The CLI will exit with one of the following values:

0      Successful execution.
1      Failed executions.

```

## 编译及安装

请参考后面对于的章节。

## 初始化节点

```

#初始化节点
ipfs init
initializing IPFS node at /home/hiup/.ipfs
generating 2048-bit RSA keypair...done
peer identity: QmXhJw36JVpGNxiLK7Vxim6bJ2Qj2mPB8AdvBZ7u8HRTyE
to get started, enter:

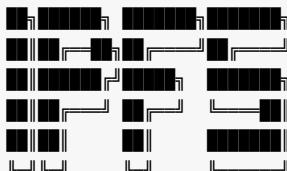
ipfs cat /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv/readme

```

```

#查看信息
ipfs cat /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv/readme
Hello and Welcome to IPFS!

```



If you're seeing this, you have successfully installed  
IPFS and are now interfacing with the ipfs merkledag!

---

Warning:	
This is alpha software. Use at your own discretion!	

```
| Much is missing or lacking polish. There are bugs. |
| Not yet secure. Read the security notes for more. |
```

Check out some of the other files in this directory:

```
./about
./help
./quick-start      <-- usage examples
./readme          <-- this file
./security-notes
```

## 开启服务

```
ipfs daemon &
[1] 4546
Initializing daemon...
Successfully raised file descriptor limit to 2048.
Swarm listening on /ip4/10.0.3.15/tcp/4001
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/172.16.172.71/tcp/4001
Swarm listening on /ip4/172.17.0.1/tcp/4001
Swarm listening on /ip4/172.18.0.1/tcp/4001
Swarm listening on /ip4/172.19.0.1/tcp/4001
Swarm listening on /ip6::1/tcp/4001
Swarm listening on /p2p-circuit/ipfs/QmXhJw36JVpGNxiLK7Vxim6bJ2Qj2mPB8AdvBZ7u8HRTyE
Swarm announcing /ip4/10.0.3.15/tcp/4001
Swarm announcing /ip4/127.0.0.1/tcp/4001
Swarm announcing /ip4/172.16.172.71/tcp/4001
Swarm announcing /ip4/172.17.0.1/tcp/4001
Swarm announcing /ip4/172.18.0.1/tcp/4001
Swarm announcing /ip4/172.19.0.1/tcp/4001
Swarm announcing /ip6::1/tcp/4001
API server listening on /ip4/127.0.0.1/tcp/5001
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
```

## 查看到的节点

```
ipfs swarm peers
/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbtfsmvsqQLuvuJ
/ip4/104.236.151.122/tcp/4001/ipfs/QmSoLju6m7xTh3DuokvT3886QRYqxAzb1kShaanJgW36yx
...
```

## 上传文件

```
ipfs add heart.png
added QmWnt2azBmxL3Q6bZ9YpbnwHkreYbRoDzoRCKgX8HmFN6x heart.png
#注意返回的哈希值，文件地址为
#/ipfs/QmWnt2azBmxL3Q6bZ9YpbnwHkreYbRoDzoRCKgX8HmFN6x
```

## 下载文件

```
#用另外一台设备，完成IPFS安装、初始化并开启服务

#下载文件
ipfs get /ipfs/QmWnt2azBmxL3Q6bZ9YpbnwHkreYbRoDzoRCKgX8HmFN6x
Saving file(s) to QmWnt2azBmxL3Q6bZ9YpbnwHkreYbRoDzoRCKgX8HmFN6x
 4.63 KB / 4.63 KB [=====] 100.00% 0s

#文件重命名
mv QmWnt2azBmxL3Q6bZ9YpbnwHkreYbRoDzoRCKgX8HmFN6x heart_from_ipfs.png
```

## 完 结

## 编译(MacOS)

### 安装go等必须的软件

```
brew install go
```

### 设置gopath环境变量

```
#建立gopath目录  
#mkdir PAHT_TO_GOPATH  
  
#导出环境变量  
export gopath=PAHT_TO_GOPATH  
  
#设置必要的代理信息  
export http_proxy=xxxx:xx  
export https_proxy=xxxx:xx  
export no_proxy="localhost, 127.0.0.1, 192.168.99.100, 根据你自己的实际情况进行处理"
```

### 下载源码

```
mkdir PAHT_TO_GOPATH/src/github.com/ipfs/  
cd PAHT_TO_GOPATH/src/github.com/ipfs/  
git clone https://github.com/ipfs/go-ipfs.git
```

### 编译go-ipfs

```
cd go-ipfs  
make  
  
#会发现, gx无法获取所需的文件, 原来是gx不会默认读取代理配置
```

### 下载gx代码

```
#然后在gopath下创建下面的目录  
PAHT_TO_GOPATH/src/github.com/whyrusleeping/  
cd PAHT_TO_GOPATH/src/github.com/whyrusleeping/  
git clone https://github.com/whyrusleeping/gx.git  
git clone https://github.com/whyrusleeping/gx-go.git  
  
cd PAHT_TO_GOPATH/src/github.com/ipfs/  
git clone https://github.com/ipfs/go-ipfs-api.git
```

### 修改go-ipfs-api代码, 文件shell.go

```
func NewShell(url string) *Shell {  
    c := &gohttp.Client{  
        Transport: &gohttp.Transport{
```

```
        Proxy: gohttp.ProxyFromEnvironment, //请增加这一行
        DisableKeepAlives: true,
    },
}

return NewShellWithClient(url, c)
}
```

## 编译gx及gx-go

```
cd PAHT_TO_GOPATH/src/github.com/whyrusleeping/gx
make

cd PAHT_TO_GOPATH/src/github.com/whyrusleeping/gx-go
make
```

用编译好的go及go-gx替换PAHT\_TO\_GOPATH/src/github.com/ipfs/go-ipfs/bin目录下对应的文件

## 编译ipfs

```
cd PAHT_TO_GOPATH/src/github.com/ipfs/go-ipfs
make
```

## 完结

# 私链搭建

## 生成私链的Swarm密钥

### 下载并编译工具

```
go get -u github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-swarm-key-gen
```

### 生成Swarm密钥

```
./ipfs-swarm-key-gen > swarm.key
```

### 在全部三个节点运行以下内容

```
#  
  
#解压并安装  
tar -zxf go-ipfs_v0.4.14_linux-amd64.tar.gz  
cd go-ipfs/  
sudo ./install.sh  
  
#初始化节点  
ipfs init  
initializing IPFS node at /home/hiup/.ipfs  
generating 2048-bit RSA keypair...done  
peer identity: QmUYmog2D8gt7WCGUg76MZKESGSyZdSwwDPYauaFETjppJ  
to get started, enter:  
  
ipfs cat /ipfs/QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv/readme  
  
#部署swarm.key  
cp swarm.key ~/.ipfs  
  
#移除默认bootstrap节点  
ipfs bootstrap rm --all  
/dnsaddr/bootstrap.libp2p.io/ipfs/QmNnooDu7bfjPFoTZYxMNLWUQJyrVwtbZg5gBMjTezGAJN  
/dnsaddr/bootstrap.libp2p.io/ipfs/QmQCU2EcMqAqQPR2i9bChDtGNJchTbq5TbXJJ16u19uLTa  
/dnsaddr/bootstrap.libp2p.io/ipfs/QmbLHAnMoJPWSCR5Zhtx6BHJX9KiKNN6tpvbUcqanj75Nb  
/dnsaddr/bootstrap.libp2p.io/ipfs/QmcZf59bWwK5XF176CZX8cbJ4BhTza3gU1ZjYzcYW3dwt  
/ip4/104.131.131.82/tcp/4001/ipfs/QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbdUtfsmvsqQLuvuJ  
/ip4/104.236.179.241/tcp/4001/ipfs/QmSoLPppuBtQSGwKDZT2M73ULpjvfd3aZ6ha4oFGL1KrGM  
/ip4/104.236.76.40/tcp/4001/ipfs/QmSoLV4Bbm51jm9C4gDYZQ9Cy3U6aXMJDAbzgu2fzaDs64  
/ip4/128.199.219.111/tcp/4001/ipfs/QmSoLSafTMBsPKadTEgaXctDQVcqN88CNLHXMkTNwMKPnu  
/ip4/178.62.158.247/tcp/4001/ipfs/QmSoLer265NRgSp2LA3dPaeykiS1J6D1fTC88f5uVQKNAd  
/ip6/2400:6180:0:d0::151:6001/tcp/4001/ipfs/QmSoLSafTMBsPKadTEgaXctDQVcqN88CNLHXMkTNwMKPnu  
/ip6/2604:a880:1:20::203:d001/tcp/4001/ipfs/QmSoLppuBtQSGwKDZT2M73ULpjvfd3aZ6ha4oFGL1KrGM  
/ip6/2604:a880:800:10::4a:5001/tcp/4001/ipfs/QmSoLV4Bbm51jm9C4gDYZQ9Cy3U6aXMJDAbzgu2fzaDs64  
/ip6/2a03:b0c0:0:1010::23:1001/tcp/4001/ipfs/QmSoLer265NRgSp2LA3dPaeykiS1J6DifTC88f5uVQKNAd
```

### 在节点2、3将节点1添加为bootstrap节点

```
ipfs bootstrap add /ip4/172.16.172.81/tcp/4001/ipfs/QmUYmog2D8gt7WCGUg76MZKESGSyZdSwwDPYauaFETjppJ
```

## 开启三个节点

```
ipfs deamon &
```

## 在节点2上传文件

```
ipfs add heart.png
added QmWnt2azBmxL3Q6bZ9YpbnwHkreYbRoDzoRCKgX8HmFN6x heart.png
```

## 在节点3下载文件

```
ipfs get /ipfs/QmWnt2azBmxL3Q6bZ9YpbnwHkreYbRoDzoRCKgX8HmFN6x
```

然后呢？

Learn, practice and share.