**课程报告**

|  |  |
| --- | --- |
| **课程名称：** | **企业软件项目实训** |
| **学生姓名：** | **郑宜静** |
| **学生学号：** | **201630666516** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2018-2019第二学期** |

**软件学院**

**2019年6月**

1. **区块链技术原理**
   1. **区块链概念**
2. 区块链是一个提供了拜占庭容错、并保证了最终一致性的分布式数据库；
3. 从数据结构上看，它是基于时间序列的链式数据块结构；从节点拓扑上看，它所有的节点互为冗余备份；
4. 从操作上看，它提供了基于密码学的公私钥管理体系来管理账户。
   1. **区块链的7个技术特征**
5. 区块链的存储基于分布式数据库；
6. 数据库是区块链的数据载体，区块链是交易的业务逻辑载体；
7. 区块链按时间序列化区块数据，整个网络有一个最终确定状态；
8. 区块链只对添加有效，对其他操作无效；
9. 交易基于非对称加密的公私钥验证；
10. 区块链网络要求拜占庭将军容错；
11. 共识算法能够“解决”双花问题。
    1. **区块链的类型**
12. 公有区块链：全公开区块结构，网络中每一节点权利均等且可与任意节点交互。
13. 联盟链（也叫许可链）：多中心结构，参与主体、交易由联盟中的成员事先确认。
14. 私有区块链：中心化系统，由中心控制者指定参与人员，只对特定主体开放。
15. 侧链：是在主区块链之外的另一区块链，锚定与主区块链的一个节点，并可以与主区块链进行一定的数据交互。

|  |  |  |  |
| --- | --- | --- | --- |
| 区块链类型 | 私有链 | 联盟链 | 公有链 |
| 参与者 | 个体或公司内部 | 特定人群 | 所有人 |
| 信任机制 | 自行背书 | 集体背书 | POW/POS/DPOS |
| 记账者 | 自拟 | 参与者协商 | 所有人 |
| 激励机制 | 无需 | 可选 | 需要 |
| 中心化程度 | 中心化 | 多中心 | 去中心化 |
| 特性 | 可追溯 | 提升效率 | 信用的自建立 |
| 使用场景 | 审计、发行 | 结算 | 虚拟货币 |
| 代表公司或组织 | Overstock | R3银行联盟 | 比特币和以太坊 |
| 承载能力 | 1000-10万笔/秒 | 1000-1万笔/秒 | 3-10秒 |

* 1. **区块链的核心技术组成**
     1. **P2P网络协议**

1. 网络连接：以太坊 P2P 网络是一个完全加密的网络，提供 UDP 和 TCP 两种连接方式，主网默认 TCP 通信端口是 30303，推荐的 UDP 发现端口为 30301
2. 拓扑结构：全分布式的拓扑结构
   * 1. **分布一致性算法**
        1. **POW**

通常是指在给定的约束下，求解一个特定难度的数学问题，谁解的速度快，谁就能获得记账权（出块）权利。这个求解过程往往会转换成计算问题，所以在比拼速度的情况下，也就变成了谁的计算方法更优，以及谁的设备性能更好。比特币本身的演化很好地诠释了这个问题，中本聪设计的思路本来是由 CPU 计算。随着市场发展，人们发现 GPU 也可以参与其中，而且效率可以达到十倍百倍，现在，这项工作基本以 ASIC 专业挖矿芯片为主。

* + - 1. **POS**

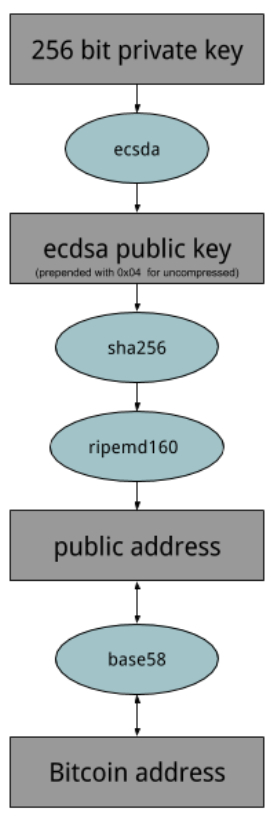
这是一种股权证明机制，它的基本概念是产生区块的难度应该与你在网络里所占的股权（所有权占比）成比例，目前有三个版本 PoS1.0、 PoS2.0、 PoS3.0。它实现的核心思路是：使用你所锁定代币的币龄（CoinAge）以及一个小的工作量证明，去计算一个目标值，当满足目标值时，你将可能获取记账权。

* + - 1. **DPOS**

简单来理解就是将 PoS 共识算法中的记账者转换为指定节点数组成的小圈子，而不是所有人都可以参与记账，这个圈子可能是 21 个节点，也有可能是 101 个节点。这一点取决于设计，只有这个圈子中的节点才能获得记账权。这将极大地提高系统的吞吐量，因为更少的节点也就意味着网络和节点的可控。

* + 1. **加密签名算法**
       1. **哈希算法**

1. 4个特性：原像不可逆、难题友好性、发散性、抗碰撞性
2. 过程：把任意的交易数据做成数据摘要，然后再一个一个链接起来，形成数据块的链式结构。通过验证每个区块间接地验证交易，然后每个交易原数据也可以做成哈希数据摘要，用于验证交易数据的完整性。这种链式结构具备发散传导性，越往历史以前的篡改，越容易导致大面积的影响，这也叫做历史逆向修改困难。
   * + 1. **非对称加密算法**

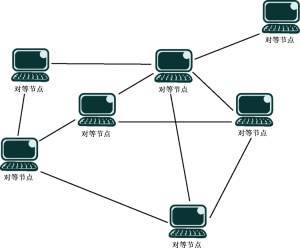


* + 1. **账户与交易模型**

区块链的账本特性，通常分为 UTXO 结构以及基于 Account-Balance 结构的账本结构，我们也称为账本模型。 UTXO 是“unspent transaction input/output”的缩写，翻译过来就是指“未花费的交易输入输出”。

* 1. **区块链运行机制**
     1. **公开记账**

公开的分布式记账，解决了账本的安全性及可信任的问题，使得账本数更据安全，并建立起了信任，让账本更加具备权威性。



企业或者银行，如果能够采用这种公开的(或者内部公开的)分布式记账方式，可以大大解决腐败问题，也可以提高财务数据的安全性，从本质上解决中心化方式带来的弊端。

* + 1. **创建创世区块**

创世区块是区块链里面所有区块的共同祖先，这意味着你从任一区块，循链向后回溯，最终都将到达创世区块。每一个节点都“知道”创世区块的哈希值、结构、被创建的时间和里面的一个交易。因此，每个节点都把该区块作为区块链的首区块，从而构建了一个安全的、可信的区块链的根。

* + 1. **交易**

是指实际上发生区块链移动的交易，只有在用各个用户的私钥签名被指定的用户之间方可进行交易。交易记录被储存到区块内，经过认证过程确认以后不得变更或伪造。为了完成交易，需经过工作量证明得到批准，需要一些时间。

* + 1. **打包Transaction（挖矿）**

有了 Transaction，还需要东西把 Transaction 装起来，区块链用一个特殊的信封把Transaction 装起来，这个信封就是区块链中的“区块”，这个封装过程就是“打包交易”。

* + 1. **广播交易**

交易广播的英文名为Transaction Broadcast，是区块链中与交易过程相关的词汇。将交易信息在区块链网络中“广播”，并由节点验证即确认的过程。

[交易确认](https://baike.baidu.com/item/%E4%BA%A4%E6%98%93%E7%A1%AE%E8%AE%A4/22415245)（Confirmation）表示该笔交易被区块链网络所记录并确认，当交易发生时，记录该笔交易的区块将进行第一次确认，并在该区块之后的链上的每一个区块进行再次确认；当确认数达 到六个及以上时，通常认为这笔交易比较安全并难以篡改。

1. **联盟链和公有链的异同**
   1. **联盟链**

目前大多数人都倾向于认为联盟链是介于公有链与私有链之间的一种链。同时也有人把联盟链归结于‘私有链’的范围。因为联盟链是半公开的，只有预先指定的几个节点才能获取到记账权。其他加入的节点仅有交易权利。联盟链的权限设计较之于‘公有链’会更加的复杂。但是由于其无需竞争记账与全网确认账单的特性下，联盟链的TPS较之于公有链来说比较好一些。

* 1. **公有链**

公有链是指全世界任何人均可读取，确认交易所有人均可参与其共识的区块链。其中的代表为比特币。公有链的节点运行在互联网上,世界上任何一台可以连接互联网的设备均可以对区块链产品进行读写,确权操作。公有链得益于起共识机制 (PoW或PoS)能够共同的维护链上的数据。

公有链由于其公共开放的特性，所以任何人均可访问同时无法进行篡改等。但是由于其特性，所以吞吐量较低（TPS）,交易过程较为缓慢。例如比特币真实的吞吐量只有7,也就是每秒仅支持7笔交易。所以很难适用到目前的商业场景中。

1. **链式存储和 MPT 存储**
   1. **链式存储**
      1. **定义**

链式存储结构，又叫链接存储结构。在计算机中用一组任意的存储单元存储线性表的数据元素(这组存储单元可以是连续的,也可以是不连续的)。它不要求逻辑上相邻的元素在物理位置上也相邻.因此它没有顺序存储结构所具有的弱点,但也同时失去了顺序表可随机存取的优点。

* + 1. **特点**

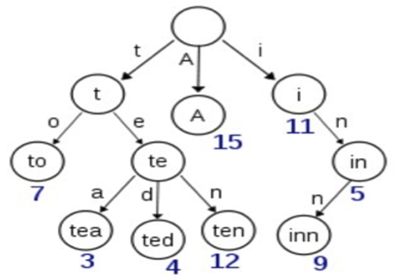
1. 比顺序存储结构的存储密度小(链式存储结构中每个结点都由数据域与指针域两部分组成，相比顺序存储结构增加了存储空间)。
2. 逻辑上相邻的节点物理上不必相邻。
3. 插入、删除灵活 (不必移动节点，只要改变节点中的指针)。
4. 查找节点时链式存储要比顺序存储慢。
5. 每个节点是由数据域和指针域组成。
6. 由于簇是随机分配的，这也使数据删除后覆盖几率降低，恢复可能提高。
   1. **MPT 存储**
      1. **定义**

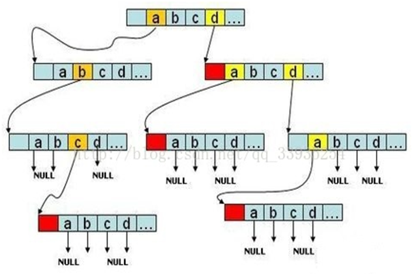
一种经过改良的、融合了默克尔树和前缀树两种树结构优点的数据结构，以太坊中，MPT是一个非常重要的数据结构，在以太坊中，帐户的交易信息、状态以及相应的状态变更，还有相关的交易信息等都使用MPT来进行管理，其是整个数据存储的重要一环。交易树，收据树，状态树都是采用的MPT结构。

* + 1. **作用**

1. 存储任意长度的key-value键值对数据；
2. 提供了一种快速计算所维护数据集哈希标识的机制；
3. 提供了快速状态回滚的机制；
4. 提供了一种称为默克尔证明的证明方法，进行轻节点的扩展，实现简单支付验证；
   * 1. **前缀树**

前缀树（又称字典树），用于保存关联数组，其键（key）的内容通常为字符串。前缀树节点在树中的位置是由其键的内容所决定的，即前缀树的key值被编码在根节点到该节点的路径中。

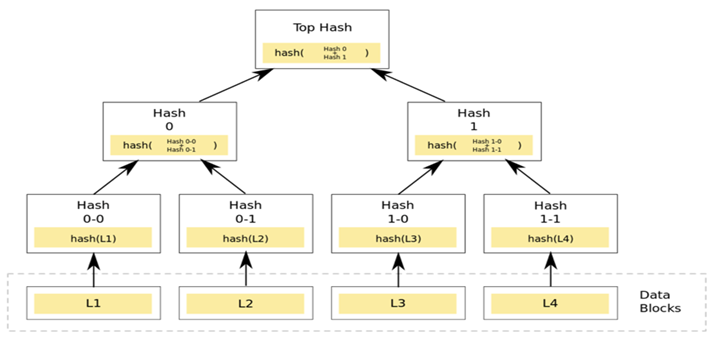




如下图所示，图中共有6个叶子节点，其key的值分别为（1）to（2）tea（3）ted（4）ten（5）A（6）inn。

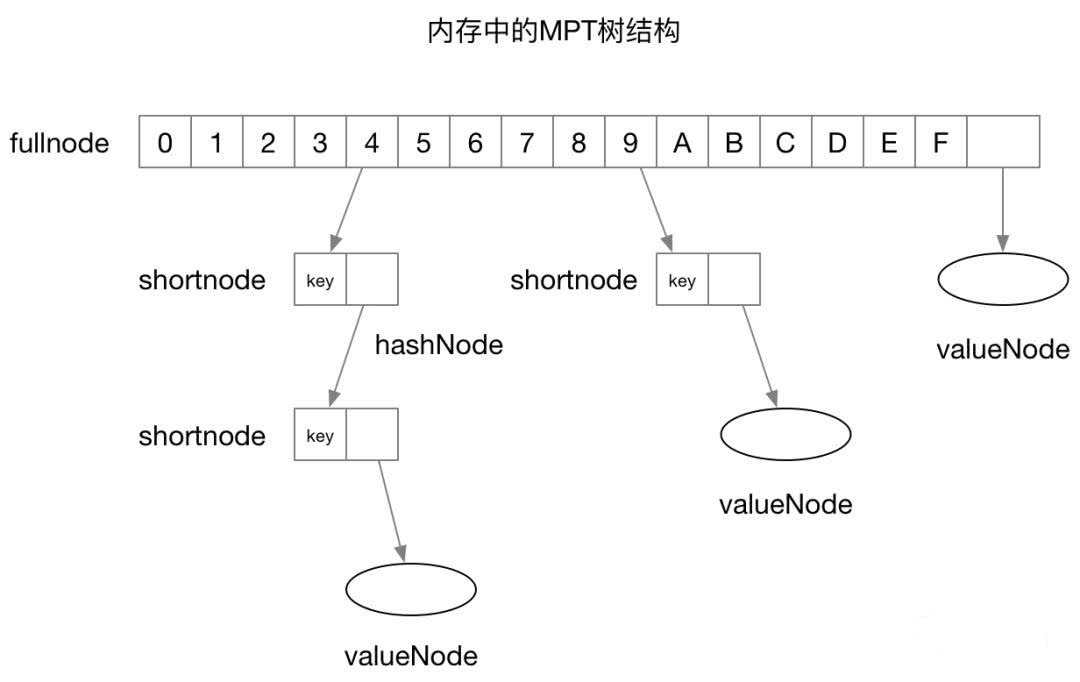
* + 1. **默克尔树**

merkle树是自底向上构建的。在下图的例子中，首先将L1-L4四个单元数据哈希化，然后将哈希值存储至相应的叶子节点。



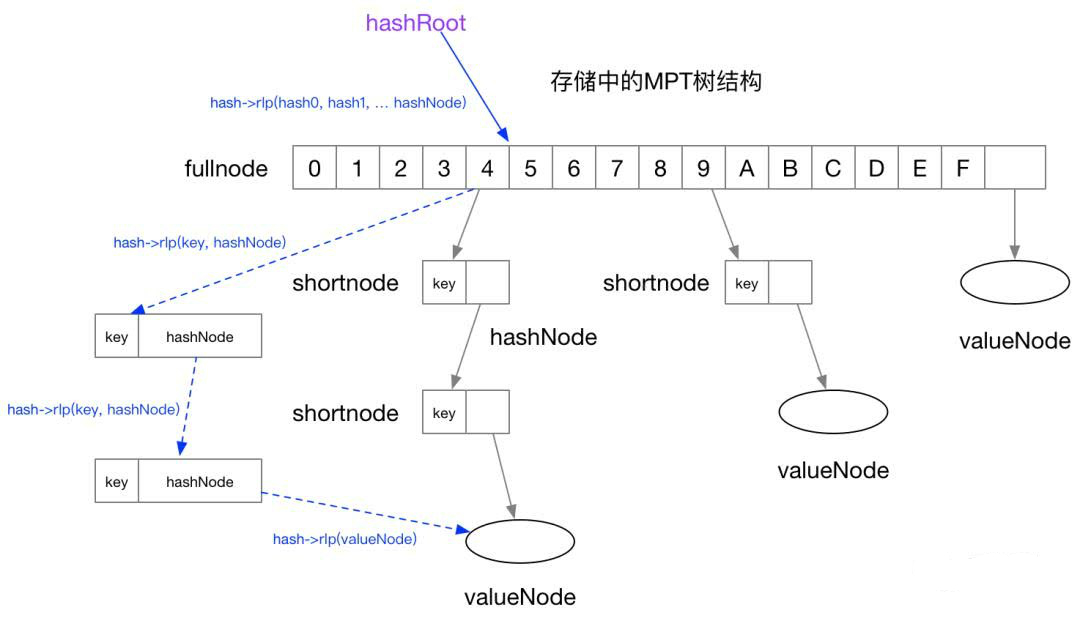
将相邻两个节点的哈希值合并成一个字符串，然后计算这个字符串的哈希，得到的就是这两个节点的父节点的哈希值。

* + 1. **MPT 树的内存结构**

MPT树的内存结构，如下图。一颗MPT树由4种节点连接组成：fullnode是分支节点，shortnode代表扩展节点或者叶子节点（取决于value是hashNode还是valueNode）。

需要指出的是：MPT的叶子节点（账户信息）是按照账户的地址（Address）的字典排序形成。也就是说，MPT树上的一个叶子节点路径上的所有的key组成的是账户地址。

* + 1. **MPT 树物理存储**

在内存中，节点和节点的连接关系可以通过“指针”完成。为了将连接关系持久化，需要将节点内容生成对应的，唯一的“地址”。其他引用节点内容的节点，只需要记录引用节点的地址即可，示意如下图中的蓝色部分。

以太坊中，节点内容的地址就是节点内容的hash。比如一个由“key”以及一个“hashNode”组成的节点，节点内容是rlp（key，hashNode），对应的地址是内容的hash。在存储中，存储的是hash到rlp的KV对。其他引用节点的“连接”用hash值代替，从而隐性的实现“连接”。

再举fullnode的例子，在存储中，fullnode的内容是rlp（hashNode，hashNode ... hashNode, valueNode)组成。

* + 1. **账户修改**

从上述可见，一个叶子节点的改变，导致节点内容的改变，节点的“地址”也会发生变化。也就是说，父亲节点和该节点的“连接”地址发生变化。因为父亲节点的内容包括“连接”地址，所以父亲节点的内容也发生变化，父亲节点的“地址”也随着变化。随即，父亲的父亲也发生变化，一直变化到MPT的树根。

总的来说，一个账户的改变，导致MPT树，从叶子到树根，整条路径上的所有节点的KV对都发生变化，需要更新到存储中。

以太坊中的节点间的“连接”关系用hash来代替，主要原因，hash结果唯一。还有个原因，有关账户中的Storage数据的MPT树。在Storage的数据构建的MPT树，节点排序是通过Storage数据内容的hash，这样保证MPT树的最大高度，避免攻击者通过构造Storage创建深度很高的MPT树。

MPT树中的分支节点的分支个数设置为16，而不是更大的原因，可能是怕分支节点内容变的太大，在节点内容大小和树高之间的一个平衡。

1. **Gas在智能合约中的作用**
   1. **“Gas”是以太坊使用的特殊单位的名称。**

它衡量一个动作或一系列动作需要执行多少“工作”：例如，计算一个Keccak256密码散列，每计算一次散列需要30个gas，每256位 数据被哈希。Ethereum平台上的一项交易或合同可以执行的每项操作都会花费一定数量的gas，其运营所需的计算资源比计算资源要求较少的运算需要更多的gas。

Gas的重要性在于它有助于确保提交给网络的交易支付适当的费用。通过要求交易支付每个操作的执行（或导致合同执行），我们确保网络不会因为执行大量对任何人无价值的密集工作而陷入困境。这与比特币交易费用不同，它仅基于交易的千字节大小。由于以太坊允许运行任意复杂的计算机代码，所以短的代码实际上可能导致大量计算工作的完成。所以衡量直接完成的工作非常重要，而不是仅仅根据交易或合同的长度选择费用。

所以，如果gas基本上是交易费用，那么你如何支付？这是一个棘手的地方。虽然gas是一个可以测量物质的单位，但gas并没有任何实际的标志。也就是说，你不能拥有1000 gas。相反，gas只存在于以太坊虚拟机内部，作为正在执行多少工作的计数。在实际支付gas时，交易费用是ether的一定数量，以太坊网络上的内置令牌和矿工奖励生产块的令牌。

这可能看起来很奇怪。为什么不直接用ether衡量成本？答案是，就像比特币一样，以太网的市场价格可能会迅速变化！但是计算的代价并不是因为以太的价格变化而上升或下降的。所以将计算价格与以太币的价格区分开来是很有用的，这样每次市场走势就不需要改变操作成本。

这里的术语有点混乱。EVM中的操作具有gas成本，但gas本身也具有以ether的gas价格。每笔交易都规定了每个gas单位愿意支付的gas价格，从而使市场能够决定gas价格和计算成本（以gas计量）之间的关系。这是两者的总和，即所用gas总量乘以gas price，得到交易支付的全部费用。

尽管这很棘手，但了解这个区别是很重要的，因为这会导致以太坊交易对最初的学习者来说最混乱的一件事情：您的交易没有用完，交易也没有足够高费用。如果我在我的交易中设定的gas price太低，那么没有人会在第一时间去管理我的交易。它不会被矿工包括在区块链中。但如果我提供一个可以接受的gas价格，那么我的交易就会产生如此多的计算工作，以至于合并后的gas成本超过了我所附加的费用数额，那么这个gas就会被计算为“花费”，我不会收回。矿工将停止处理交易，恢复所做的任何更改，但仍将其作为“失败的交易”包含在区块链中，收取费用。这看起来可能很苛刻，但是当你意识到矿工真正的工作是在执行计算的时候，你可以看到他们永远也不会获得这些资源。所以，即使你设计糟糕的交易用完了，你付给他们的工作也是公平的。

提供太多的费用也不同于提供太多的ether。如果你设置了一个非常高的gas price，那么你只需要付出很少的代价，就像在比特币中设置超高的交易费用一样。你肯定会被排在最前面，但你的钱已经没有了。但是，如果您提供了正常的gas price，并且只需要支付比您购买gas所需的更多的ether，那么超额部分将退还给您。矿工只收取你实际工作的费用。 你可以把gas价格看作矿工的小时工资，把gas成本看作是工作时间表。

gas还有许多其他的微妙之处，但这应该给你基本的东西！gas是使以太坊中的复杂计算“安全”的关键机制，因为任何失控的程序只会在请求运行的人提供的资金的情况下持续下去。当资金停止时，矿工们就停止工作。而你在程序中犯的错误只会影响付费使用它的人——网络的其他部分不会因为你的错误而遭受性能问题。当性能问题消耗掉所有的ether时，他们只会得到一个大的薪水！如果没有这个关键技术，通用区块链的想法将是完全不可能的。

总结:

* gas是费用的计算方式
* 不过，这些费用仍然以ether支付，不同于gas
* gas成本就是劳动时间之类的工作量，而gas price就像你为完成工作而支付的小时工资。 两者的组合决定了您的总交易费用。
* 如果您的**gas**价格太低，没有人会处理您的交易
* 如果你的**gas**价格没问题，但是你的交易的**gas**成本超过了预算，交易就会失败，但是仍然会进入区块链，你不会为劳动者的工作收回这笔钱。
* 这可以确保没有任何东西可以永远运行，人们会仔细考虑它们运行的代码。 它保持矿工和用户的安全不受恶劣的代码影响！
  1. **gas是在以太坊进行的每一项操作的执行费用。**

以太坊在区块链上实施了一个名为以太坊虚拟机（EVM）的执行环境。 当您运行分散式应用程序（dApp）时，每个指令都会在网络的每个节点上执行。这有一个代价：对于脚本可以执行的每个操作，都有一个指定的成本，用gas单位数表示，您可以在EVM规范中看到。

**gas**价格由矿主决定，目前约为5〜21 GWei（1 GWei为10 ^ 9 Wei或10 ^ -9 Ether）。 以太坊使用以太币作为其内部货币/标记。您的帐户持有以ether表示。当您部署合同或执行交易时，gas将从您的账户余额中提取。你可以自由指定一个gas price，或保留建议。

从理论上的PoV来看，每个采矿节点应该选择一个最大化其利润的**gas**价格。

* 1. **gas是以太坊“世界电脑”使用的计量单位。**

Gas 是执行智能合约操作的燃料，智能合约的每一个步骤都会消耗 Gas，Gas 是由以太坊的平台代币以太币转化而来，最小单位是 wei， 1ETH 相当于 10 的 18 次方 wei。

以太币可以通过 PoW 挖矿而产生，目前以太坊主要通过 GPU 挖矿。挖出一个块可以换得 5 个以太币，并且还有一定的交易费、以及叔伯块的奖励。

* 1. **gas和ether**

gas应该是网络资源/利用的不变成本。您希望发送交易的实际成本总是相同的，所以您不能真正期望gas发行，货币一般是不稳定的。

gas有多个相关的条款：Gas Prices，Gas Cost,Gas Limit 和Gas Fees。gas的原理是为以太坊网络的交易或计算成本有一个稳定的价值。

* Gas Cost是一个静态值，表示气体的计算成本是多少，目的是gas的实际价值永远不会改变，所以这个成本应该始终保持稳定。
* Gas Prices是多少gas成本在另一个货币或像Ether的象征。为了稳定gas的价值，gas价格是一个浮动价值，如果令牌或货币的成本波动，gas价格变化保持相同的实际价值。
* Gas Limit是指每个区块可以使用的最大的gas数量，它被认为是区块的最大计算负荷，交易量或区块大小，矿工可以随着时间慢慢地改变这个值。
* Gas Fees实际上是运行特定交易或计划（称为合同）所需支付的gas量。块的gas费用可以用来暗示块的计算负荷，交易量或大小。gas费用支付给矿工（或PoS保税承包商）。
  1. **gas基本上是执行交易或合同的内部定价**

每笔交易或合同的gas价格是为了处理以太坊及其EVM（以太坊虚拟机代码）。所以，交易或者操作越复杂，花费的gas就越多。

1. **EVM中的数据存储结构**
   1. **Solidity中的基本数据类型**
      1. **值类型**

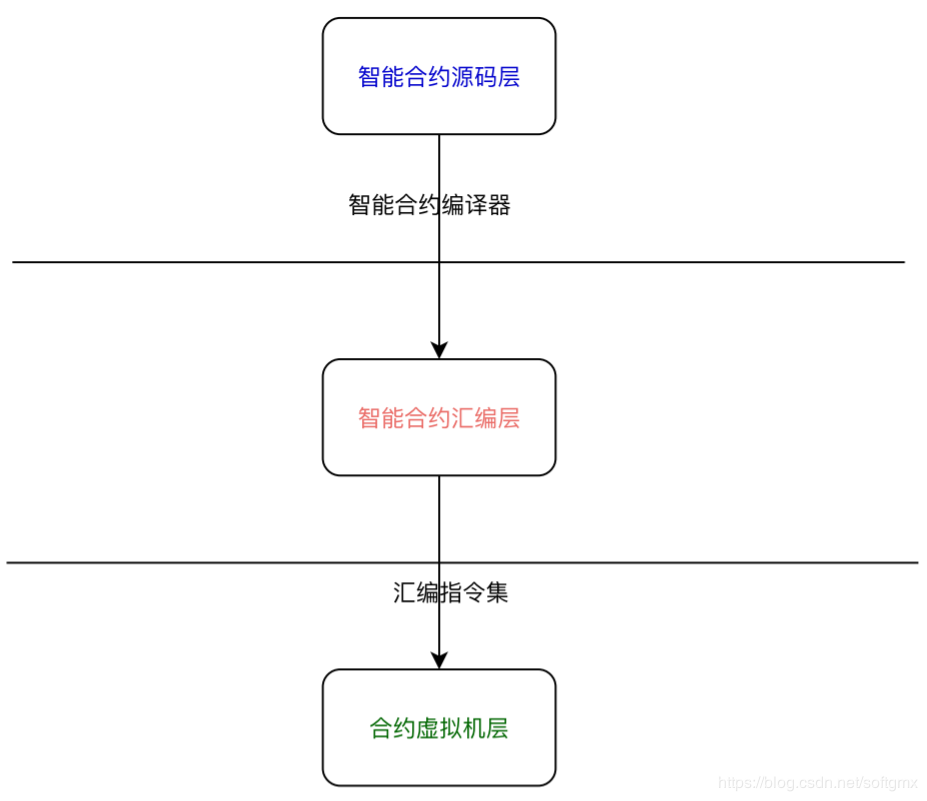
包括整型(int/uint)，地址型(address)，布尔(Booleans)，定长字节数组(fixed byte arrays)，枚举类型(Enums)...在内的基本数据类型，在合约中都是以值的类型进行传递(赋值)；

* + 1. **引用类型**

那些复杂类型，占用空间较大的。考虑到在拷贝到内存时成本比较大。所以考虑通过引用传递。比如，不定长字节数组（bytes），字符串（string），数组（Array），结构体（Struts）。

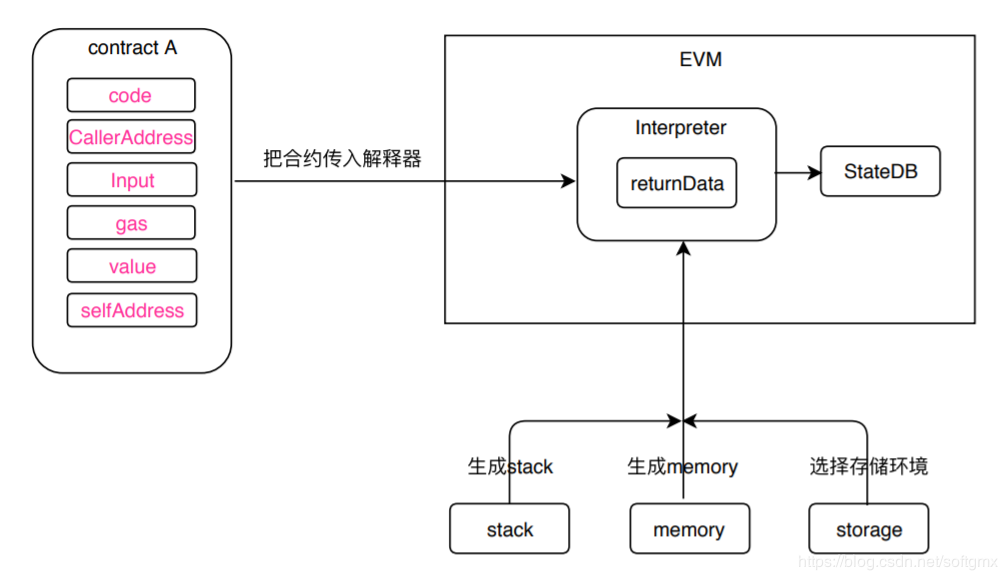
* + 1. **存储位置**

1. memory (内存，生命周期仅为整个方法执行期间，函数调用后回收，因为仅保存临时变量，故GAS开销很小)
2. storage (永久储存在区块链中，由于会永久保存合约状态变量，故GAS开销也最大)
3. 栈 (存放部分局部值类型变量，几乎免费使用的内存，但有数量限制)
   1. **EVM机制原理**
      1. **智能合约的层次**

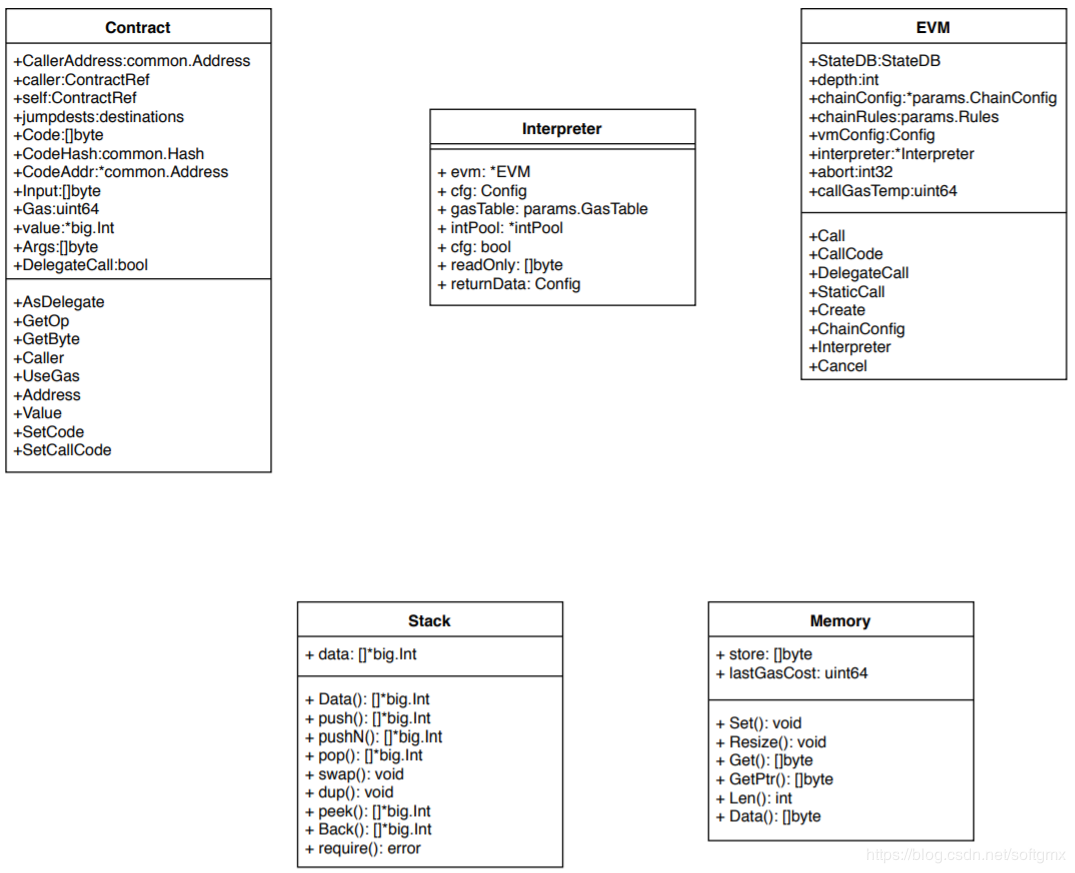


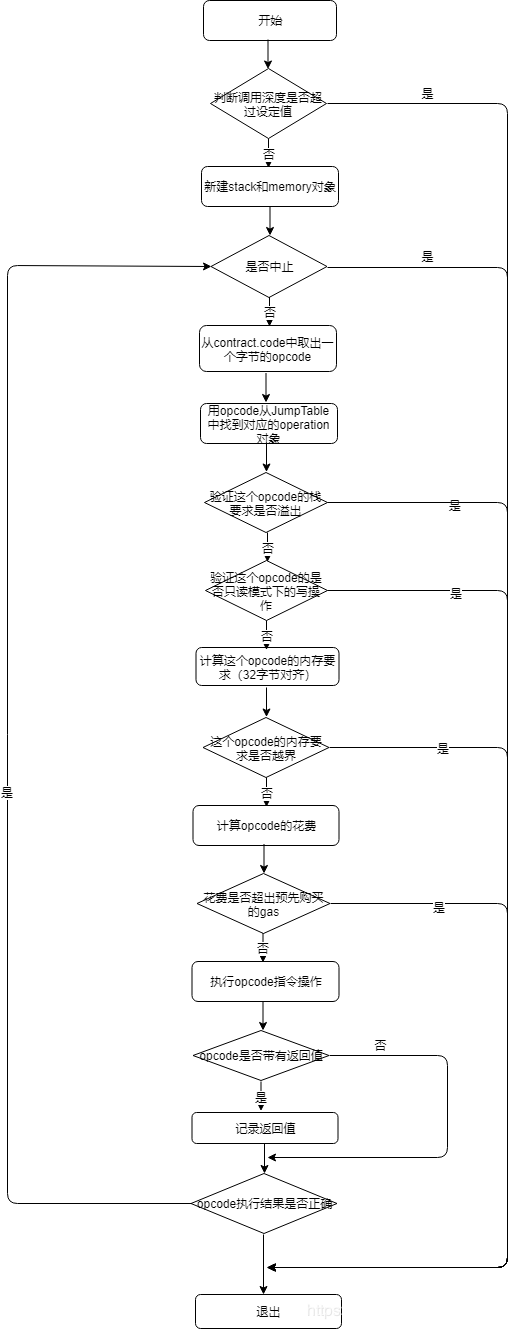
* + 1. **以太坊的智能合约机（EVM）构成及工作原理**

每次我们call一个合约方法时，在call的函数实现里，首先会创建一个contract类对象，并填充对应的字段值（code, CallerAddress, Input, gas, value, selfAddress），然后把这个contract对象传入EVM的解释器Interpeter进行逐条指令的解释执行，但在开始解释之前，会生成一个新的stack和一个memory对象以用于后面程序的运行，并把结果写入合约地址对应的StateDB。



* + 1. **EVM的几个关键类的定义**





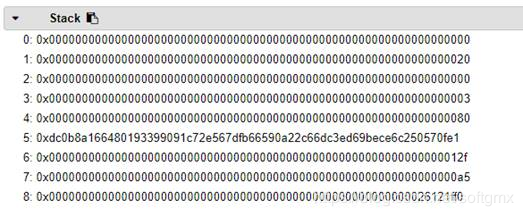
* 1. **EVM三大核心部件**
     1. **Stack的实现**

type Stack struct {

data []\*big.Int

}

1. 最小单元32字节（最小对齐单位）
2. 初始容量1024个元素
3. 以动态数组方式实现，理论上可扩容（但实际上一个方法只有1024个元素可用）
4. 主要指令push、pop、swap 、dup
5. 数据不持久化
6. 合约间调用，方法之间不共用栈（一个方法分配一个栈）
7. 同一合约的方法调用不产生call, 直是简单的 jump



* + 1. **Memory的实现**

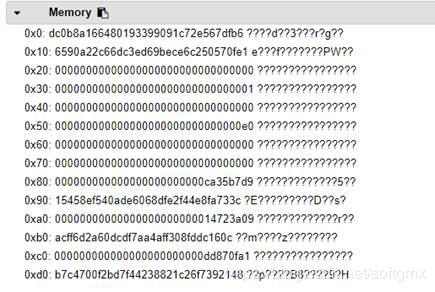
type Memory struct {

store []byte

lastGasCost uint64

}

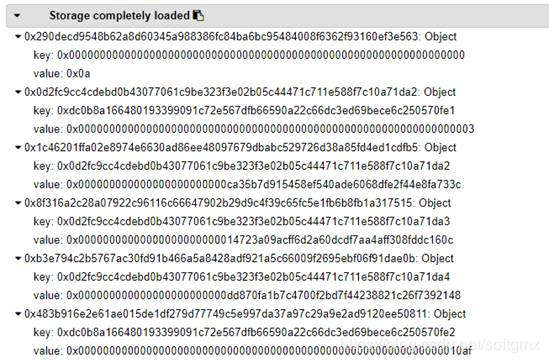
1. 最小单位有一个字节
2. 以动态字节数组方式实现，可以扩容
3. 类似x86架构里的堆，数据不具有持久性，合约执行完成，数据消失
4. 主要指令mload、mstore



* + 1. **Storage的实现**

1. 通过StateDB把数据存储在区块链上
2. 主要指令sload、sstore
3. 数据被持久化（写在链上了，并在所有矿机上同步）
4. 容量是2的256次幂，storage[slot]=value, key和value都是32位对齐的

slot=[ 0x0000000000000000000000000000000000000000000000000000000000000000,……,0xffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff]



1. **群组架构的好处**
   1. **群组的特点**
2. 群组架构允许网络存在多个账本，每个账本是一个独立的小组。

解决的问题：系统吞吐能力能够横向扩展

1. 各群组独立执行共识流程

解决的问题：解决了多群组同一个共识模块导致的性能瓶颈问题。按群组的粒度划分，由群组内参与者决定如何进行共识，一个群组内的共识不受其他群组影响，群组内维护自己的交易事务和数据，使得各群组之间解除耦合，独立运作，也便于进行横向扩展。

1. 一个组相当于一条链

解决的问题：实现了传统多链的扩展目的，同时一个节点可以参与到多条链，能够极大地简化运维复杂度，降低管理成本

* 1. **灵活扩展：保证业务接入和扩展像拉群聊天一样方便**

群的建立非常灵活，几个人就可以快速拉个主题群进行交流。同一个人可以参与到自己感兴趣的多个群里，并行地收发信息。现有的群也可以继续增加成员。

看回群组架构，采用群组架构的网络中，根据业务场景的不同，可存在多个不同的账本，区块链节点可以根据业务关系选择群组加入，参与到对应账本的数据共享和共识过程中。群组架构具有良好的扩展性，一个机构一旦参与到这样的联盟链里，有机会灵活快速地丰富业务场景和扩大业务规模，而系统的运维复杂度和管理成本也线性下降。

* 1. **多群组架构的网络消息隔离和数据隔离**

1. 网络消息隔离：

通过网络准入和账本白名单实现各账本间网络消息隔离。

1. 数据隔离：

每个群组独立运行各自的不同或者相同的共识算法。

每个账本模块自底向上主要包括核心层，接口层和调度层三层，三层相互协作，共同保证FISCO BCOS单个群组独立健壮地运行。

* 1. **隐私保护：各群组之间解除耦合独立运作**

回想一下群聊场景：群聊用户都在你的通信录中，都是经过验证才添加的，且不在群里的用户看不到群聊信息。这与联盟链准入机制不谋而合，所有参与者的机构身份可知。另一方面，群组架构中各群组独立执行共识流程，各组独立维护自己的交易事务和数据，不受其他群组影响。这样的好处是，可以使得各群组之间解除耦合独立运作，从而达成更好的隐私隔离。在跨群组之间的消息互通，则会带上验证信息，是可信和可追溯的。

1. **分布式存储有什么优势？** 
   1. **概念**

分布式存储系统，是将数据分散存储在多台独立的设备上。传统的网络存储系统采用集中的存储服务器存放所有数据，存储服务器成为系统性能的瓶颈，也是可靠性和安全性的焦点，不能满足大规模存储应用的需要。分布式网络存储系统采用可扩展的系统结构，利用多台存储服务器分担存储负荷，利用位置服务器定位存储信息，它不但提高了系统的可靠性、可用性和存取效率，还易于扩展。

* 1. **特性**

1. 可扩展。分布式存储系统可以扩展到几百台甚至几千台这样的一个集群规模，系统的整体性能线性增长。
2. 低成本。分布式存储系统的自动容错、自动负载均衡的特性，允许分布式存储系统可以构建在低成本的服务器上。另外，线性的扩展能力也使得增加、减少服务器的成本低，实现分布式存储系统的自动运维。
3. 高性能。无论是针对单台服务器，还是针对整个分布式的存储集群，都要求分布式存储系统具备高性能。
4. 易用。分布式存储系统需要对外提供方便易用的接口，另外，也需要具备完善的监控、运维工具，并且可以方便的与其他的系统进行集成。
   1. **要考虑的问题**
5. 数据分布：如何将数据均匀的分布到整个分布式存储集群中的各台服务器？如何从分布式存储集群中读取数据？
6. 一致性：如何将数据的多个副本复制到多台服务器，即使在异常情况下，也能保证不同副本之间的数据一致性。
7. 容错：如何可以快速检测到服务器故障，并自动的将在故障服务器上的数据进行迁移
8. 负载均衡：新增的服务器如何在集群中保障负载均衡？数据迁移过程中如何保障不影响现有的服务。
9. 事务与并发控制：如何实现分布式事务。
10. 易用性：如何设计对外接口，使得设计的系统易于使用
11. 压缩/加压缩：如何根据数据的特点设计合理的压缩/解压缩算法？如何平衡压缩/解压缩算法带来的空间和CPU计算资源？
    1. **分布式存储的六大优点**
       1. **高性能**

一个具有高性能的分布式存户通常能够高效地管理读缓存和写缓存，并且支持自动的分级存储。分布式存储通过将热点区域内数据映射到高速存储中，来提高系统响应速度;一旦这些区域不再是热点，那么存储系统会将它们移出高速存储。而写缓存技术则可使配合高速存储来明显改变整体存储的性能，按照一定的策略，先将数据写入高速存储，再在适当的时间进行同步落盘。

* + 1. **支持分级存储存储**

由于通过网络进行松耦合链接，分布式存储允许高速存储和低速存储分开部署，或者任意比例混布。在不可预测的业务环境或者敏捷应用情况下，分层存储的优势可以发挥到最佳。解决了目前缓存分层存储最大的问题是当性能池读不命中后，从冷池提取数据的粒度太大，导致延迟高，从而给造成整体的性能的抖动的问题。

版权声明：本文为博主原创文章，转载请附上博文链接！

* + 1. **多副本的一致性**

与传统的存储架构使用RAID模式来保证数据的可靠性不同，分布式存储采用了多副本备份机制。在存储数据之前，分布式存储对数据进行了分片，分片后的数据按照一定的规则保存在集群节点上。为了保证多个数据副本之间的一致性，分布式存储通常采用的是一个副本写入，多个副本读取的强一致性技术，使用镜像、条带、分布式校验等方式满足租户对于可靠性不同的需求。在读取数据失败的时候，系统可以通过从其他副本读取数据，重新写入该副本进行恢复，从而保证副本的总数固定;当数据长时间处于不一致状态时，系统会自动数据重建恢复，同时租户可设定数据恢复的带宽规则，最小化对业务的影响。

* + 1. **容灾与备份**

在分布式存储的容灾中，一个重要的手段就是多时间点快照技术，使得用户生产系统能够实现一定时间间隔下的各版本数据的保存。特别值得一提的是，多时间点快照技术支持同时提取多个时间点样本同时恢复，这对于很多逻辑错误的灾难定位十分有用，如果用户有多台服务器或虚拟机可以用作系统恢复，通过比照和分析，可以快速找到哪个时间点才是需要回复的时间点，降低了故障定位的难度，缩短了定位时间。这个功能还非常有利于进行故障重现，从而进行分析和研究，避免灾难在未来再次发生。多副本技术，数据条带化放置，多时间点快照和周期增量复制等技术为分布式存储的高可靠性提供了保障。

* + 1. **弹性扩展**

得益于合理的分布式架构，分布式存储可预估并且弹性扩展计算、存储容量和性能。分布式存储的水平扩展有以下几个特性：

1. 节点扩展后，旧数据会自动迁移到新节点，实现负载均衡，避免单点过热的情况出现;
2. 水平扩展只需要将新节点和原有集群连接到同一网络，整个过程不会对业务造成影响;
3. 当节点被添加到集群，集群系统的整体容量和性能也随之线性扩展，此后新节点的资源就会被管理平台接管，被用于分配或者回收。
   * 1. **存储系统标准化**

随着分布式存储的发展，存储行业的标准化进程也不断推进，分布式存储优先采用行业标准接口(SMI-S或OpenStack Cinder)进行存储接入。在平台层面，通过将异构存储资源进行抽象化，将传统的存储设备级的操作封装成面向存储资源的操作，从而简化异构存储基础架构的操作，以实现存储资源的集中管理，并能够自动执行创建、变更、回收等整个存储生命周期流程。基于异构存储整合的功能，用户可以实现跨不同品牌、介质地实现容灾，如用中低端阵列为高端阵列容灾，用不同磁盘阵列为闪存阵列容灾等等，从侧面降低了存储采购和管理成本。

1. **并行计算**
   1. **定义**

并行计算（Parallel Computing）是指同时使用多种计算资源解决计算问题的过程，是提高计算机系统计算速度和处理能力的一种有效手段。它的基本思想是用多个处理器来协同求解同一问题，即将被求解的问题分解成若干个部分，各部分均由一个独立的处理机来并行计算。并行计算系统既可以是专门设计的、含有多个处理器的超级计算机，也可以是以某种方式互连的若干台的独立计算机构成的集群。通过并行计算集群完成数据的处理，再将处理的结果返回给用户。

并行计算可分为时间上的并行和空间上的并行。

* 1. **特征**

为利用并行计算，通常计算问题表现为以下特征：

1. 将工作分离成离散部分，有助于同时解决；
2. 随时并及时地执行多个程序指令；
3. 多计算资源下解决问题的耗时要少于单个计算资源下的耗时。
   1. **基本体系结构**

并行计算科学中主要研究的是空间上的并行问题。从程序和算法设计人员的角度来看，并行计算又可分为数据并行和任务并行。一般来说，因为数据并行主要是将一个大任务化解成相同的各个子任务，比任务并行要容易处理。

空间上的并行导致了两类并行机的产生，按照Flynn的说法分为：单指令流多数据流（SIMD）和多指令流多数据流（MIMD）。我们常用的串行机也叫做单指令流单数据流（SISD）。MIMD类的机器又可分为以下常见的五类：并行向量处理机（PVP）、对称多处理机（SMP）、大规模并行处理机（MPP）、工作站机群（COW）、分布式共享存储处理机（DSM）。

* + 1. **访存模型**

并行计算机有以下五种访存模型：

1. 均匀访存模型（UMA）
2. 非均匀访存模型（NUMA）
3. 全高速缓存访存模型（COMA）
4. 一致性高速缓存非均匀存储访问模型（CC-NUMA）
5. 非远程存储访问模型（NORMA）。
   * 1. **计算模型**

不像串行计算机那样，全世界基本上都在使用冯·诺伊曼的计算模型；并行计算机没有一个统一的计算模型。不过，人们已经提出了几种有价值的参考模型：PRAM模型，BSP模型，LogP模型，C^3模型等。

* 1. **基本术语**

1. 节点度:射入或射出一个节点的边数。在单向网络中，入射和出射边之和称为节点度。
2. 网络直径:网络中任何两个节点之间的最长距离，即最大路径数。
3. 对剖宽度:对分网络各半所必须移去的最少边数。
4. 对剖带宽:每秒钟内，在最小的对剖平面上通过所有连线的最大信息位（或字节)。
   1. **性能度量**
      1. **基本指标**

执行时间

工作负载

存储性能

* + 1. **加速比评测**

Amdahl定理

Gastofson定理

Sun-Ni定理

* + 1. **可扩放性标准**

等效率标准

等速度标准

平均延迟标准

并行计算与云计算

云计算是在并行计算之后产生的概念，是由并行计算发展而来， 两者在很多方面有着共性。学习并行计算对于理解云计算有很大的帮助。并行计算是学习云计算必须要学习的基础课程。

但并行计算不等于云计算，云计算也不等同并行计算。两者区别如下。

1. 云计算萌芽于并行计算

云计算的萌芽应该从计算机的并行化开始，并行机的出现是人们不满足于CPU摩尔定率的增长速度，希望把多个计算机并联起来，从而获得更快的计算速度。这是一种很简单也很朴素的实现高速计算的方法，这种方法后来被证明是相当成功的。

1. 并行计算、网格计算只用于特定的科学领域，专业的用户

并行计算、网格计算的提出主要是为了满足科学和技术领域的专业需要，其应用领域也基本限于科学领域。传统并行计算机的使用是一个相当专业的工作，需要使用者有较高的专业素质，多数是命令行的操作，这是很多专业人士的噩梦，更不用说普通的业余级用户了。

1. 并行计算追求的高性能

在并行计算的时代，人们极力追求的是高速的计算、采用昂贵的服务器，各国不惜代价在计算速度上超越他国，因此，并行计算时代的高性能机群是一个“快速消费品”，世界TOP500高性能计算机地排名不断地在刷新，一台大型机群如果在3年左右不能得到有效的利用就远远的落后了，巨额投资无法收回。

1. 云计算对于单节点的计算能力要求低

而云计算时代我们并不去追求使用昂贵的服务器，我们也不用去考虑TOP500的排名，云中心的计算力和存储力可随着需要逐步增加，云计算的基础架构支持这一动态增加的方式，高性能计算将在云计算时代成为“耐用消费品”。

**参考文献**

[1] 郭彬, 于飞, 陈劲. 区块链技术与信任世界的构建[J]. 企业管理, 2016(11):110-113.

[2] [区块链-以太坊MPT存储](https://cloud.tencent.com/developer/news/305864)

[3] [解密EVM实现机制](https://blog.csdn.net/softgmx/article/details/84336914)

[4] [分布式存储的概念及特性](https://blog.csdn.net/master_ning/article/details/80880164)

[5] [分布式存储的六大优点](https://blog.csdn.net/weixin_33901843/article/details/87521625)

[6] [并行计算](https://baike.baidu.com/item/%E5%B9%B6%E8%A1%8C%E8%AE%A1%E7%AE%97)

[7] [【智能合约】以太坊中智能合约调用中用的gas相关概念详解](https://blog.csdn.net/diandianxiyu_geek/article/details/78475639)