**MPT研究之前期介绍**

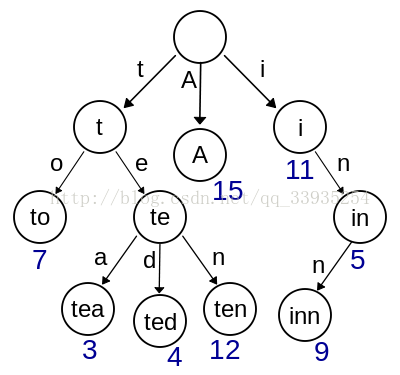
Merkle Patricia Trie是一种经过改良的、融合了默克尔树（Merkle Trie）和前缀树（Patricia Trie）两种树结构优点的数据结构，是以太坊中用来存储键值数据对（Key, Value）的重要树形数据结构。

由于MPT结合了默克尔树及前缀树的优势，因此，在对其进行介绍之前，我们会首先对这两种树进行简要的介绍，同时我们还会分析以太坊中需要借助到MPT进行存储管理的数据结构。

**一．Trie树**

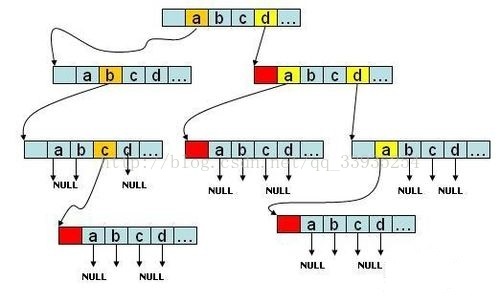
Trie树，又称前缀树或字典树，是一种有序树，用于保存关联数组，其中的键通常是字符串。与二叉查找树不同，键不是直接保存在节点中，而是由节点在树中的位置决定。一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。一般情况下，不是所有的节点都有对应的值，只有叶子节点和部分内部节点所对应的键才有相关的值。

在图示中，键标注在节点中，值标注在节点之下。每一个完整的英文单词对应一个特定的整数。键不需要被显式地保存在节点中。图示中标注出完整的单词，只是为了演示trie的原理。trie中的键通常是字符串，但也可以是其它的结构。



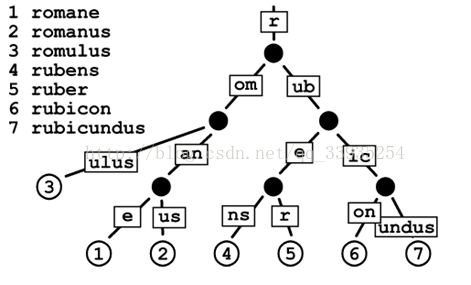
上图是一个简略视图，实际上trie每个节点是一个确定长度的数组，数组中每个节点的值是一个指向子节点的指针，最后有个标志域，标识这个位置为止是否是一个完整的字符串，并且有几个这样的字符串。

常见的用来存英文单词的trie每个节点是一个长度为27的指针数组，index0-25代表a-z字符，26为标志域。如图：



**二．Patricia树**

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。



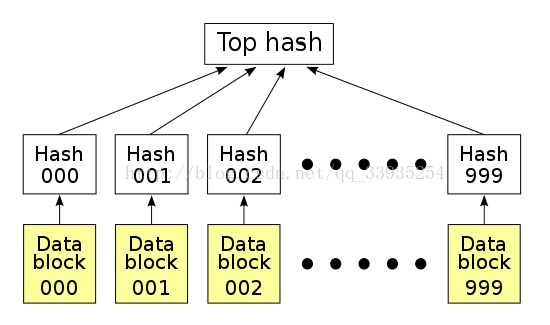
**三．Merkle树**

Merkle Tree，通常也被称作Hash Tree，顾名思义，就是存储hash值的一棵树。Merkle树的叶子是数据块(例如，文件或者文件的集合)的hash值。非叶节点是其对应子节点串联字符串的hash。

要了解Merkle Tree就要先从Hash List说起：

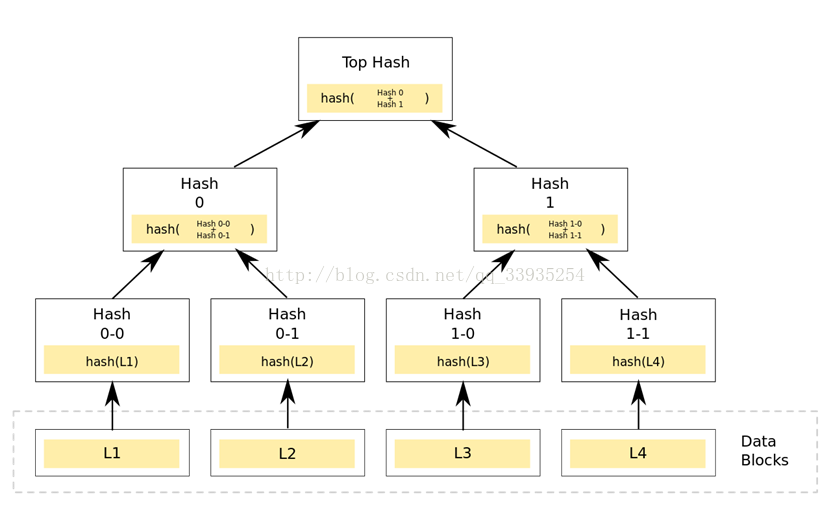
在点对点网络中作数据传输的时候，会同时从多个机器上下载数据，而且很多机器可以认为是不稳定或者不可信的。为了校验数据的完整性，更好的办法是把大的文件分割成小的数据块（例如，把分割成2K为单位的数据块）。这样的好处是，如果小块数据在传输过程中损坏了，那么只要重新下载这一快数据就行了，不用重新下载整个文件。

怎么确定小的数据块没有损坏哪？只需要为每个数据块做Hash。BT下载的时候，在下载到真正数据之前，我们会先下载一个Hash列表。那么问题又来了，怎么确定这个Hash列表是正确的？答案是把每个小块数据的Hash值拼到一起，然后对这个长字符串在作一次Hash运算，这样就得到Hash列表的根Hash(Top Hash or Root Hash)。下载数据的时候，首先从可信的数据源得到正确的根Hash，就可以用它来校验Hash列表了，然后通过校验后的Hash列表校验数据块。



Merkle Tree可以看做Hash List的泛化（Hash List可以看作一种特殊的Merkle Tree，即树高为2的多叉Merkle Tree。

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。但是往上走，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。于是往上推，依然是一样的方式，可以得到数目更少的新一级哈希，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。



在p2p网络下载网络之前，先从可信的源获得文件的Merkle Tree树根。一旦获得了树根，就可以从其他从不可信的源获取Merkle tree。通过可信的树根来检查接受到的Merkle Tree。如果Merkle Tree是损坏的或者虚假的，就从其他源获得另一个Merkle Tree,直到获得一个与可信树根匹配的MerkleTree。

Merkle Tree和Hash List的主要区别是，可以直接下载并立即验证Merkle Tree的一个分支。因为可以将文件切分成小的数据块，这样如果有一块数据损坏，仅仅重新下载这个数据块就行了。如果文件非常大，那么Merkle tree和Hash list都很到，但是Merkle tree可以一次下载一个分支，然后立即验证这个分支，如果分支验证通过，就可以下载数据了。而Hash list只有下载整个Hash list才能验证。

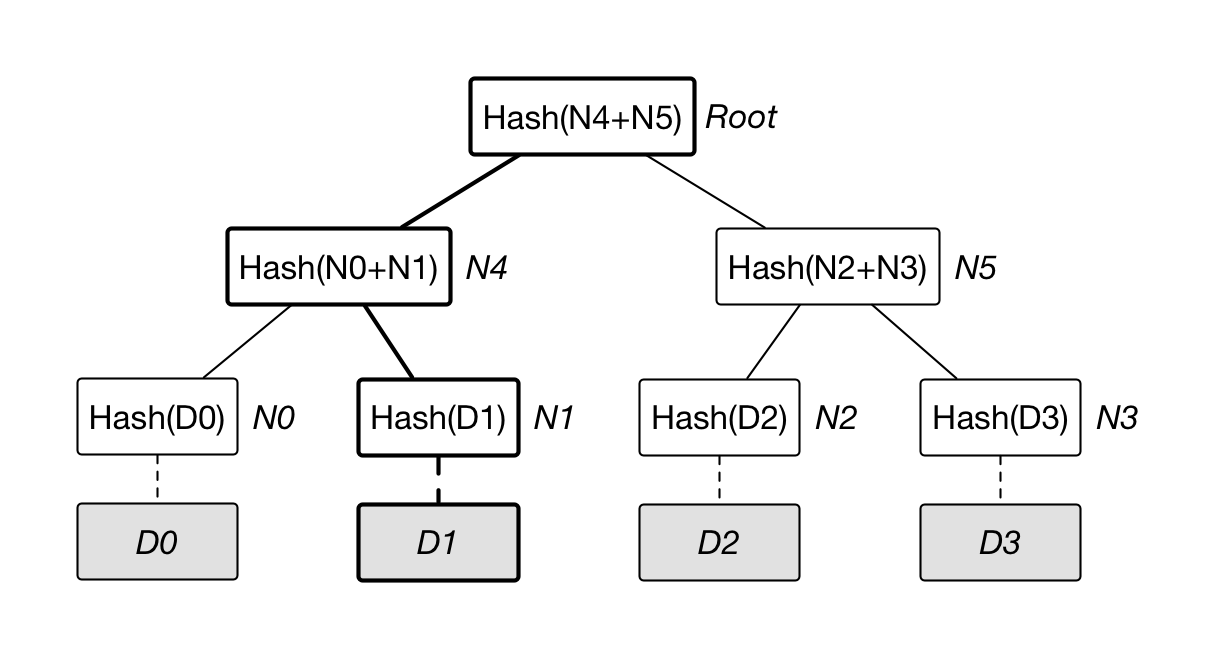
1.主要特点

(1)最下面的叶节点包含存储数据或其哈希值。

(2)非叶子节点（包括中间节点和根节点）都是它的两个孩子节点内容的哈希值。

2.典型应用

对于如下图所示的默克尔树：



它的典型应用包括如下几个场景：

(1)快速比较大量数据

当两个默克尔树根相同时，则意味着所代表的两组数据必然相同。否则，必然不同。由于 Hash 计算的过程可以十分快速，预处理可以在短时间内完成。利用默克尔树结构能带来巨大的比较性能优势。

(2)快速定位修改（错误）

如果 D1 中数据被修改，会影响到 N1，N4 和 Root。因此，一旦发现某个节点如 Root 的数值发生变化，沿着 Root --> N4 --> N1，最多通过 O(lgN) 时间即可快速定位到实际发生改变的数据块 D1。

(3)证明某个集合中存在或不存在某个元素

假设我要验证 D0 在集合中，那么我只需要获得 N1 和 N5，再计算得到根节点的哈希。最后与公开的 Root 进行比较，我就可以知道 D0 是否包含在集合之中。

在以太坊中，利用默克尔树可以进行轻节点扩展，对于每个区块，仅仅需要存储约80个字节大小的区块头数据，而不存储交易列表，回执列表等数据，便可以验证一笔交易是否被包含在这个区块当中。同时利用默克尔树也可以实现快速重哈希，当树节点内容发生变化时，能够在前一次哈希计算的基础上，仅仅将被修改的树节点进行哈希重计算，便能得到一个新的根哈希用来代表整棵树的状态

**四．MPT（Merkle Patricia Tree）树**

了解了Merkle Tree和Patricia Tree，顾名思义，MPT（Merkle Patricia Tree）就是这两者混合后的产物。

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；

2.分类

由于MPT可以存储任意的键值对数据，因此，在以太坊的实现过程中，它将会借助这棵树来记录世界状态，账户存储内容，交易以及交易收据。即以太坊中存在四种MPT，分别为：

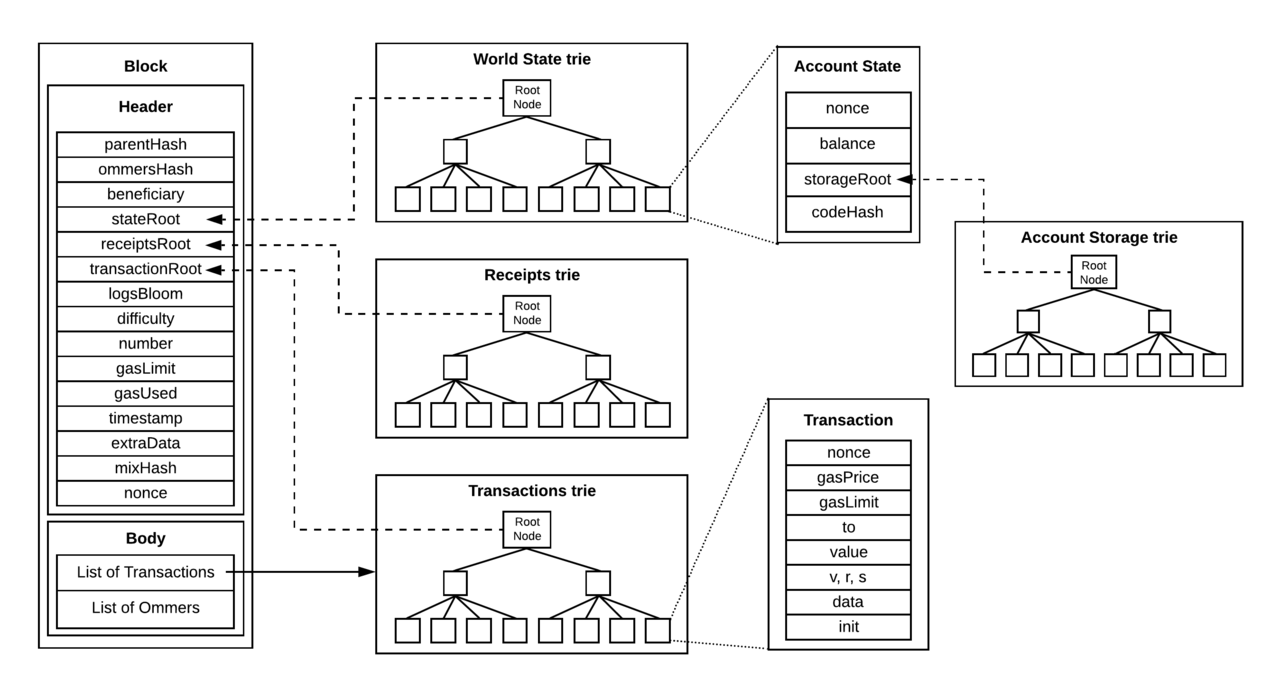
(1)世界状态树包括了从地址到账户状态之间的映射。 世界状态树的根节点哈希值由区块保存（stateRoot字段），它标识了区块创建时的当前状态。整个网络中只有一个世界状态树。

(2)账户存储树保存了与某一智能合约相关的数据信息。账户存储树的根节点由账户状态保存（storageRoot字段）。每个账户都有一个账户存储树。

(3)交易树记录了一个区块中的所有交易信息。交易树的根节点哈希值由区块保存（transactionsRoot字段），它是当前区块内所有交易组成的树。每个区块都有一棵交易树。

(4)交易收据树记录了一个区块中的所有交易收据信息。同样由区块保存（receiptsRoot字段），它是当前区块内所有交易收据组成的树。每个区块都有一棵交易收据树。

我们可以用一张图来概括上面的四种MPT：



3.重要概念

在以太坊（ethereum）中，使用了一种特殊的十六进制前缀(hex-prefix, HP)编码，所以在字母表中就有16个字符。这其中的一个字符为一个nibble。

(1)节点

MPT树中的节点包括空节点、叶子节点、扩展节点和分支节点:

A.空节点，简单的表示空，在代码中是一个空串。

B.叶子节点（leaf），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。

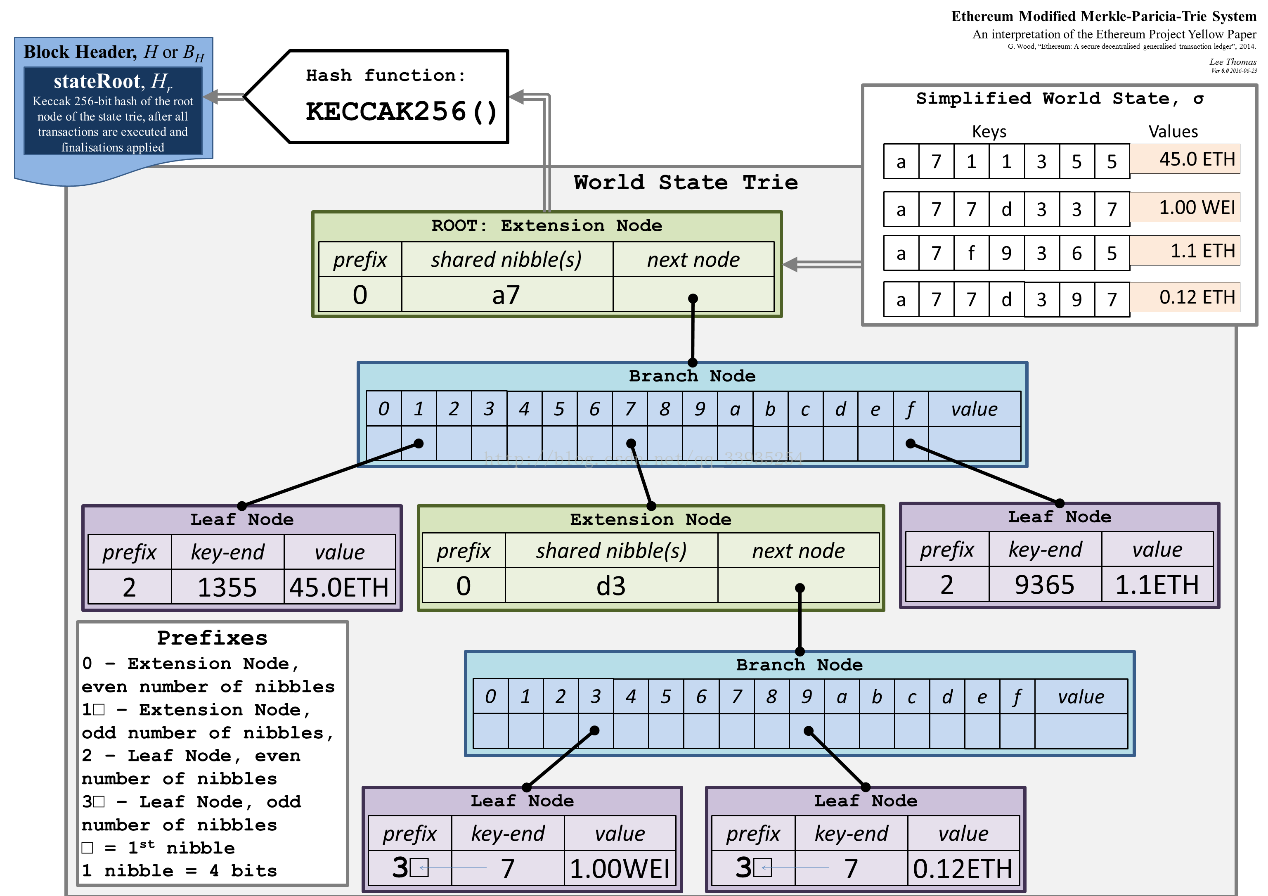
C.扩展节点（extension），也是[key，value]的一个键值对，但是这里的value是其他节点的hash值，这个hash可以被用来查询数据库中的节点。也就是说通过hash链接到其他节点。

D.分支节点（branch），因为MPT树中的key被编码成一种特殊的16进制的表示，再加上最后的value，所以分支节点是一个长度为17的list，前16个元素对应着key中的16个可能的十六进制字符，如果有一个[key,value]对在这个分支节点终止，最后一个元素代表一个值，即分支节点既可以搜索路径的终止也可以是路径的中间节点。

(2)编码

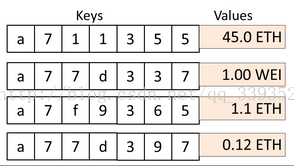
MPT树中另外一个重要的概念是一个特殊的十六进制前缀(hex-prefix, HP)编码，用来对key进行编码。因为字母表是16进制的，所以每个节点可能有16个孩子。因为有两种[key,value]节点(叶节点和扩展节点)，引进一种特殊的终止符标识来标识key所对应的是值是真实的值，还是其他节点的hash。如果终止符标记被打开，那么key对应的是叶节点，对应的值是真实的value。如果终止符标记被关闭，那么值就是用于在数据块中查询对应的节点的hash。无论key奇数长度还是偶数长度，HP都可以对其进行编码。最后我们注意到一个单独的hex字符或者4bit二进制数字，即一个nibble。

HP编码很简单。一个nibble被加到key前（下图中的prefix），对终止符的状态和奇偶性进行编码。最低位表示奇偶性，第二低位编码终止符状态。如果key是偶数长度，那么加上另外一个nibble，值为0来保持整体的偶特性。

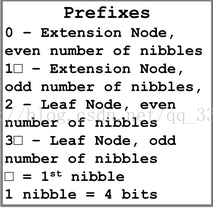


如图所示，总共有2个扩展节点，2个分支节点，4个叶子节点。

其中叶子结点的键值情况为：



节点的前缀：



参考文献：https://blog.csdn.net/qq\_55179414/article/details/118935827

参考文献：https://blog.csdn.net/qq\_33935254/article/details/55505472