**课程报告**

|  |  |
| --- | --- |
| **课程名称：** | **企业软件项目实训** |
| **学生姓名：** | **冯韵** |
| **学生学号：** | **201636664318** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2018-2019第二学期** |

**软件学院**

**2019年7月**

# 1. 区块链技术原理

区块链（Blockchain）是比特币的一个重要概念，货币联合清华大学五道口金融学院互联网金融实验室、新浪科技发布的《2014—2016全球比特币发展研究报告》提到区块链是比特币的底层技术和基础架构。本质上是一个去中心化的数据库，同时作为比特币的底层技术。区块链是一串使用密码学方法相关联产生的数据块，每一个数据块中包含了一次比特币网络交易的信息，用于验证其信息的有效性（防伪）和生成下一个区块。

# 1.1 技术逻辑

区块链可以简单的分为三个层次，协议层、扩展层和应用层。其中，协议层又可以分为存储层和网络层，它们相互独立但又不可分割。

# 协议层

所谓的协议层，就是指代最底层的技术。这个层次通常是一个完整的区块链产品，类似于我们电脑的操作系统，它维护着网络节点，仅提供API供调用。通常官方会提供简单的客户端（通称为钱包），这个客户端钱包功能也很简单，只能建立地址、验证签名、转账支付、查看余额等。这个层次是一切的基础，构建了网络环境、搭建了交易通道、制定了节点奖励规则，至于你要交易什么，想干什么，它一概不过问，也过问不了。典型的例子，自然是比特币，还有各种二代币，比如莱特币等，本书介绍的亿书币也是。这个层次，是现阶段开发者聚集的地方，这说明加密货币仍在起步当中。

从用到的技术来说，协议层主要包括网络编程、分布式算法、加密签名、数据存储技术等4个方面，其中网络编程能力是大家选择编程语言的主要考虑因素，因为分布式算法基本上属于业务逻辑上的实现，什么语言都可以做到，加密签名技术是直接简单的使用（请看书中相关的加密解密文章，不建议自由发挥，没有过多的编码逻辑），数据库技术也主要在使用层面，只有点对点网络的实现和并发处理才是开发的难点，所以对于那些网络编程能力强，对并发处理简单的语言，人们就特别偏爱。也因此，Nodejs开发区块链应用，逐渐变得更加流行，Go语言也在逐渐兴起。

我把这个层面进一步分成了存储层和网络层。数据存储可以相对独立，选择自由度大一些，可以单独来讨论。选择的原则无非是性能和易用性。我们知道，系统的整体性能，主要取决于网络或数据存储的I/O性能，网络I/O优化空间不大，但是本地数据存储的I/O是可以优化的。比如，比特币选择的是谷歌的LevelDB，据说这个数据库读写性能很好，但是很多功能需要开发者自己实现。目前，困扰业界的一个重大问题是，加密货币交易处理量远不如现在中心化的支付系统（银行等），除了I/O，需要全方位的突破。

分布式算法、加密签名等都要在实现点对点网络的过程中加以使用，所以自然是网络层的事情，也是编码的重点和难点，《Nodejs开发加密货币》全书分享的基本上就是这部分的内容。当然，也有把点对点网络的实现单独分开的，把节点查找、数据传输和验证等逻辑独立出来，而把共识算法、加密签名、数据存储等操作放在一起组成核心层。无论怎么组合，这两个部分都是最核心、最底层的部分，都是协议层的内容。

# 1.1.2 扩展层

这个层面类似于电脑的驱动程序，是为了让区块链产品更加实用。目前有两类，一是各类交易市场，是法币兑换加密货币的重要渠道，实现简单，来钱快，成本低，但风险也大。二是针对某个方向的扩展实现，比如基于亿书侧链，可为第三方出版机构、论坛网站等内容生产商提供定制服务等。特别值得一提的就是大家听得最多的“智能合约”的概念，这是典型的扩展层面的应用开发。所谓“智能合约”就是“可编程合约”，或者叫做“合约智能化”，其中的“智能”是执行上的智能，也就是说达到某个条件，合约自动执行，比如自动转移证券、自动付款等，目前还没有比较成型的产品，但不可否认，这将是区块链技术重要的发展方向。

扩展层使用的技术就没有什么限制了，可以包括很多，上面提到的分布式存储、机器学习、VR、物联网、大数据等等，都可以使用。编程语言的选择上，可以更加自由，因为可以与协议层完全分离，编程语言也可以与协议层使用的开发语言不相同。在开发上，除了在交易时与协议层进行交互之外，其他时候尽量不要与协议层的开发混在一起。这个层面与应用层更加接近，也可以理解为B/S架构的产品中的服务端（Server）。这样不仅在架构设计上更加科学，让区块链数据更小，网络更独立，同时也可以保证扩展层开发不受约束。

从这个层面来看，区块链可以架构开发任何类型的产品，不仅仅是用在金融行业。在未来，随着底层协议的更加完善，任何需要第三方支付的产品都可以方便的使用区块链技术；任何需要确权、征信和追溯的信息，都可以借助区块链来实现。我个人觉得，这个目标应该很快就能实现。

# 1.1.3 应用层

这个层面类似于电脑中的各种软件程序，是普通人可以真正直接使用的产品，也可以理解为B/S架构的产品中的浏览器端（Browser）。这个层面的应用，目前几乎是空白。市场亟待出现这样的应用，引爆市场，形成真正的扩张之势，让区块链技术快速走进寻常百姓，服务于大众。大家使用的各类轻钱包（客户端），应该算作应用层最简单、最典型的应用。很快，亿书将基于亿书网络推出文档协作工具，这个就是典型的应用层的产品。

限于当前区块链技术的发展，亿书只能从协议层出发，把目标指向应用层，同时为第三方开发者提供扩展层的强大支持。这样做既可以避免贪多，又可以避免无法落地，是真正理性的开发路线。因为纯粹的开发协议层或扩展层，无法真正理解和验证应用层，会脱离实际，让第三方开发者很难使用。如果仅仅考虑应用层，市面上又找不到真正牢固、易用的协议层或扩展层的产品。所以，我们只好全面发力，采取完全开源开放的态度，通过社区的力量，共同去做一件有意义的事情，也算为中国区块链技术发展做点技术积累和微薄贡献。

# 1.2 技术原理

如今的区块链技术概括起来是指通过去中心化和去信任的方式集体维护一个可靠数据库的技术。其实，区块链技术并不是一种单一的、全新的技术，而是多种现有技术（如加密算法、P2P文件传输等）整合的结果，这些技术与数据库巧妙地组合在一起，形成了一种新的数据记录、传递、存储与呈现的方式。简单的说，区块链技术就是一种大家共同参与记录信息、存储信息的技术。过去，人们将数据记录、存储的工作交给中心化的机构来完成，而区块链技术则让系统中的每一个人都可以参与数据的记录、存储。区块链技术在没有中央控制点的分布式对等网络下，使用分布式集体运作的方法，构建了一个P2P的自组织网络。通过复杂的校验机制，区块链数据库能够保持完整性、连续性和一致性，即使部分参与人作假也无法改变区块链的完整性，更无法篡改区块链中的数据。区块链技术涉及的关键点包括：、去中心化、集体维护、时间戳、可靠数据库、去信任、非对称加密等。

区块链技术重新定义了网络中信用的生成方式：在系统中，参与者无需了解其他人的背景资料，也不需要借助第三方机构的担保或保证，区块链技术保障了系统对价值转移的活动进行记录、传输、存储，其最后的结果一定是可信的。

区块链技术原理的来源可归纳为一个数学问题：拜占庭将军问题。拜占庭将军问题延伸到互联网生活中来，其内涵可概括为：在互联网大背景下，当需要与不熟悉的对手方进行价值交换活动时，人们如何才能防止不会被其中的恶意破坏者欺骗、迷惑从而做出错误的决策。进一步将拜占庭将军问题延伸到技术领域中来，其内涵可概括为：在缺少可信任的中央节点和可信任的通道的情况下，分布在网络中的各个节点应如何达成共识。区块链技术解决了闻名已久的拜占庭将军问题——它提供了一种无需信任单个节点、还能创建共识网络的方法。[1]

# 2. 联盟链和公有链的异同

# 2.1 公有链

公有链通常也称为非许可链（Permissionless Blockchain），任何人都可以参与区块链数据维护和读取，容易部署应用程序，完全去中心化不受任何机构控制。

公有链的应用非常广泛，例如资产证券化、数字资产的跨链流通 ……现在市场上的主流大势区块链项目比特币、以太坊、量子链、EOS、唯链以及Neo等都是公有链项目。

公有链是真正意义上的完全去中心化的区块链，它通过密码学保证交易不可篡改，同时也利用密码学验证以及经济上的激励，在互为陌生的网络环境中建立共识，从而形成去中心化的信用机制。在公有链中的共识机制一般是工作量证明（PoW）和权益证明（PoS）。

公有链具有通过去中介化的方式打破当前中心化商业模式的潜力，而且本身无需维护服务器或管理系统，从根本上降低创建和运行去中心化应用程序（dApp）的成本。

# 2.2 联盟链

联盟链是一种需要注册许可的区块链，这种区块链也称为许可链（Permissioned Blockchain）。联盟链仅限于联盟成员参与，联盟规模可以大到国与国之间，也可以是不同的机构企业之间。

区块链上的读写权限、参与记账权限按联盟规则来制定。整个网络由成员机构共同维护，网络接入一般通过成员机构的网关节点接入，共识过程由预先选好的节点控制。因此联盟链一般不采用工作量证明的挖矿机制，而是多采用权益证明（PoS）或PBFT（Practical Byzantine Fault Tolerant）、RAFT等共识算法。

和公有链最高每秒完成交易3-20相比，联盟链可以达到1000-10000 ，交易速度更快且交易成本大幅降低。

联盟链可以解决结算问题，降低两地结算的成本和时间，适合于机构间的交易、结算等B2B场景，因此金融行业应用最广泛。其中最知名的就是R3CVE组织，即R3联盟，有包括花旗银行、中国平安银行、纽约梅隆银行在内的50多家银行机构加入。

# 2.3 私有链

私有链，仅限于企业、国家机构或者单独个体使用，不完全能够解决信任问题，但是可以改善可审计性。常用于企业内部的数据库管理、审计等，政府的预算和执行，或者政府的行业统计数据等。他们彼此之间需要透明，但没必要对外公众透明。

私有链的价值主要是提供安全、可追溯、不可篡改、自动执行的运算平台，可以同时防范来自内部和外部对数据的安全攻击，这个在传统的系统是很难做到的。

任何人都可以创建私链的平台的Multichain项目本身就是一个私有链项目。

# 2.4 对比



公有链，完全去中心化特质在各行各业都不愁用武之地，现如今已成大势；联盟链，超级俱乐部，因为区块链透明性，再也不用担心谁会在背后开黑枪；私有链，私人化订制，更透明安全。

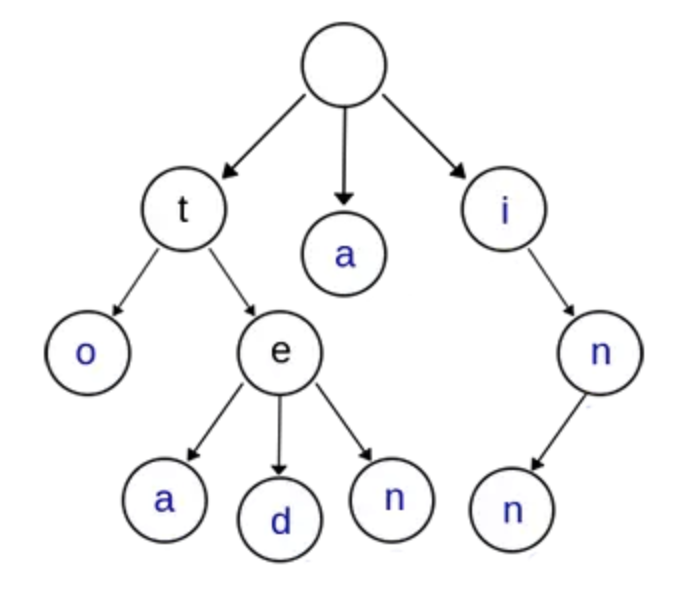
尽管分类不同，应用领域不同，但是却都从区块链技术中受益匪浅，也正是因为区块链的包罗万象，让区块链保持活力的同时也给予了它不断更新和发展的动力。[2]

# 3. 链式存储和MPT存储

MPT(Merkle Patricia Tries)是以太坊存储数据的核心数据结构，它是由Merkle Tree和Patricia Tree结合的一种树形结构，理解MPT有助于我们更好的理解以太坊的数据存储。在了解MPT数据结构之前，我们需要先来看看基本的Tree结构和Merkle Tree、Patricia Tree。

# 3.1 Trie字典树

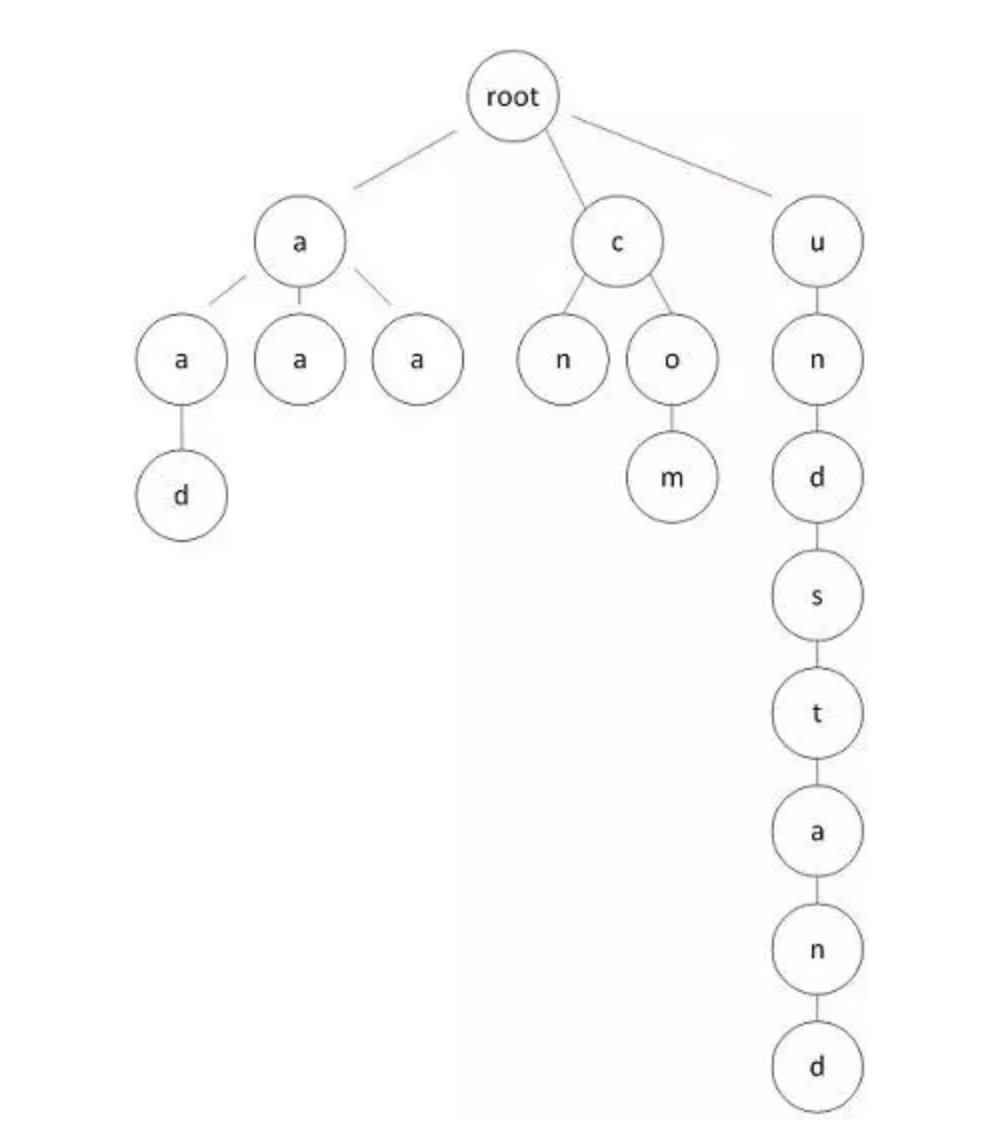
  Trie树，又称前缀树或字典树，是一种有序树，用于保存关联数组，其中的键通常是字符串。一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。



上图是一棵Trie树，表示了字符串集合{“a”, “to”, “tea”, “ted”, “ten”, “i”, “in”, “inn”} ，从上图中我们可以看出Trie树的特点：

* 根节点不包含字符，除根节点外的每一个子节点都包含一个字符。
* 从根节点到某一个节点，路径上经过的字符连接起来，为该节点对应的字符串。
* 每个节点的所有子节点包含的字符互不相同。

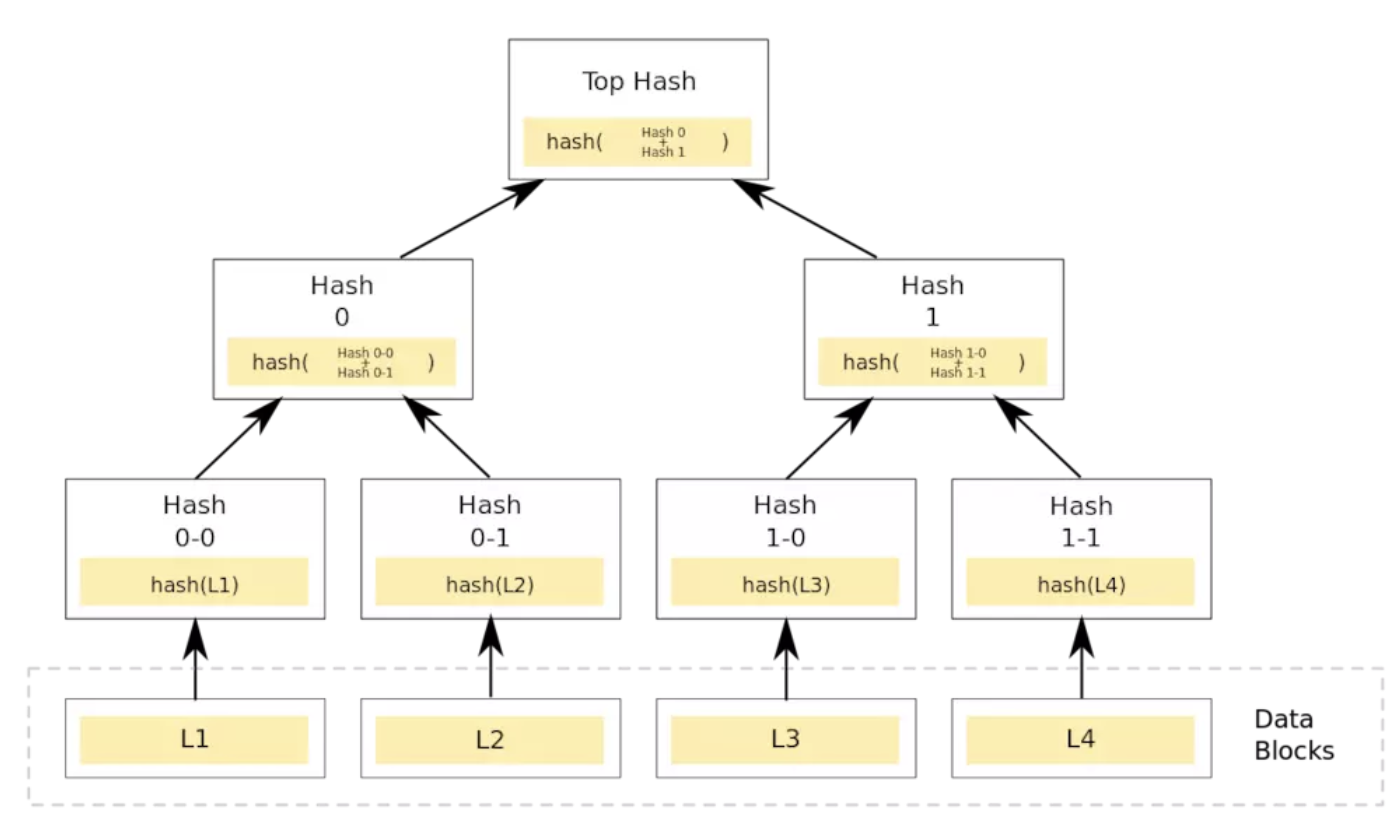
但是从上面的结构也可以看出一个问题：高度不可控，如下图所示。所以就有了Patricia树（压缩前缀树），后面会介绍到



# 3.2 Merkle Tree

Merkle Tree，也被称为Hash Tree，中文名称：默克尔树，主要用于数据集较大时的文件校验。其主要特点为：

* 叶节点存储着数据块的Hash（如：文件块、一段数据集）
* 非叶子节点（包括中间节点和根节点）存储着对应子节点Hash值串联字符串之后的Hash值。



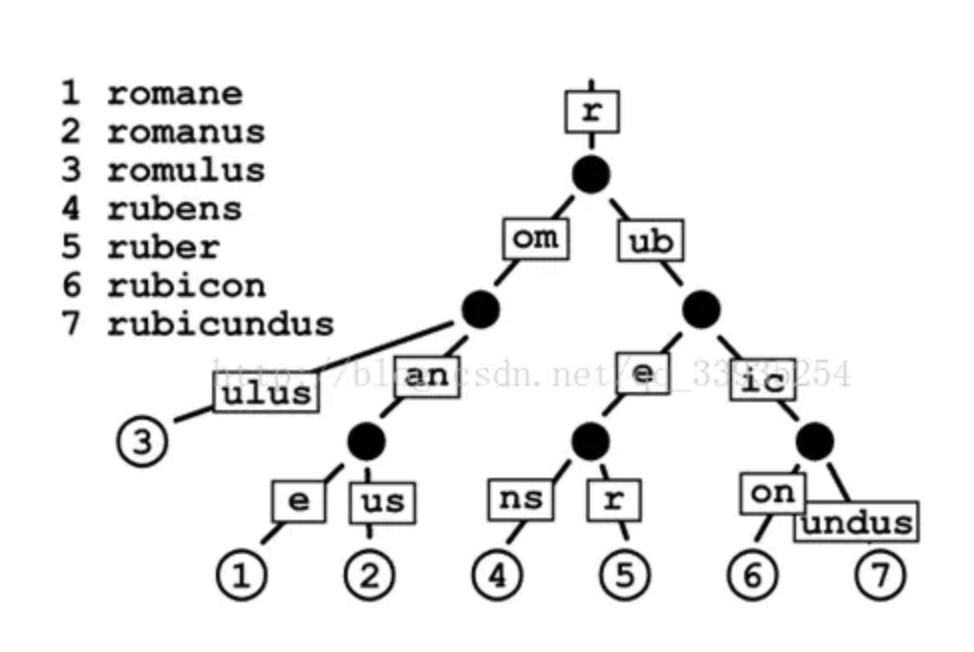
解释：  
1. 在最底层，和哈希列表一样，我们把数据分成小的数据块，有相应地哈希和它对应；

2. 往上走，并不是直接去运算根哈希，而是把相邻的两个哈希合并成一个字符串，然后运算这个字符串的哈希，这样每两个哈希就结婚生子，得到了一个”子哈希“。如果最底层的哈希总数是单数，那到最后必然出现一个单身哈希，这种情况就直接对它进行哈希运算，所以也能得到它的子哈希再往上推，依然是一样的方式，可以得到数目更少的新一级哈希，  
3. 最终必然形成一棵倒挂的树，到了树根的这个位置，这一代就剩下一个根哈希了，我们把它叫做 Merkle Root。

对于这种数据结构，在实际应用中会有哪些应用场景了，举个例子，我们知道现在从网上下载文件，很多都是P2P下载，文件会切分成很多小的数据块，每个数据块从不同的来源上下载，这些机器可以认为是不稳定或不可信的，文件下载完之后我们需要校验文件的完整性，这时我们总不能把文件再次切分然后分别计算它的Hash和下载前的Hash做对比吧，这时就需要用到Merkle Tree。  
  在下载前，先从可靠的源获得文件的Merkle Tree树根，下载后，在合并文件之前先对比小文件的Hash是否一样，如果一样就认为是可靠的，如果不一样，就判定文件被损坏，从新的来源重新下载，文件合并之后，计算小数据块的Hash并最终计算根Hash，也成为Merkle Root，然后对比根Hash是否一致。这样就避免了对整个文件进行Hash计算，因为当文件太大时，这种计算是很耗时。

# 3.3 Patricia Tree

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



# 3.4 MPT（Merkle Patricia Tree）

上面我们介绍了Merkle Tree和Patricia Tree，而MPT（Merkle Patricia Tree），顾名思义就是这两者的结合。MTP树种的节点包含空节点、叶子节点、扩展节点和分支节点。

**Nibble：**它是key的基本单元，是一个四元组（四个bit位的组合例如二进制表达的0010就是一个四元组）  
**\*\*空节点\*\*\*\*：**简单的表示空，在代码中是一个空串。  
**叶子节点（leaf）：**只有两个元素，分别为key和value，表示为[key,value]的一个键值对，其中key是key的一种特殊十六进制编码，value是value的RLP编码。  
**扩展节点（extension）：**也是[key，value]的一个键值对，但是这里的value是其他节点的hash值，这个hash可以被用来查询数据库中的节点。也就是说通过hash链接到其他节点。  
**分支节点（branch）：**分支节点有17个元素，回到Nibble，四元组是key的基本单元，四元组最多有16个值。所以前16个必将落入到在其遍历中的键的十六个可能的半字节值中的每一个。第17个是存储那些在当前结点结束了的节点(例如， 有三个key,分别是 (abc ,abd, ab) 第17个字段储存了ab节点的值)

这里还有一些知识点需要了解的，为了将MPT树存储到数据库中，同时还可以把MPT树从数据库中恢复出来，对于Extension和Leaf的节点类型做了特殊的定义：如果是一个扩展节点，那么前缀为0，这个0加在key前面。如果是一个叶子节点，那么前缀就是1。同时对key的长度就奇偶类型也做了设定，如果是奇数长度则标示1，如果是偶数长度则标示0。

以太坊的每一个区块头，并非只包含一棵MPT树，而是包含了三棵MPT树，分别对应了四种对象：

State Trie 区块头中的状态树

* key => sha3(以太坊账户地址address)
* value => rlp(账号内容信息account)

Transactions Trie 区块头中的交易树

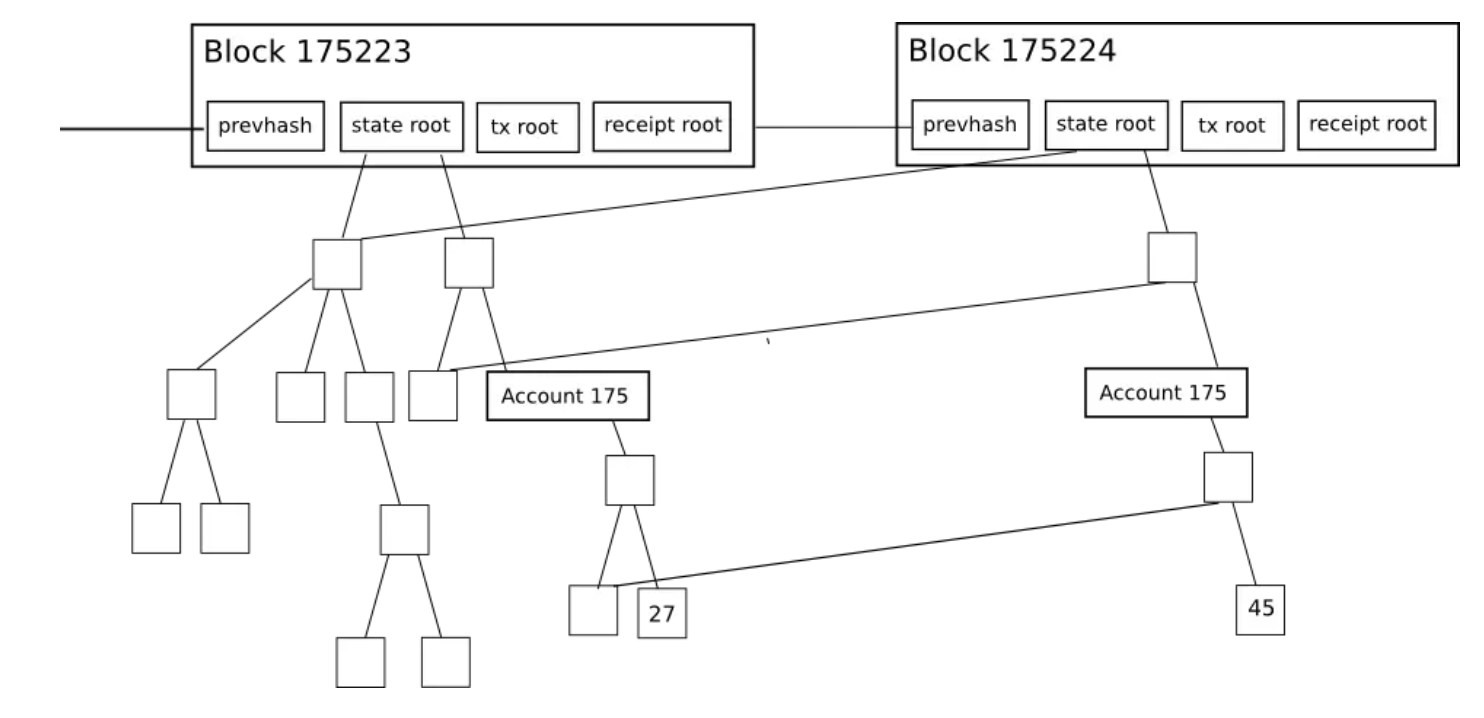
* key => rlp(交易的偏移量 transaction index)
* 每个块都有各自的交易树，且不可更改

Receipts Trie 区块头中的收据树

* key = rlp(交易的偏移量 transaction index)
* 每个块都有各自的交易树，且不可更改

Storage Trie 存储树

* 存储只能合约状态
* 每个账号有自己的Storage Trie



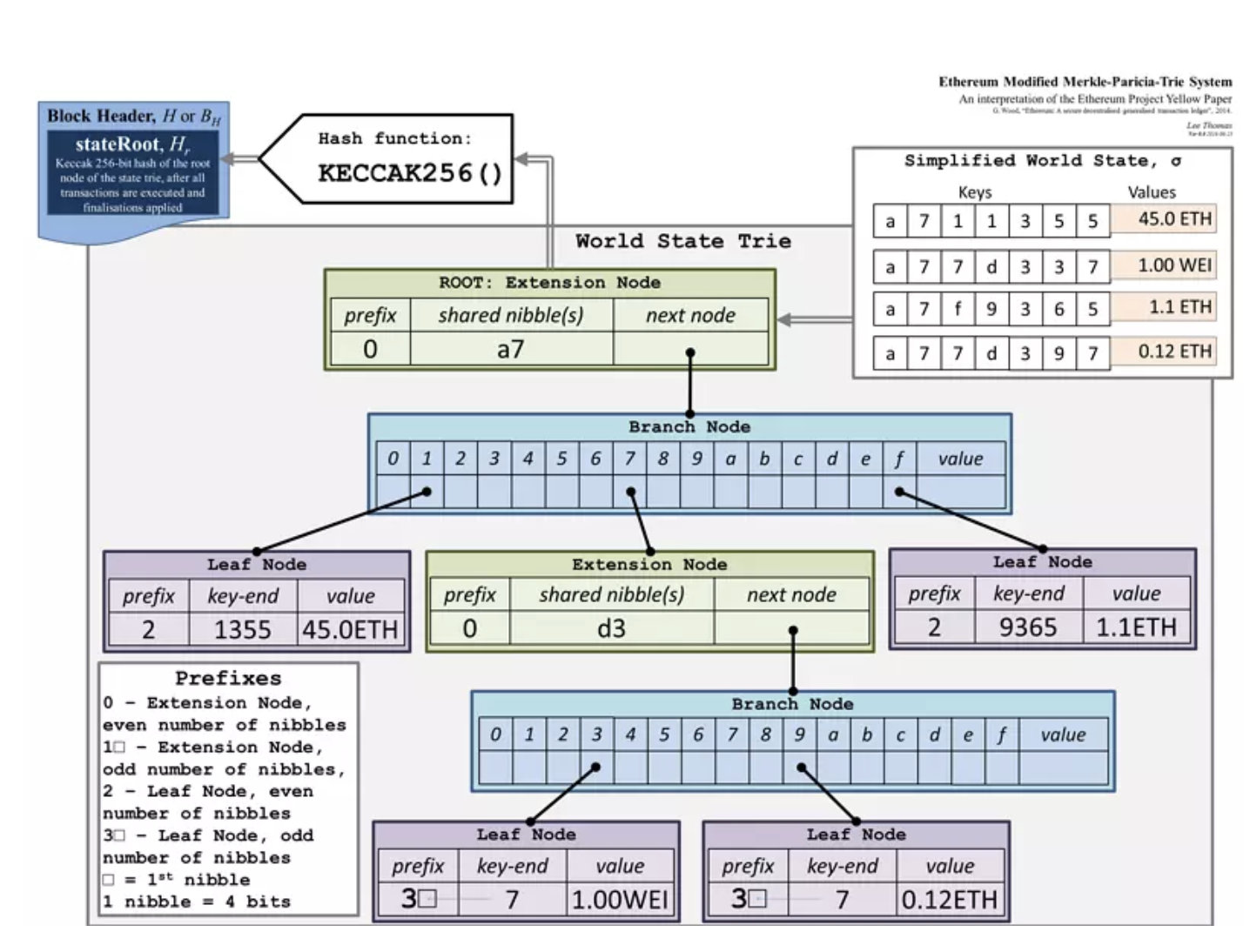
MPT树种还有一个重要的概念：特殊的十六进制前缀(hex-prefix, HP)编码来对key编码，我们先来了解一下编码定义规则：

* RAW 原始编码，对输入不做任何变更
* HEX 十六进制编码
  + RAW编码输入的每个字符分解为高4位和低4位。比如key=>"bob",b的ASCII十六进制编码为0x62，o的ASCII十六进制编码为0x6f，分解成高四位和第四位，16表示终结 0x10，最终编码结果为[6 2 6 15 6 2 16]，
  + 如果是叶子节点，则在最后加上Hex值0x10表示结束
  + 如果是分支节点不附加任何Hex值

HEX-Prefix 十六进制前缀编码

* 输入key结尾为0x10，则去掉这个终止符
* key之前补一个四元组这个Byte第0位区分奇偶信息，第1位区分节点类型
* 如果输入key的长度是偶数，则再添加一个四元组0x0在flag四元组后
* 将原来的key内容压缩，将分离的两个byte以高四位低四位进行合并
* 十六进制前缀编码相当于一个逆向的过程，比如输入的是[6 2 6 15 6 2 16]，根据第一个规则去掉终止符16。根据第二个规则key前补一个四元组，从右往左第一位为1表示叶子节点，从右往左第0位如果后面key的长度为偶数设置为0，奇数长度设置为1，那么四元组0010就是2。根据第三个规则，添加一个全0的补在后面，那么就是20.根据第三个规则内容压缩合并，那么结果就是[0x20 0x62 0x6f 0x62][3]

官方有一个详细的结构的示例：



# 4. Gas在智能合约中的应用

“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个气体，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天然气，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天然气。

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1000gas。 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。

起初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（以天然气计量）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天然气价格，那么我的交易就会产生如此多的计算工作，以至于合并后的天然气成本超过了我所附加的费用数额，那么这个天然气就会被计算为“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。 你可以把煤气价格看作矿工的小时工资，把煤气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。
* 如果您的天然气价格太低，没有人会处理您的交易
* 如果你的天然气价格没问题，但是你的交易的天然气成本超过了预算，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！[4]

# 5. 群组架构的好处

有别于传统区块链平台整个网络维护一个账本，所有节点参与到这个账本的共识和存储的做法，群组架构允许网络中存在多个不同的账本，每个账本是一个独立的小组，节点可以选择加入某些小组，参与到该组账本的共识和存储。该架构的特点是：

* 各群组独立执行共识流程，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，各群组拥有独立的账本，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，可以达成更好的隐私隔离；
* 机构的节点只需部署一次，通过群组设置即可参与到不同的多方协作业务中，或将一个业务按用户、时间等维度分到各群组，群组架构可快速地平行扩展，在扩大了业务规模同时，极大简化了运维复杂度，降低管理成本。[5]

# 6. 分布式存储有什么优势？

FISCO BCOS 2.0新增了对分布式数据存储的支持，节点可将数据存储在远端分布式系统中，克服了本地化数据存储的诸多限制。该方案有以下优点：

* 支持多种存储引擎，选用高可用的分布式存储系统，可以支持数据简便快速地扩容；
* 将计算和数据隔离，节点故障不会导致数据异常；
* 数据在远端存储，数据可以在更安全的隔离区存储，这在很多场景中非常有意义；
* 分布式存储不仅支持Key-Value形式，还支持SQL方式，使得业务开发更为简便；
* 世界状态的存储从原来的MPT存储结构转为分布式存储，避免了世界状态急剧膨胀导致性能下降的问题；
* 优化了数据存储的结构，更节约存储空间。[5]

# 7. 并行计算

过去将交易打包成一个区块，在一个区块中交易顺序串行执行的做法，在FISCO BCOS 2.0中，会通过一套并行交易处理模型实现自定义交易互斥变量。 区块执行过程中，系统将会根据交易互斥变量自动构建交易依赖关系图——DAG，基于DAG并行执行交易，最好情况下性能可提升数倍（取决于CPU核数）。[5]

# 8. 当前区块链实施的难度

# 8.1 开发过程更加严格也更慢

创建一个完全统一的系统并不是简单的任务。很小的失误都会使得整个数据库被破坏，或者导致一些数据库变得和其他的不同。当然，被破坏或者分裂的数据库再也没有任何统一性的保证。并且，所有这类系统都需要设计成从外部来看具有统一性。在区块链中，并没有“快速前进，打破常规）”的说法。如果你打破常规，你就失去了统一性，那么区块链就会变得破坏以及没有价值。

那么也许你会想，为什么你不能只是修复这个数据库，或者重新启动再推进了？在中心化的系统中，这很容易解决，但是在去中心化系统中却非常困难。你需要系统中所有参与者的共识或者同意，来对数据库进行修改。区块链应该是开源的，并且不被任何中心化单位所控制。

# 8.2 激励制度设计有很大难度

增加激励制度，并且保证系统中所有的参与者都不会攻击或者破坏数据库，是非常值得考虑的事情。区块链也许具有一致性，但是如果其中含有很多无用数据的时候，这就不会非常有用，因为将数据输入的成本会非常低。也不会有任何具有一致性的区块链会有用，如果其中不包含任何数据，因为将数据放入的代价非常高。

　　那么如何确保数据最终的准确性？你怎么确保奖励是和网络的目标是一致的呢？为什么节点要去维持或者更新数据，在有冲突的时候，他们为什么要选择其中一个？这些都是关于激励的问题，现在需要解答，而且不止是在开始的时候需要统一好，后续随着公司和技术的变化，也要时刻保持统一。不然区块链就没用了。

而且，也许你想知道为什么你不能修改一些有问题的激励模式。再一次强调下，这在中心化系统中很容易实现，但是在去中心化系统中，在没有达成共识的前提下，你不能修改任何东西。除非从每个人那边获得共识，不然你不能修改任何东西。

# 8.3 维护非常花费成本

传统的中心化数据库只需要写入一次就可以。但是区块链需要写入几千次，传统的中心化数据可以只需要一次性检测数据。区块链需要检查几千次数据。传统中心化数据库需要只需要一次性将数据转移到存储器。区块链则需要将数据转移几千次。

　　维护区块链的成本是数量级增长，而且这部分成本还需要根据使用性能进行更改。大多数应用都在追求之前说的一致性和稳定性等特性，但是如果使用完整性检查，收据和备份，在成本上就会便宜很多。

# 8.4 用户为王

这样会很好，因为公司不喜欢将用户的数据放在首要责任。但是如果用户做出什么不好的行为，这也非常不好，因为你无法将这个用户踢出，或者指出这样做会影响其他很多的便利性。

以上可以说明，激励制度的设计非常非常重要，可以使得用户不想去放弃，特别是如果对这个用户来说有利可图。也许你会想说，你可以简单地拒绝为欺诈者服务，这在中心化服务过程中很容易实现。但是，和中心化服务不同，在区块链上拒绝服务很困难因为没有任何个人有权利去将别人踢出。区块链必须要遵守软件中写下的规则。如果这个规则不能惩罚不好的行为，那么只能说你运气不好。区块链上，没有“法律”的精神，你只能花费很长时间去处理恶意行为。

# 8.5 维护非常花费成本

　强制升级并不是选项。网络上的其他人员没有义务去更改你的软件。如果他们这样做，那么不如打造更加容易，快速以及便宜的中心化系统。区块链的关键在于它不受控制于单个主体，同时这也和强制升级相违反。

　　但是，所有的升级都要和之前兼容。显然这很困难，特别是当你想添加新的特性时，如果从测试角度来看，就更加困难了。每个版本的软件都会添加很多的测试矩阵，也会花费更多的时间来进行释放。

# 8.6 扩容非常困难

最后，在扩容性方面至少要比传统中心化系统要难几个数量级。原因很明显。同样的数据需要在成百上千个地方，而不是像中心化那样，在同个地方。传输，验证和存储的开销是很大的，因为区块链上任何数据库的复制都需要花钱，而不像中心化的数据库那样，只需要付费一次就可以。[6]

# 参考文献：

# beanxyy. 一文读懂区块链技术逻辑和原理[J/OL]. <http://www.elecfans.com/blockchain/624019.html>, 2018-01-26/2019-07-09.

# BIBI. 公有链、联盟链、私有链，同一区块链不同演绎 [J/OL]. <https://zhuanlan.zhihu.com/p/33259928>, 2018-01-24/2019-07-09.

# JouyPub. 以太坊数据结构MPT [J/OL]. [https://www.jianshu.com/p/ 839153116857](https://www.jianshu.com/p/839153116857), 2018-01-26/2019-07-09.

# 小雨同学\_. 以太坊中智能合约调用的gas相关概念详解 [J/OL]. <https://blog.csdn.net/diandianxiyu_geek/article/details/78475639>, 2017-11-08/2019-07-09.

# weixin \_33724046. FISCO BCOS 2.0发布：新增群组架构克服吞吐瓶颈 [J/OL]. <https://blog.csdn.net/weixin_33724046/article/details/89190169>, 2019-03-20/2019-07-09.

# 巴比特. 说起来容易做起来难 区块链实施需要注意哪些？ [J/OL]. <http://finance.sina.com.cn/blockchain/roll/2018-06-11/doc-ihcufqif9848287.shtml>

# , 2018-06-11/2019-07-09.