**Bitcoin**

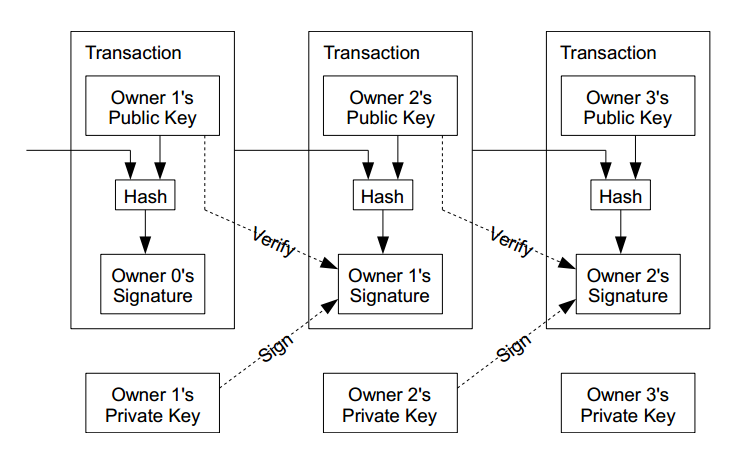
1. 从技术上来说，[bitcoin](https://so.csdn.net/so/search?q=bitcoin&spm=1001.2101.3001.7020) 是一个点对点的电子现金系统，它可以实现分布式的记账，不依赖中心化的账务系统（比如银行，支付宝），它的账本完整的分布在每一个比特币的客户端，并且实时同步。
2. 比特币的目的：我们需要的是一个基于加密证明而不是信任的电子支付系统，允许任何两个当事方直接相互交易，而不需要可信的第三方。

收款方不能确定coin是否被double-spending—>让收款人知道以前的所有者没有签署任何早期的交易—>收款方需要了解所有交易—>没有可信第三方的前提下，应将所有交易公开—>需要一个系统让参与者对历史达成一致

使用peer-to-peer网络解决double-spending 问题（双花攻击）

双花攻击：“双花”是指在区块链系统中的一种欺诈行为，也被称为“双重支付”。它指的是一个人或组织在通过区块链进行交易时，同时向不同的接收方发送相同的数字资产，以获得更多的利益。比特币是第一个解决双重支付问题的分布式加密货币

1. 如何解决double-spending问题：using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions
2. 安全性：诚实节点控制比攻击者更多的CPU资源
3. 电子货币：数字签名链



1. 交易的签名包括：(前一个交易+下一个拥有者公钥)的hash值，组成该用户的签名
2. Timestamp server
3. 在peer-to-peer的基础上实现一个分布式timestamp server：使用proof-of-work系统
4. Proof-of-work：其核心思想在于，通过一种对CPU运算资源上“证明者(Prover)低效，验证者(Verifier)高效”（很难解决，但容易验证）的算法，达到证明者向验证者证明其对特定量的计算资源的投入的目的
5. Bitcoin中的proof-of-work：寻找一个值使得block的哈希有需要的0位
6. 比特币是一种数字货币，它的产生依赖于“挖矿”（Bitcoin mining）这一过程。挖矿是指通过计算机算力解决一系列数学难题，从而获得比特币的过程。这个难题的解决方案是找到一个适当的随机数，使块的哈希值小于目标值Z，称为PoW。当前难题的多个解决方案可能导致区块链分叉。为了解决这个问题，所有矿工都遵循最长的链来解决谜题，因此，区块链最终可以以高概率达到全局一致性
7. 比特币工作量证明难题的解决没有内在价值，这种计算(通常被称为“挖掘”)是昂贵的，对环境不友好的，除了验证事务之外没有任何用途。
8. 比特币首先提出了中本共识，该共识将工作量证明应用于区块链结构，以建立一个无需信任的附加分类账。中本共识解决了公共网络的分布式一致性问题，但浪费了太多的计算能力
9. 比特币挖矿的计算密集型性质是一个关键属性，确保所有矿工都被激励将他们的挖矿工作集中在一条链上，从而达成共识。
10. 区块链是一种去中心化数据库技术
11. 把区块链的共识机制和fuzzing test结合有什么用呢？比单独使用fuzzing test好的地方在哪里？是为了产生一个分布式fuzzing test的数据库并有一个激励机制吗？
12. 怎么才算挖掘新的区块？又怎么添加到主链后面

区块头：

1. 当前区块hash值
2. 前一个区块hash值
3. 时间戳：区块产生的近似时间
4. Merkle根：该区块中交易的merkle树根的hash值
5. 难度目标：该区块工作量证明算法的难度目标
6. Nonce

新区块验证过程：

1. 判断当前链的最后一个区块的hash值是否和新区块的preHash相等
2. 验证新区块hash值的正确性：计算当前区块的hash值（preHash、当前块交易、nonce），判断是否和新区块hash值相等

# Bitcoin源码解读v0.0.1

## Main.h

* Cblock：区块类。Block被添加到磁盘上的blk0001.dat等文件中，这些文件的位置由内存中的CblockIndex对象索引到。

属性：

1. Blcok header包括nVersion、hashPrevBlock、hashMerkleRoot、nTime、nBits、nNonce
2. Transactions
3. vMerkleTree

* CblockIndex：表示一个BlockChain的前后连接关系（待确认）

**Bicoin源码**

**net\_processing.cpp**

ProcessMessage() ：处理别的节点发送的消息的方法，以msg\_type判断，bitcoin定义了很多消息类型

Msg\_type: GETHEADERS —— 获取最新区块头

GETBLOCKS ——发送给请求者所请求的区块后的链上其余区块

**validation.cpp**

ProcessNewBlock()：调用CheckBlock()校验区块合法性，如果合法性校验通过，调用AcceptBlock()将区块存入磁盘

AcceptBlock()中存入磁盘前检查的步骤：

1. 主链加锁，保证线程安全
2. 校验区块头的有效性
3. 检查区块索引
4. 检查工作量：判断当前区块的工作量是否大于或等于活动链顶的工作量。如果当前没有活动链顶，则默认为 true。
5. **检查高度是否过高**: 判断当前区块的高度是否超过活动高度加上保留的最小区块数 MIN\_BLOCKS\_TO\_KEEP。

解释：活动高度是指当前节点认为的最长有效链的最新区块的高度。检查高度的目的：**防止过早接收区块**: 如果一个区块的高度远高于当前活动高度，可能意味着该区块是在一个不太可能很快连接到当前链的分支上。这可能会导致网络中的资源浪费，因为节点可能会接收到许多与当前链不相干的区块。**保持链的连贯性**: 通过限制处理过高的区块，节点可以确保只处理与当前链相关的区块，避免处理那些在链中可能不会被接受的区块。

MIN\_BLOCKS\_TO\_KEEP 的作用：

**保留的最小区块数**: MIN\_BLOCKS\_TO\_KEEP 是一个常量，表示在活动链和当前区块之间允许的最大高度差。这个数值的存在是为了考虑到网络延迟或暂时的链分叉情况。加上保留的最小区块数的原因：允许在活动链高度之上保留一定数量的区块，可以帮助节点在面对网络波动或短暂的分叉时，仍能有效地接收和处理新区块。例如，如果当前活动高度是 100，MIN\_BLOCKS\_TO\_KEEP 为 10，那么允许的最高区块高度为 110。如果接收到的区块高度为 111，则此区块被认为过高，节点将忽略它。

1. **已有区块处理**: 如果已经存在该区块，返回 true，表示无需处理。

VerifyDB(): 有level0 - level4这5个校验等级

**rpc/mining.cpp**

bool GenerateBlock() 挖矿代码 默认最大尝试次数为1000000，nNonce最大值为

generateBlock()：rpc请求， 该方法调用GenerateBlock

generatetoaddress()：用于在指定地址上挖掘新区块并返回生成的区块哈希，该方法调用generateBlocks()方法

随着专用矿机的出现，CPU 挖矿基本已经被废弃了，这里的 CPU 挖矿代码只是用来学习以及测试用，实际矿工并不会使用下面的代码来挖矿。开启 CPU 挖矿的话，可通过一个 generatetoaddress RPC 调用开启挖矿到一个特定的地址。

由于 nonce 只有 32 位，在用尽所有 nonce 可能性（约 40 亿）后，挖矿硬件会更改区块头（调整 coinbase 额外的 nonce 空间、版本位或时间戳）并重置 nonce 计数器，测试新的组合。

submitblock()：rpc请求，提交新区块到区块链网络

getnetworkhashps()：rpc请求，返回估计的每秒网络hash处理数，基于上n个块

generatetodescriptor()：rpc请求

getmininginfo()：rpc请求，返回挖矿信息的json对象

在实际的比特币挖矿中，矿工通常使用专用的挖矿软件与矿池进行交互，而不是直接调用比特币节点的 RPC 方法。以下是一些常见的挖矿调用和相关的方法：

### 1. submitblock

* 功能：当矿工成功挖掘出一个符合难度要求的区块时，矿工会使用 submitblock RPC 方法将该区块提交到比特币节点。
* 用法：
  + 矿工将构建好的区块数据（包括区块头和交易）作为参数传递给 submitblock。
  + 比特币节点会验证区块的有效性，并将其添加到区块链中。

### 2. getblocktemplate

* 功能：矿工可以使用 getblocktemplate RPC 方法获取一个新的区块模板，这个模板包含了当前可以用于挖矿的交易信息和难度目标。
* 用法：
  + 矿工向节点请求区块模板，节点会返回当前的区块头信息和包含的交易数据。
  + 矿工基于这个模板进行挖矿，尝试找到一个有效的工作量证明（Proof of Work）。

### 3. generatetoaddress（仅用于测试）

* 功能：如前所述，generatetoaddress RPC 方法可以启动 CPU 挖矿，但这主要用于测试和开发目的，而不是实际挖矿。
* 用法：
  + 矿工可以指定生成新区块的数量和接收奖励的地址。

### 4. 连接到矿池

* 矿池交互：大多数矿工选择加入矿池，而矿池通常有自己的协议（如 Stratum 协议）来进行挖矿。矿工将通过矿池提供的挖矿软件与矿池进行交互，提交工作和接收任务。
  + 挖矿软件：矿工使用如 CGMiner、BFGMiner 等挖矿软件，软件会处理与矿池的通信，包括获取工作、提交结果等。

实际的挖矿过程中，矿工主要使用 getblocktemplate 来获取区块模板，并使用 submitblock 将挖掘的区块提交给比特币节点。大多数矿工会通过矿池进行挖矿，使用矿池的挖矿软件和协议来完成这些操作。

**rpc/blockchain.cpp**

getDifficulty(), 获取难度

getblockfrompeer()：rpc请求，参数是区块hash和peer\_id

getblockhash()，rpc请求，获取某一高度的区块的hash值

getblockheader()，rpc请求

CheckBlockDataAvailability：用于检查特定区块的数据可用性，确保所请求的数据（区块或撤销数据）是否存在

getblock\_vin: 结构定义了比特币区块中交易输入的详细信息，提供了有关输入的前一个输出、其状态和相关脚本的信息

getblock: rpc请求，用于获取指定区块的信息

GetPruneHeight(): 用于获取已经被修剪的最高区块的高度，或者在没有任何区块被修剪的情况下返回 std::nullopt

注：修剪区块（Pruned Blocks）是指在区块链中，某些区块的数据被删除或不再存储，以节省磁盘空间。特别适用于希望参与区块链网络但又不具备足够存储能力的用户或设备（如移动设备、嵌入式系统等）。它们仍然能够验证新交易和区块的有效性，但无法提供完整的历史记录。

pruneblockchain():rpc请求，旨在修剪区块链，以减少存储需求。

GetUTXOStats：用于计算未花费交易输出集（UTXO）的统计信息。

UTXO（Unspent Transaction Output）:

当新区块被添加到链中时，在这个区块中的所有交易都从”unconfirmed”状态变为“confirmed”状态，bitcoin中存放在块中的交易数量不固定，不过区块大小需要小于1MB。

UTXO是比特币交易的基本单位。在比特币系统中,余额的概念是通过聚合属于某个地址的所有UTXO得出的,并不是一个独立的账户。每一笔比特币交易都由若干输入(Input)和输出(Output)组成。输入来自之前交易的UTXO,输出会产生新的UTXO。一个UTXO就像一张不能分割的钞票,要么全部花掉,要么就是找零产生新的UTXO。UTXO的面值可以是任意的。还没有被用作新交易输入的输出,就是UTXO,代表着某个地址当前可以支配的比特币。所有UTXO的总和等于比特币总供应量。UTXO模型是比特币独特的设计,不同于[以太坊](https://so.csdn.net/so/search?q=%E4%BB%A5%E5%A4%AA%E5%9D%8A&spm=1001.2101.3001.7020)等使用账户余额模型的区块链。它是比特币多重支付保护和验证机制的基础。

所以,UTXO作为比特币的核心概念,代表着可以支配的、未花费的比特币。

gettxoutsetinfo(): rpc请求，返回UTXO统计信息

gettxout()：rpc请求，返回一个UTXO的具体信息

verifychain()：rpc请求，验证区块链完整性，这个命令适合需要进行区块链数据完整性检查的场景，例如在节点同步过程中或在怀疑区块链数据不一致时使用。该方法调用VerifyDB()

SoftForkDescPushBack()：是一个用于构建和收集软分叉信息的实用工具，特别是针对埋藏部署的软分叉。它通过检查部署状态、构建 JSON 对象并将其添加到结果集中，使得区块链的状态描述更加清晰。这个函数在区块链节点需要报告其支持的软分叉时非常有用，帮助用户理解当前网络的协议状态。

invalidateblock()：rpc请求，永久标记某个区块为不合理的

reconsiderblock()：rpc请求，重新考虑某区块

getchaintxstats()：rpc请求，计算链中交易的全部数量和速率

getblockstats()：rpc请求，允许用户获取特定区块的详细统计信息

scantxoutset()：rpc请求，用于扫描未花费交易输出集（UTXO）以查找匹配特定输出描述符的条目

getchainstats()：rpc请求

**chain.cpp**

FindFork()

FindEarliestAtLeast(): 用于在区块链中查找符合特定时间和高度条件的区块。通过使用二分查找的方式，函数能快速定位到合适的区块，优化了查找效率。这在区块链节点的同步、查询和验证过程中非常重要，可以帮助节点快速定位到相关区块。

GetAncestor()

GetBlockProof(block): 计算某个区块的工作量证明

**pow.cpp**

GetNextWorkRequired()：计算下一个工作量证明的难度

PermittedDifficultyTransition()：检查难度调整是否超过限制

**node/miner.h**

BlockAssembler：是一个类，负责构建新的区块模板，供矿工用于创建有效的区块。主要任务是将交易组装成新的区块模板。它考虑了多种因素，如区块大小限制、交易费用以及交易内存池（mempool）的当前状态。

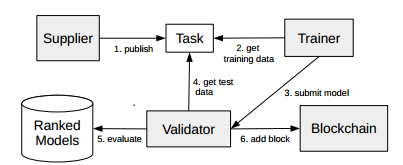
**node/miner.cpp**

BlockAssembler::CreateNewBlock 是一个用于构建新的区块模板的方法。这个方法将组装一个新区块，包括创建一个虚拟的 coinbase 交易，并从内存池中选择符合条件的交易。

BlockAssembler::addPackageTxs 方法用于从内存池中选择交易并将其添加到正在构建的区块中。这个方法通过分析交易的费用和祖先关系来确定哪些交易应该被包含在内。

**Proof-of-Learning**

1. WekaCoin：一种p2p加密货币，基于一种新型分布式共识协议proof-of-learning。该共识协议通过对给定任务的机器学习系统进行排名来达到分布式共识
2. proof-of-learning的目的：减轻基于hash的难题的计算浪费，创建一个公共的、可验证的最先进的机器学习模型和实验的分布式数据库
3. WekaCoin节点在一个点对点网络中运行，在这个网络中，一组被称为“trainer”的节点为之前由网络中被称为“supplier”的其他节点发布的任务提交机器学习模型。“validator”是网络中随机选择的节点，负责对模型进行排名并向链加入新的区块。整个过程用于验证区块链中的交易块



拥有最好的模型的trainer被supplier奖励WekaCoin（这个WekaCoin是可以在市场上交易的吗？）

1. 每个块包含 ，其中包含用于验证区块的机器学习竞争的元数据。
2. 验证WekaCoin交易的过程称为Proof-of-learning
3. WekaCoin中包含三种交易：这些交易都是添加到最长的那条链上吗？交易和区块的关系
4. 标准交易
5. 任务发布交易

Training dataset(trd)、test dataset(ted)、testing release block height(tbh)、evaluation script(es)、baseline model(bmod)、baseline performance score(bscore)

1. 模型交易
2. 恶意的supplier可能会参与自己发布的竞争，供应商将需要对网络进行大量控制才能从这种行为中获得奖励
3. 防止恶意节点DoS攻击：trainer需要付参与费
4. 防止trainer抄袭别人的模型：在模型交易阶段只提交模型的hash值
5. 向链中添加新块的过程由随机选择的验证者组成的委员会完成，委员会成员需达成一致
6. 每个验证者从尚未合并到区块链中的交易池中构建候选块，通过模型的hash值下载每个模型进行评估，产生候选排名。被提议的区块和候选排名通过八卦协议发送给其他验证者。
7. 恶意验证者的行为会通过拜占庭协议中的投票机制减轻

**Spacemint**

1. 为了在SpaceMint中挖掘区块，矿工们投入了磁盘空间而不是计算能力，并且投入更多的磁盘空间会产生更高的成功挖掘区块的期望。
2. 为了用PoSpace取代PoW，在区块链上达成共识，必须解决以下问题
3. 交互性
4. 确定胜利者：任何矿工获胜的概率应该与它所投入的空间成正比，我们希望矿工在没有任何交互的情况下了解他是否可能成为赢家
5. Nothing-at-stake problem：当用计算上易于生成的证明(如PoStake或PoSpace)取代PoW时，会出现一系列问题，称为无利害关系问题
6. Nothing-at-stake problem（无利害关系问题，很容易导致区块链分叉）

Nothing At Stake, a situation where someone loses nothing when behaving badly, but stands to gain everything.

问题的本质是 “作恶无成本，好处无限多”

问题：（1）同时在多条链上挖矿（2）尝试用单个proof创建多个不同的块（block grinding）（3）Grinding challenges：矿工通过利用过去和未来区块之间的关系来生成不成比例的高质量区块序列

解决方案：

1. 为了阻止矿工扩展多条链，我们确保检测到这种行为并对其进行惩罚。
2. 将proof和transaction解耦。矿工只能为每个challenge生成一个有效的proof，通过并行运行两条链实现，一条是“proof chain”，另一条是“signature chain”
3. 过去的区块会影响未来区块的短序列的质量

共识系统出现分叉（fork）的情况时，出块节点可以在“不受任何损失”的前提下，同时为多条链出块，从而有可能获得“所有收益”。聪明的出块节点会有动力产生新的分叉，支持或发起不合法交易，其他逐利的出块节点会同时在多条链（窗口）上排队出块支持新的分叉。随着时间的推移，分叉越来越多，非法交易，作恶猖狂。区块链将不再是唯一链，所有出块节点没有办法达成共识。无利害关系问题还让双花攻击更容易。PoW 机制天生避免了这个问题。因为在出块时，矿工会付出机会成本 — 算力资源。如果分叉出现，矿工需要慎重的选择在哪条链上出块，一旦选错，付出的算力成本则没有收益。矿工也不会选择在两条链上均分算力，这样只会将原链的出块概率缩小一半，可能得不偿失。

4、Poof-of-space：两个阶段

Prover P and verifier V

P是普通节点，需要存储较大的信息数据，V存储数据库以及证明人的一小部分存储信息，以便验证

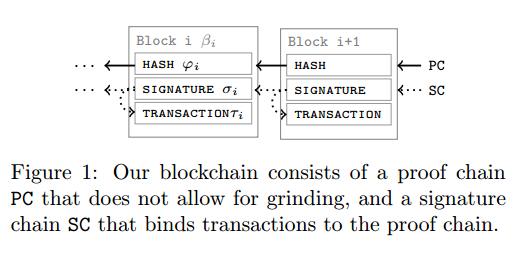
（1）Initialization phase: P stores of size N， contains the labels of all the nodes of G（有向无环图），sends nonce and commitment to V

V stores a short commitment to , is a Merkle-tree commitment to these n labels

（2）Execution phase: V sends a challenge c to P, P returns a short answer a after reading a small fraction of

P picks a graph G depending on the space it wants to dedicate

P then stores a label for each node in G



5、挖矿过程

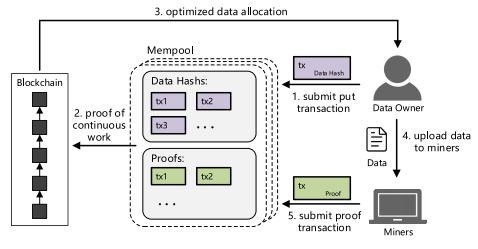
1. Initialization：miner generates (pk,sk), nonce set to pk

Minier stores (,sk), announces (pk, ) to prevent grinding attack

1. Mining

**Proof of continuous work**

1. 许多改进的共识方案通过利用矿工的存储资源来解决这个问题，而不是消耗计算资源。然而，这些方案无法让矿工持续存储数据，通常依赖经销商分配数据，难以构建可靠的分散存储系统。
2. 现有的方案很难建立一个可靠的存储系统。
3. 这些共识算法只关注存储容量。他们缺乏激励机制来鼓励矿工不断存储数据。
4. 这些系统通常依赖于中心化的一方为矿工分配数据,可能导致区块链存储系统弱化为集中式云存储系统
5. 提出了一种具有存储相关激励机制的变种中本共识，以构建基于区块链的存储系统，该系统将数据分配给多个矿工，而无需任何可信的第三方。 矿工可以通过不断提交存储证明来积累挖矿优势。
6. Consistent hash（一致性hash）：用于数据分配。是一种特殊的哈希算法，目的是解决分布式缓存的问题。在移除或者添加一个服务器时，能够尽可能小地改变已存在的服务请求与处理请求服务器之间的映射关系。一致性哈希解决了简单哈希算法在分布式[哈希表](https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8/5981869?fromModule=lemma_inlink)( Distributed Hash Table，DHT) 中存在的动态伸缩等问题。
7. POR机制（2007）：proof of retrievability , 可检索证明，challenge-response protocol，证明者使验证者相信其拥有的文件可以被正确检索。
8. Local POR机制：将其应用于区块链，将POW计算用于数据保存。矿工必须将数据存储在本地，以便按时生成证明以获得奖励，从而限制了矿工的数据外包行为。在我们的系统中，我们将本地POR视为加密原语。矿工用它来计算存储证明。
9. 本系统有三个实体：data owner（在系统中上传和检索数据的用户）、miner（区块链网络中参与挖矿并提供存储）、blockchain（接受交易并更新全局状态的公共分布式账本）



数据拥有者上传数据，数据被分配给指定矿工，矿工将数据存储在本地并生成存储证明。

1. 两种交易：
2. Put transaction：requests leasing storage and decides the length of the lease.
3. Proof transaction：pk表示矿工的公钥来代表他们的身份，我们使用local POR方案中的证明算法来构造存储π的证明，其中涉及随机挑战R。一旦矿工广播此交易，其他人执行验证算法来检查它是否有效。
4. 矿工的收入：数据拥有者给的租赁费和成功挖矿的奖励

**CoinAI**

1. proof-of-useful-work
2. 矿工要解决的问题是人工智能模型的训练，更具体地说，是深度学习架构
3. 要将挖矿方案引入区块链网络，必须遵循以下几个方面：
4. 这个问题必须是复杂的，需要一定的计算量，才能保证矿工进行了一些实际的工作，从而能够获得与区块挖掘相关的奖励。
5. 为了保证区块链的完整性，必须将前一个区块的哈希值作为问题的变量引入。
6. 挖矿方案必须有一个竞争性的组成部分，这样第一个解决问题的矿工(或者相反，提供最佳解决方案的矿工)就是挖出区块并获得奖励的矿工。
7. 给定一个问题解决方案，必须很容易验证该解决方案的有效性并评估其质量。
8. 一旦矿工发现了一个区块并将其附加到区块链中，其他矿工正在挖掘的所有其他潜在区块都必须被丢弃。这保证了矿工不能“保存区块”以供将来发现。
9. Hash-to-architecture mapping：在hash值和有效的深度学习架构设置之间建立满射函数
10. Proof-of-storage：我们建议只有某些区块链节点（称为keepers）存储深度学习模型，keepers提供存储能力，keepers也和矿工一样被奖励
11. 哪些深度学习问题需要被解决：由加密货币持有者决定

