在以太坊中，有三棵树的说法，分别是状态树、收据树和交易树。了解了这三棵树，就弄清楚了以太坊的基础数据结构设计。

1. **引入**

首先，我们要实现从账户地址到账户状态的映射。在以太坊中，账户地址为160字节，表示为40个16进制数额。状态包含了余额(balance)、交易次数(nonce),合约账户中还包含了code(代码)、存储(stroge)。

直观地来看，其本质上为Key-value键值对，所以直观想法便用哈希表实现。若不考虑哈希碰撞，查询直接为常数级别的查询效率。但采用哈希表，难以提供Merkle proof。

1. **思考如何组织账户的数据结构**
2. 我们能否像BTC中，将哈希表的内容组织为Merkle Tree？

但当新区块发布，哈希表内容会改变，再次将其组织为新的Merkle Tree?如果这样，每当产生新区块(ETH中新区块产生时间为10s左右)，都要重新组织Merkle Tree，很明显这是不现实的。

需要注意的是，比特币系统中没有账户概念，交易由区块管理，而区块包含上限为4000个交易左右，所以Merkle Tree不是无限增大的。而ETH中，Merkle Tree来组织账户信息，很明显其会越来越庞大。实际中，发生变化的仅仅为很少一部分数据，我们每次重新构建Merkle Tree代价很大

1. 那我们不要哈希表了，直接使用Merkle Tree，每次修改只需要修改其中一部分即可，这个可以吗？

实际中，Merkle Tree并未提供一个高效的查找和更新的方案。此外，将所有账户构建为一个大的Merkle Tree，为了保证所有节点的一致性(不同节点组建Merkle Tree的交易顺序不同，得到的根哈希就不同)和查找速度，必须进行排序。

|  |
| --- |
| 注意：BTC系统中，虽然每个节点构建的Merkle Tree不一致（不排序），但最终是获得记账权的节点的Merkle Tree才是有效的。 |

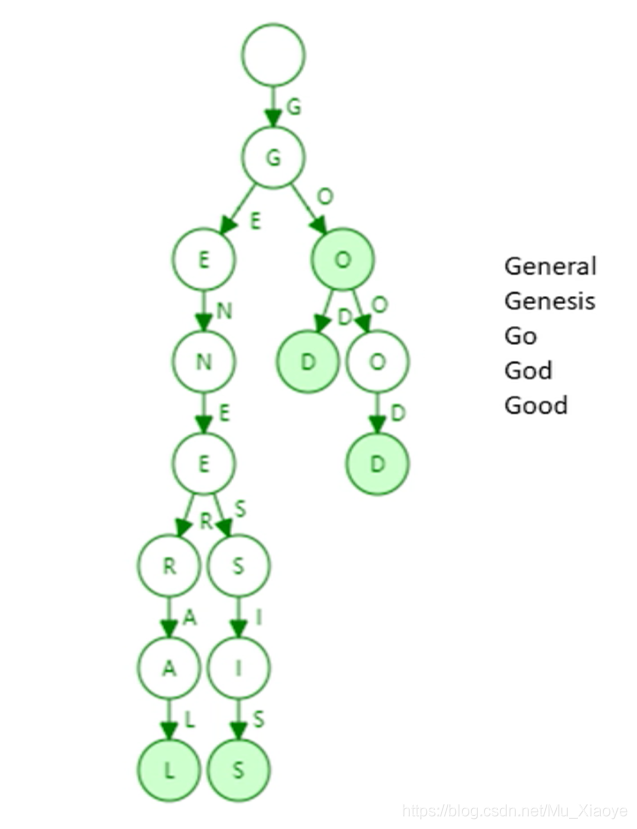
1. 那么经过排序，使用Sorted Merkle Tree可以吗？

新增账户，由于其地址随机，插入Merkle Tree时候很大可能在Tree中间，发现其必须进行重构。所以Sorted Merkle Tree插入、删除(实际上可以不删除)的代价太大。

既然哈希表和 Merkle Tree都不可以，那么我们看一下实际中以太坊采取的数据结构：MPT。但在此之前需要先介绍两种简单的树：

1. **一个简单的数据结构——trie(字典树、前缀树)**

如下为一个通过5个单词组成的trie数据结构（只画出key，未画出value）



特点：

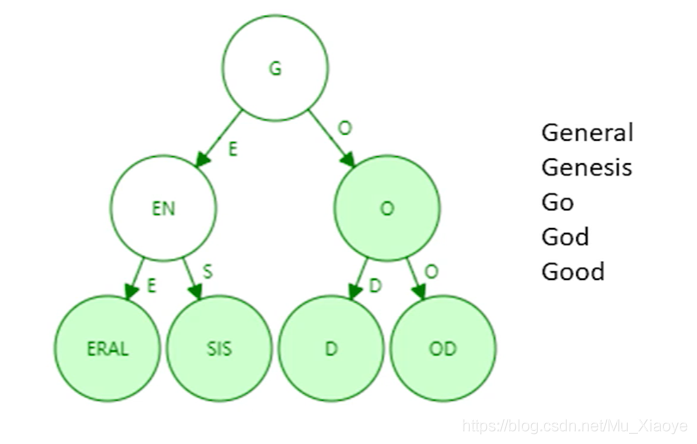
* trie中每个节点的分支数目取决于Key值中每个元素的取值范围(图例中最多26个英文字母分叉+一个结束标志位)。
* trie查找效率取决于key的长度。实际应用中（以太坊地址长度为160byte）。
* 使用哈希表会出现碰撞，而trie上面不会发生碰撞。
* 给定输入，无论如何顺序插入，构造的trie都是一样的。
* 更新操作局部性较好

那么trie有缺点吗？当然有：

trie的存储浪费。很多节点只存储一个key，但其“儿子”只有一个，过于浪费。因此，为了解决这一问题，我们引入Patricia tree/trie

1. Patricia trie(Patricia tree)

Patricia trie就是进行了路径压缩的trie。如上图例子，进行路径压缩后如下图所示：



需要注意的是，如果新插入单词，原本压缩的路径可能需要扩展开来。那么，需要考虑什么情况下路径压缩效果较好？树中插入的**键值分布较为稀疏的情况**下，可见**路径压缩效果较好**。在以太坊系统中，160位的地址存在2^160 种，该数实际上已经非常大了，和账户数目相比，可以认为地址这一键值非常稀疏。

**因此，我们可以在以太坊账户管理种使用Patricia tree这一数据结构！但实际上，在以太坊种使用的并非简单的PT(Patricia tree),而是MPT(Merkle Patricia tree)。**

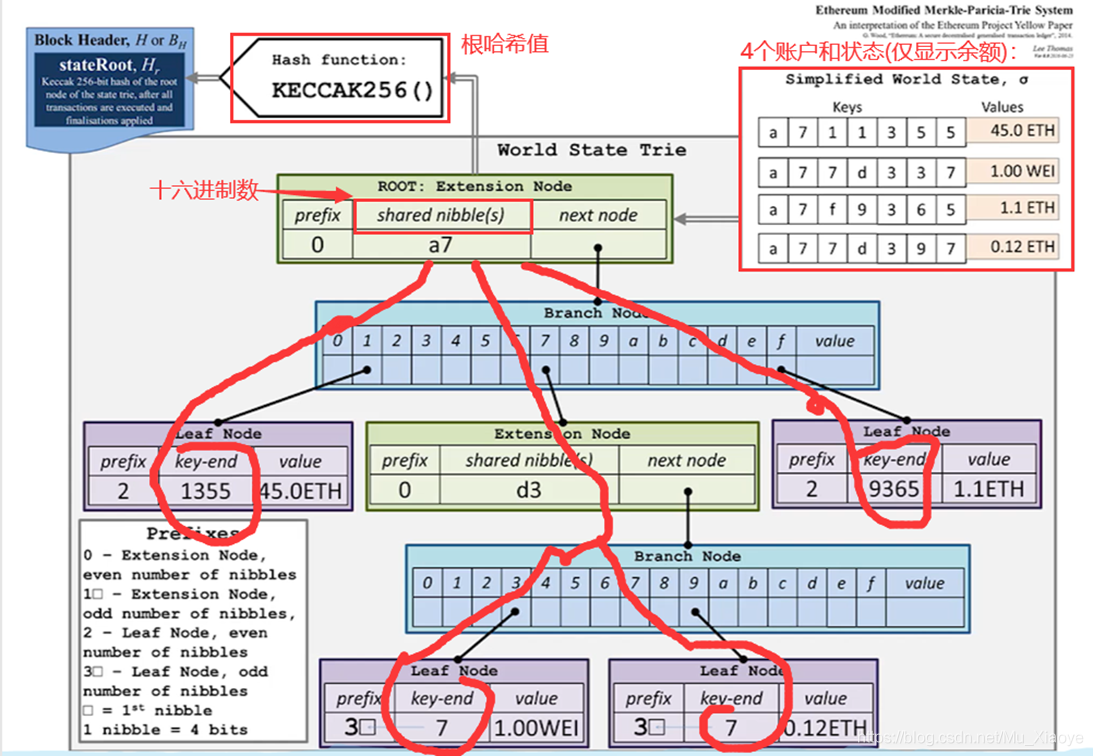
1. **MPT(Modified Patricia tree)**

以太坊系统将所有账户组织为一个经过路径压缩和排序的Merkle Tree，其根哈希值存储于block header中。BTC系统中只有一个交易组成的Merkle Tree，而以太坊中有三个(三棵树)。也就是说，在以太坊的block header中，存在有三个根哈希值。

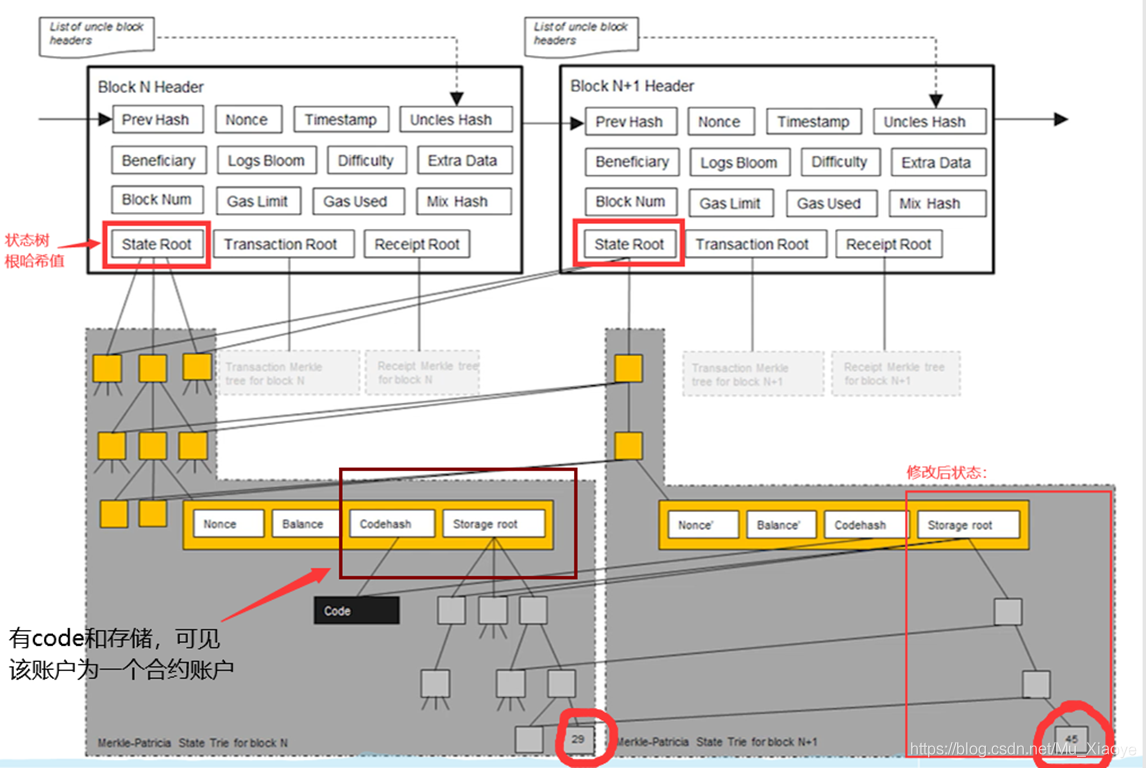
**根哈希值的用处：**

1. 防止篡改。
2. 提供Merkle proof，可以证明账户余额，轻节点可以进行验证。
3. 证明某个发生了交易的账户是否存在

下图为以太坊中使用的MPT结构示意图。右上角表示四个账户(为了更直观显示原理，账户keys值只显示较少位)和其状态(只显示账户余额)。（需要注意这里的指针都是哈希指针）

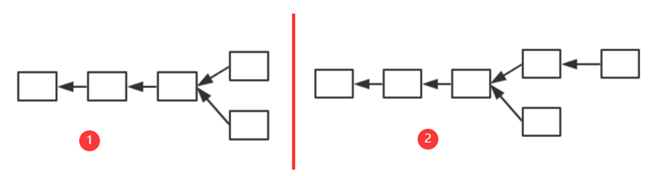


每次发布新区块，状态树中**部分节点状态会改变**。但改变**并非在原地修改，而是新建一些分支，保留原本状态**。如下图中，仅仅有新发生改变的节点才需要修改，其他**未修改节点直接指向前一个区块中的对应节点**。(下图中只有最右边一个节点的storage 树的最右边节点余额由29变为45)



所以，系统中全节点并非维护一棵MPT，而是每次发布新区块都要新建MPT。只不过MPT中的大部分节点共享(或者说是继承)。

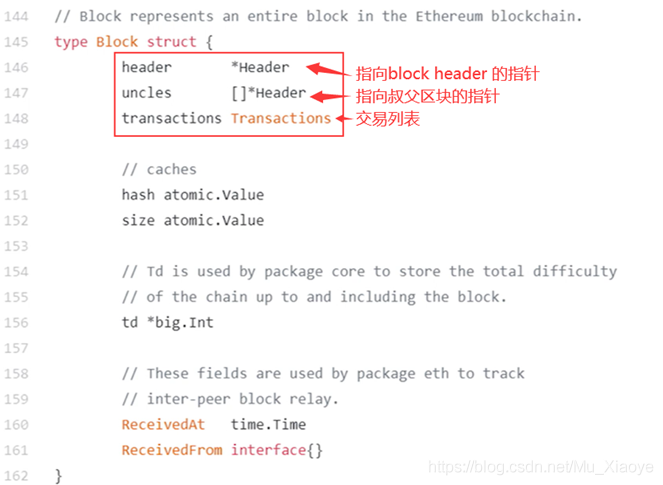
|  |
| --- |
| 为什么MPT要保存原本状态？为何不直接修改？  答案是为了便于回滚。如下1中产生分叉，而后上分叉中节点胜出，变为2中状态。那么，下分叉中节点中状态的修改便需要进行回滚并继续扩展上分叉。因此，需要维护这些历史记录。 |



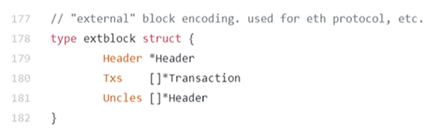
1. **通过代码看以太坊中的数据结构**
   1. block header 中的数据结构



* 1. 区块结构



* 1. 区块在网上真正发布时的信息



|  |
| --- |
| **最后说明** 状态树中保存Key-value对，key就是地址，而value状态通过**RLP**(Recursive Length Prefix，一种进行序列化的方法)编码序列号之后再进行存储。 |