**分析涉及Merkle Patricia Trie的以太坊源代码**

**概述：**

Merkle Patricia Trie简称MPT，MPT分别是由三种树组合而成的数据结构，有Trie树、Patricia Trie以及Merkle树。

Trie树，又名为字典树、单词查找或者前缀树，是一种用于快速查找的多分叉树结构，如数字的Trie树就是一个十叉树。Trie树的特点有三个，分别为若根节点不包含字符，那除根节点之外每个节点只包含一个字符、从根节点到某个节点，路径经过的节点连上就为该节点的字符串、每个节点的所有子节点包含的字符串都不同。

Patricia Trie，又名前缀树，是从Trie树优化而来的。如果存在一个父节点只有一个子节点，那么父节点将会与子节点合并，这样就可以减少存储空间而且也提高了查找效率。

Merkle树，又名Hash树，是存储哈希值(hash)的树。Merkle树的叶子都是用于存储数据（hash值），而父节点则是存储相对应的子节点串联字符串的hash值，一级接一级地往上直到根节点。因此，若是修改了子节点的数据，则对应的父节点和hash值就会改变，根节点也会随之改变，所以这Merkle树就是为了保证数据不被篡改而设计的技术。

以太坊的MPT（Merkle Patricia Trie）是结合了Merkle树和Patricia树的数据结构，所以同时拥有着查询和安全性的特点。以太坊的区块头中有三个字段是包含了MPT树相关的，分别是交易树（txTrieRoot）、收据树（receiptTrieRoot）、状态树（stateRoot），这些树分别存储了这三颗树的树根hash值。

以太坊对这两特点进行优化与改进：

1. **保证树的安全性**

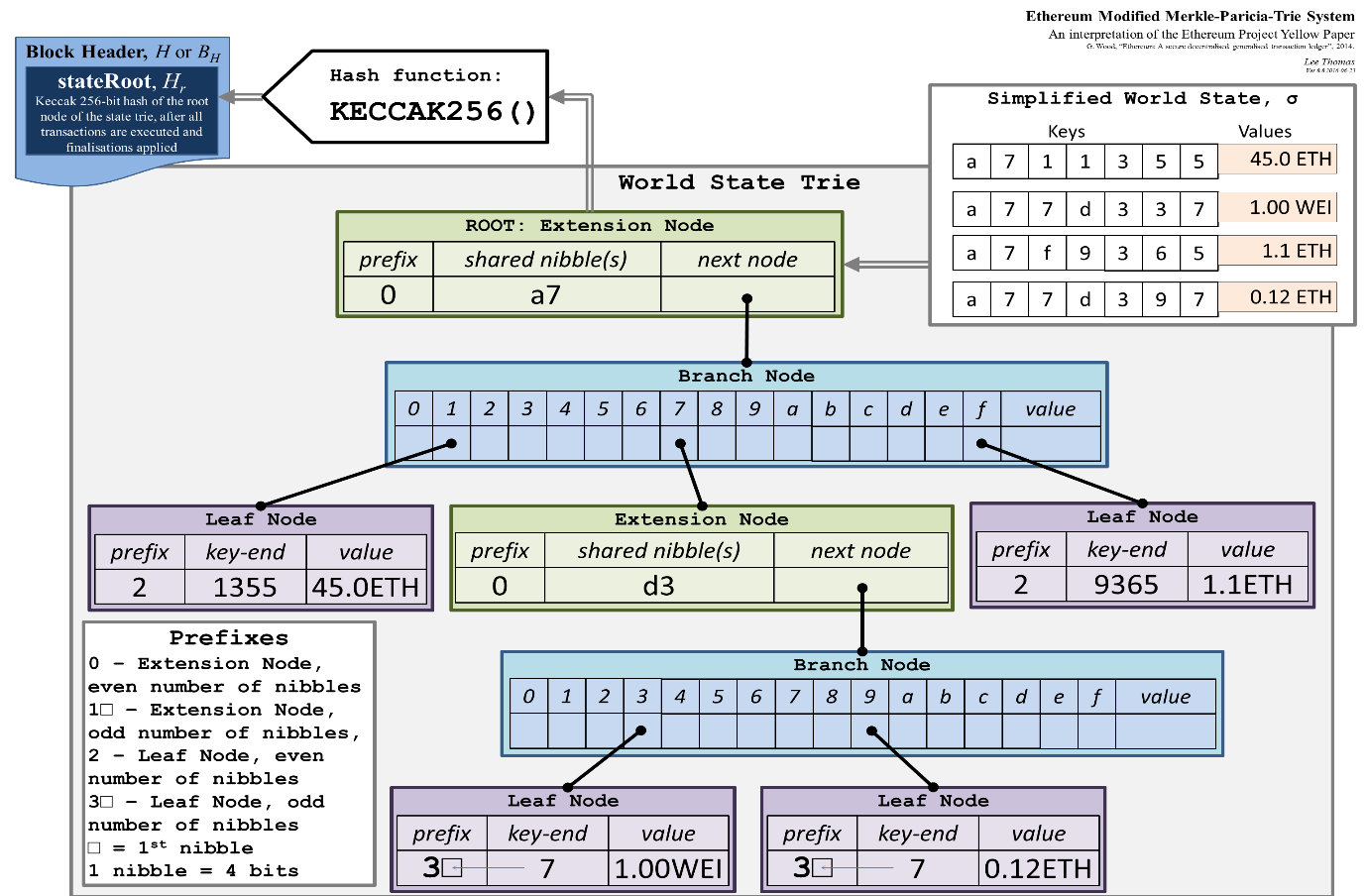
节点与节点之间的联系不再采用内存指针的方式，而是采用hash值得方式。比如由一个父节点存储一个值然后再执行下一个节点的hash值，然后将这个hash值与实际对应的节点存储在数据库中。当有人篡改此节点的值时，父节点的hash值也会被篡改直到根节点也跟着修改，所以我们只需要验证根节点的hash值就可以知道以下节点的数据有没有被篡改过。

1. **引入更多节点来提高安全性**

MPT节点分为4种节点，各个节点存在的功能不同：

1. 空节点（NULL），一个空串
2. 叶节点（Leaf Node），包含两项数据（key，value），key为16进制编码，value是RLP编码。
3. 扩展节点，value是其他节点散列值，通过这个散列值去链接到其他的节点。
4. 分支节点，长度为17的列表，key还是16进制编码，加上value，前16个元素对应key的16个十六进制字符，如果一个键值在这个分支终止了，那么最后的一个元素表示为一个值。分支节点既是搜索的终止也可是路径的中间点。

MPT简单的结构图：



**网上截取的图**

**数据结构**

MPT的数据结构可分为Node结构和Trie结构。Node结构可分为四种类型，分别是fullNode、shortNode、hashNode、valueNode，fullNode对应的是以上提到的分支节点，shortNode对应的则是叶子节点和扩展节点，这俩类型都为结构性节点。那么除了结构性节点外，还有数据节点也就是valueNode和hashNode，valueNode只用于存储value，hashNode则保存的是key，key的对应数据都保存在数据库中。



Trie结构，root为当前根节点，db是后端数据库的KV存储，trie的结构最终都是需要通过KV的形式存储到数据库里去，然后需要时才从数据库里加载。rootHash是用于存储hash值的，通过这个hash值可以在数据库里实现出整颗trie树。Cache表示是当前trie树的时代，每次调用Commit操作的时候，会增加Trie树的Cache时代。Cache时代会被附加在Node节点上，如果当前的Cache时代 – Cachelimit参数大于Node的Cache时代，则Node会从Cache里面卸载，以节约内存。****

创建一个新的trie树，初始化trie树调用New函数，函数接收一个hash值和Database参数，如果hash值不是空（!NULL），说明是从数据库加载的一个已经存在的trie树，就调用trie.resolveHash方法来加载整颗trie树。如果hash值为空，则新建一颗trie树返回。



**操作流程**

组成MPT树结构的基本操作流程分别有三种，是查找、插入或更新、删除等。

1. 查找（Get方法）



Get操作流程是为了查找Key的值（Raw编码转换成Hex编码），得到的值为搜索路径，从根节点开始搜索与搜索路径内容相同的路径。

Case节点：

若当前节点为叶子节点（valueNode），存储的内容为数据项的内容，且搜索路径的内容与叶子节点的key相同，则表示找到该节点，反之没找到则该节点在树中不存在。

若当前节点为hash结点（hashNode）且存储的内容是hash值，则利用hash索引从数据库中加载该节点，再将搜索路径作为参数，对新解析出来的节点利用递归调用查找(Get)函数。

若当前节点为扩展节点（shortNode），存储的内容是另外一个节点的引用，且当前节点的key是搜索路径的前缀，则将搜索路径减去当前节点的key，将剩余的搜索路径作为参数，对其子节点递归地调用查找函数；若当前节点的key不是搜索路径的前缀，表示该节点在树中不存在。

若当前节点为分支节点（fullnode），若搜索路径为空，则返回分支节点的存储内容，反之利用搜索路径的第一个字节选择分支节点的子节点，将剩余的搜索路径作为参数递归调用查找函数。

1. 更新/插入

Update是insert和delete的结合。当用户调用Update函数的时候，若value不为空，则转换为insert函数调用；若value为空，则转换为delete函数调用。

Insert插入过程是根据Get函数调用后，首先找到与新插入节点拥有最长相同路径前缀的节点，记为Node，又可分为几个Case节点：

若Node为叶子节点/扩展结点（shortNode），如果剩余的搜索路径与当前节点的key一致，则把当前节点Val更新。如果剩余的搜索路径与当前节点的key不完全一致，则将该节点的子节点替换成分支节点，将新节点与当前节点key的共同前缀作为当前节点的key，将新节点与当前节点的子节点作为两个子节点插入到分支节点的子列表中，同时当前节点转换成一个扩展节点；若新节点与当前节点没有共同前缀，则直接用生成的分支节点替换当前节点。若插入成功，修改节点的dirty为true，hash置空，将节点的标记更新为现在的。

若Node为分支节点（fullNode）,如果剩余的搜索路径不为空，则将新的节点作为一个叶子节点插入到对应的子列表中；如果剩余的搜索路径为空，则将新节点的内容存储在分支节点的第17个子节点项中的Value值。

若当前是一个空树（nil），直接构造一个shortNode。

若当前为hash节点（hashNode），表示当前节点还没有加载到内存里面来，还是存放在数据库里面，那么就调用t.resolveHash(n,prefix)方法来加载到内存，然后对加载出来的节点调用insert方法来进行插入。





1. 删除

Delete删除过程是根据Get函数调用后，找到与需要插入的节点拥有最长相同路径前缀的节点，记为Node，又可分为几个Case节点：

若Node为叶子节点/扩展结点(shortNode)，如果剩余的搜索路径与Node节点的key完全一致，则将整个Node删除。如果Node的key是剩余搜索路径的前缀，则对该节点的Val做递归删除调用，否则说明要删除的节点不存在于树中，删除失败

若Node节点为分支节点（fullNode），删除孩子列表中相应下标的节点。删除结束后，若Node的孩子个数只剩下一个，那么将分支节点替换成叶子节点或扩展节点。若删除成功，则将被修改节点的dirt改为true，hash置放为空，且将节点的标记更新为现在的。

若Node节点为叶子节点（valueNode），直接返回。

若Node节点为空（nil），返回空。

若当前为hash节点（hashNode），表示当前节点还没有加载到内存里面来，还是存放在数据库里面，那么就调用t.resolveHash(n,prefix)方法来加载到内存，然后对加载出来的节点调用delete方法来进行删除。









**MFT的作用**

1. MPT可以存储任意长度的key和value值数据
2. MPT提供了一种快速计算所维护数据集哈希标识的机制，即在节点哈希计算之前会对该节点的状态进行判断，只有当该节点的内容变脏，才会进行哈希重计算、数据库持久化等操作。如此一来，在某一次事务操作中，对整棵MPT树的部分节点的内容进行了修改，那么一次哈希重计算，仅需对这些被修改的节点、以及从这些节点到根节点路径上的节点进行重计算，便能重新获得整棵树的新哈希。
3. MPT提供了快速状态回滚的机制，即区块链内容发生了重组织，链头发生切换或者是区块链的世界状态（账户信息）需要进行回滚，即对之前的操作进行撤销。
4. 以太坊提供了默克尔证明的证明方法，进行轻节点的扩展，实现简单的支付验证，即在无需维护具体交易信息的前提下，证明某一笔交易是否存在于区块链中。

**学号：3190300677**

**姓名：姚熙源**