research report on MPT

MPT 研究报告

2022 年 7 月 20 日

目录

1 MPT 简介

MPT，全称为 Merkel-Patricia Tree。是一个基于密码学验证的数据结构，用于存储键值对关系。从其全称中可以看出可以看出 MPT 是以 PatriciaTree 和 MerkelTree 为基础构建出来的，而 MPT 也同时拥有这两种数据结构的优点，可以实现数据校验，同时可以快速查找并节省存储空间。

MPT 树可以实现以下几个功能：

(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 MPT 实现基础

2.1 前缀树 (Prefix tree)

前缀树，又称字典树，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。如下图所示，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。

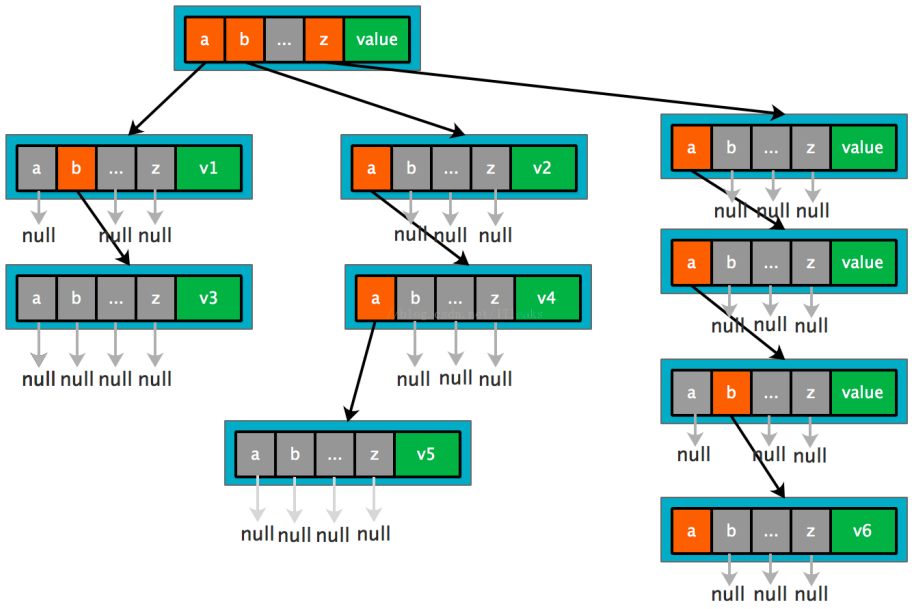


图 1: 前缀树存储图

2.2 帕特里夏树（(Patricia Tree)

帕特里夏树是一种基数树，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。而如果基数树的基数为 2 或 2 的整数次幂，就被称为“帕特里夏树”。有时也直接认为帕特里夏树就是基数树以太坊中采用 Hex 字符作为 key 的字符集，也就是基数为 16 的基数树，每个节点最多可以有 16 个子节点。帕特里夏树对比基础的前缀树而言，压缩了其冗长的层级关系，避免不必要的空间浪费，同时优化了访问效率，实现了快速查找。但是其无法实现数据校验，无法提供安全性认证，因此无法直接用于区块链。

2.3 默克尔树 (Merkel Tree)

Merkle Tree，通常也被称作 Hash Tree，顾名思义，就是存储 hash 值的一棵树。如下图所示，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。实现时用每个节点的 hash 值来建立对应的关系，底层的叶子节点都算一个 hash，这是一个二叉树，两两 hash 之间再算一次 hash，不断往上计算得出 top hash 算作一个根节点存到区块里面。

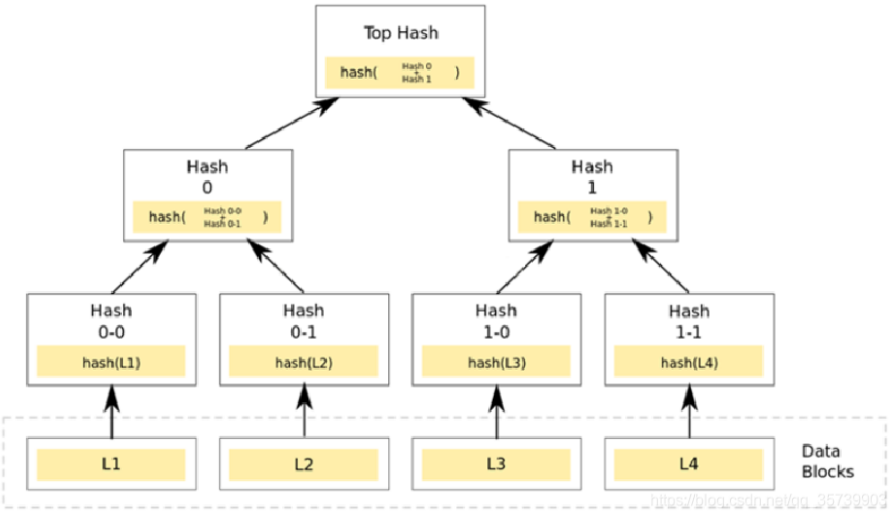


图 2: 默克尔树存储图

默克尔树存在两个优点：1、当整个树中的其中任意一个节点发生变化，那么整个树都会发生变化，这样很容易就可以检测到变化。2、当需要验证某个叶子节点时，不需要获取整个树的数据，只需要与某个叶子节点相关的节点就可以进行验证。减少了下载量和计算量。

在应用 Merkle Tree 时，因为以太坊要去做 hash 的是整个要存

储内容的 RLP 编码，所以以太坊相当于把自己的 value 先做 RLP 编码，然后再去求 hash，然后把最后得到的 hash 值作为在数据库中存储的位置，所以在 MPT 中的节点里面用 hash 作为 key，访问的的时候根据 hash 在数据库中找到对应的值。

3 MPT 结构构建

3.1 MPT 特点

以太坊不同于比特币的 UXTO 模型，在账户模型中，账户存在多个属性（余额、代码、存储信息），属性（状态）需要经常更新。因此需要一种数据结构来满足几点要求：

① 在执行插入、修改或者删除操作后能快速计算新的树根，而无需重新计算整个树，并且可以快速检验节点的正确性。

② 即使攻击者故意构造非常深的树，它的深度也是有限的。否则，攻击者可以通过构建足够深的树使得每次树更新变得极慢，从而执行拒绝服务攻击。

③ 对节点的访问效率要求较高。

要求 ① 可以通过默克尔树，但要求 ②③ 并不是默克尔树的优势。对于要求 ②，可将数据 Key 进行一次哈希计算，得到确定长度的哈希值参与树的构建。而要求 ③ 则是利用帕特里夏树提高访问效率和优化存储空间。所以以太坊里面，引入默克尔树和帕特里夏树，把这种结构定义为默克尔-帕特里夏树，这两种树都是现成的，但是以太坊提出了结合，形成了一种新的树结构 MPT。

3.2 节点分类

MPT 树中的节点包括空节点、叶子节点、扩展节点和分支节点:

空节点: 简单的表示空字符串，在代码中是一个空串。

叶子节点: 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: 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。

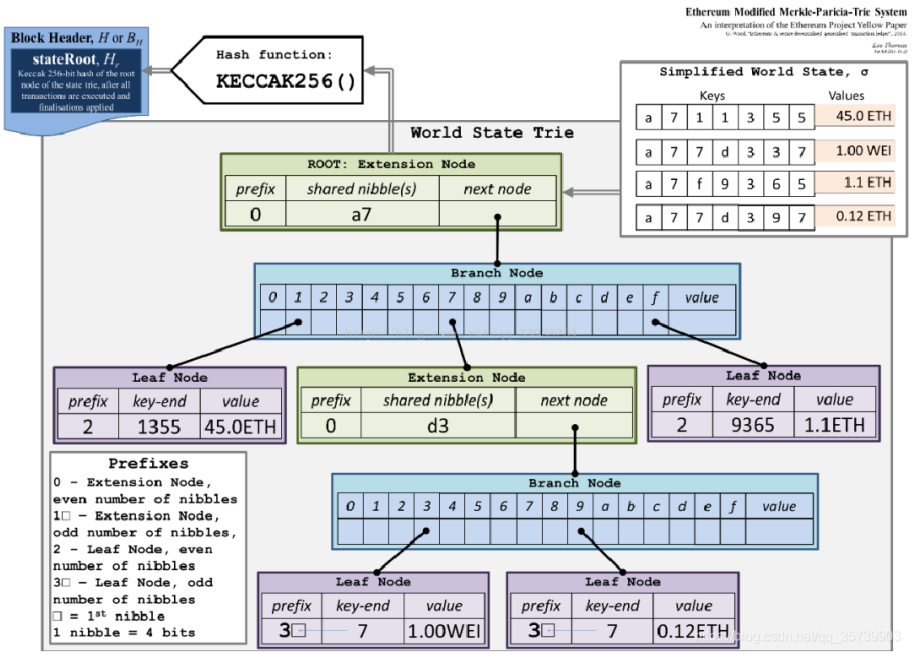
分支节点: 因为 MPT 树中的 key 被编码成一种特殊的 16 进制的表示，再加上最后的 value，所以分支节点是一个长度为 17 的 list，前 16 个元素对应着 key 中的 16 个可能的十六进制字符，如果有一个 [key,value] 对在这个分支节点终止，最后一个元素代表一个值，即分支节点既可以搜索路径的终止也可以是路径的中间节点。

3.3 HP 编码

MPT 还有一个重要的概念是特殊的十六进制前缀编码，十六进制序列的带可选结束标记的压缩编码。传统的编码十六进制字符串的方式，是将他们转为了十进制。比如 0f1248 表示的是三个字节的 [15，18，72]。然而，这个方式有点小小的问题，如果 16 进制的字符长度为奇数呢。在这种情况下，就没有办法知道如何将十六进制字符对转为十进制了。额外的，MPT 需要一个额外的特性，十六进制字符串，在结束节点上，可以有一个特殊的结束标记（一般用 T 表示）。结束标记仅在最后出现，且只出现一次。或者说，并不存在一个结束标记，而是存在一个标记位，标记当前节点是否是一个

最终结点，存着我们要查找的值。如果不含结束标记，则是表明还需指向下一个节点继续查找。

为了解决上述的这些问题。我们强制使用最终字节流第一个半字节（半个字节，4 位，也叫 nibble），编码两个标记位。标记是否是结束标记和当前字节流的奇偶性（不算结束标记），分别存储在第一个半字节的低两位。如果数据是偶数长，我们引入一个 0 值的半字节，来保证最终是偶数长，由此可以使用字节来表示整个字节流。



如上图所示，这是一个 MPT。从上面开始看，最上面是根节点，他是一个扩展结点，那扩展节点有什么特点呢，首先存一个压缩路径，然后存一个指向下一个节点的 hash，把压缩路径的前缀单独领出来了，实际上是存储的时候是合在一起存的，他的前缀给的是 0，因为后面的压缩起来的路径是偶数，偶数还是扩展结点，前缀的二进制表示就是 0000，还要补 0000，但是这里显示的只是前

缀，没有显示补 0 的操作。

然后后面存一个 hash 指向下一个节点。下一个节点是一个分支节点，因为我们发现这个地方没法去压缩路径，因为他有不同的路径出现，所以就岔开了。分支节点里面 1 这个插槽对应的是一个叶子节点，前缀是 2，因为他后面压缩的路径是偶数并且是叶子节点，后面还有 value，所以我们这里存的从根节点到分支节点，再到叶子节点。表示了一个什么键值对存储呢，他的 key 就是这个路径，从前面压缩出来的路径 a7，然后往下走到 1，然后 1355，他的值是 45eth，所以要存储的键是:a711355，值是 45 这样一个键值对，在 mpt 中就是这样组织存储。

4 以太坊数据的存储结构

以太坊数据存储的数据结构如图所示，最上面是区块头，有个 stateroot，是状态数的树根，状态树是一个世界状态，他包含了所有账户的状态的集。其下面的树根据每个账户的地址 hash 作为 key，然后存储每个账户的数据。所以说如果访问一个账户，按照刚才的规则找路径，直到找到某一个叶子节点，他的叶子节点里面存的是什么呢，是账户 account 的内容，同时，codehash 只是一个 32 字节的 hash，他还对应到真正的 code 的存储空间。storageroot 是梅克尔帕特里夏树的树根，它对应的是底层数据库中合约数据的存储位置。

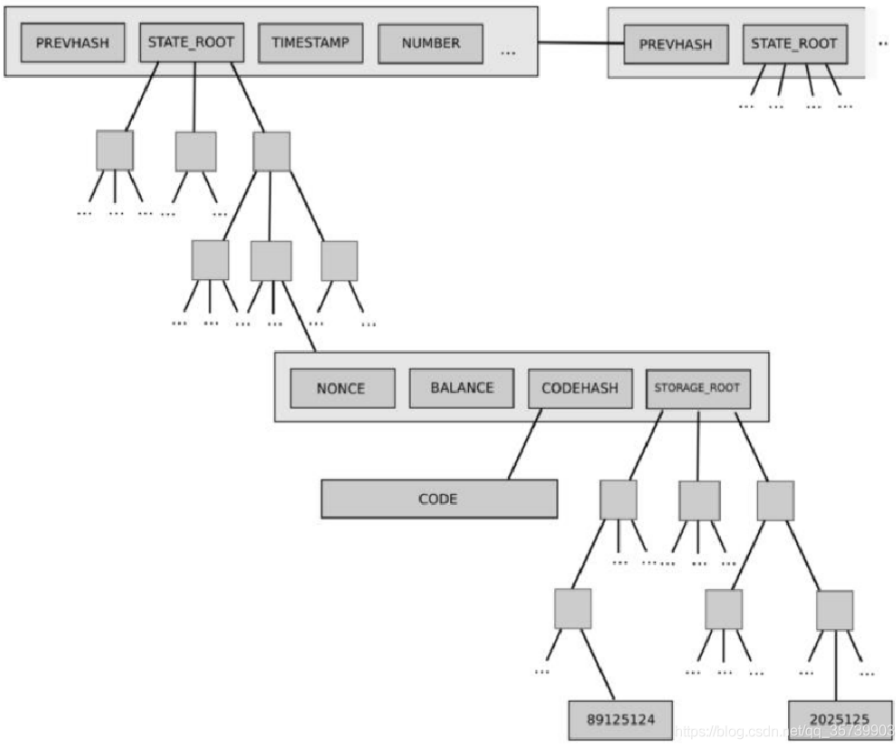


图 3: 以太坊中数据的存储结构

5 MPT 应用-轻节点扩展

MPT 能够提供的一个重要功能 - 默克尔证明，使用默克尔证明能够实现轻节点的扩展。

5.1 轻节点介绍

在以太坊或比特币中，一个参与共识的全节点通常会维护整个区块链的数据，每个区块中的区块头信息，所有的交易，回执信息等。由于区块链的不可篡改性，这将导致随着时间的增加，整个区块链的数据体量会非常庞大。运行在个人 PC 或者移动终端的可能性显得微乎其微。为了解决这个问题，一种轻量级的，只存储区块

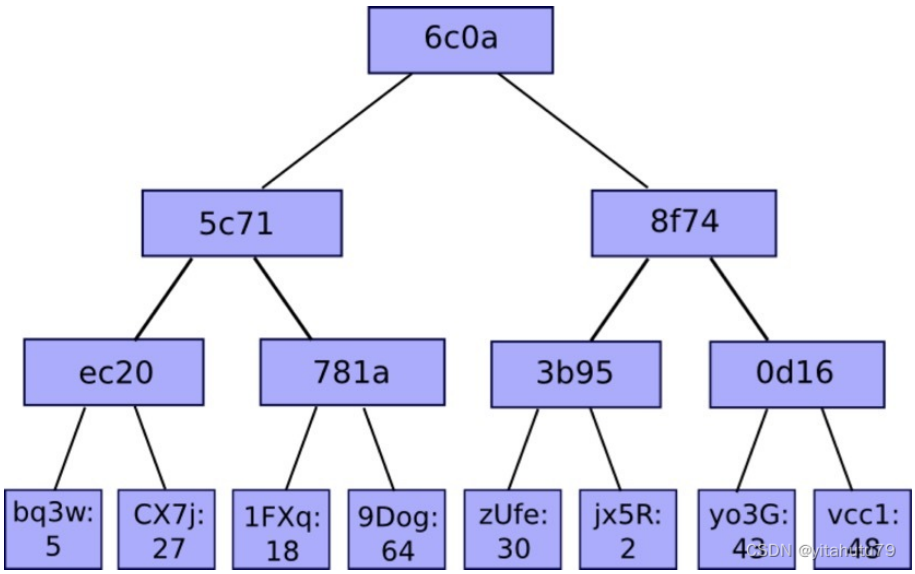
头部信息的节点被提出。这种节点只需要维护链中所有的区块头信息（一个区块头的大小通常为几十个字节，普通的移动终端设备完全能够承受出）。

在公链的环境下，仅仅通过本地所维护的区块头信息，轻节点就能够证明某一笔交易是否存在与区块链中；某一个账户是否存在与区块链中，其余额是多少等功能。

5.2 默克尔证明

默克尔证明指一个轻节点向一个全节点发起一次证明请求，询问全节点完整的默克尔树中，是否存在一个指定的节点；全节点向轻节点返回一个默克尔证明路径，由轻节点进行计算，验证存在性。

接下来举例说明默克尔证明的实现过程：



有如下图所示的 merkle 树，如果某个轻节点想要验证 9Dog:64 这个树节点是否存在与默克尔树中，只需要向全节点发送该请求，全节点会返回一个 1FXq:18, ec20,8f74 的一个路径。得到路径之后，

轻节点利用 9Dog:64 与 1FXq:18 求哈希，在与 ec20 求哈希，最后与 8f74 求哈希，得到的结果与本地维护的根哈希相比，是否相等，即可得出结果。

5.3 默克尔证明安全性

（1）若全节点返回的是一条恶意的路径？试图为一个不存在于区块链中的节点伪造一条合法的 merkle 路径，使得最终的计算结果与区块头中的默克尔根哈希相同。

由于哈希的计算具有不可预测性，使得一个恶意的“全”节点想要为一条不存在的节点伪造一条“伪路径”使得最终计算的根哈希与轻节点所维护的根哈希相同是不可能的。

（2）为什么不直接向全节点请求该节点是否存在于区块链中？

由于在公链的环境中，无法判断请求的全节点是否为恶意节点，因此直接向某一个或者多个全节点请求得到的结果是无法得到保证的。但是轻节点本地维护的区块头信息，是经过工作量证明验证的，也就是经过共识一定正确的，若利用全节点提供的默克尔路径，与代验证的节点进行哈希计算，若最终结果与本地维护的区块头中根哈希一致，则能够证明该节点一定存在于默克尔树中。

6 总结

MPT 可以用来存储内容为任何长度的 key-value 数据项。倘若数据项的 key 长度没有限制时，当树中维护的数据量较大时，仍然会造成整棵树的深度变得越来越深，会造成以下影响：

(1) 查询一个节点可能会需要许多次 IO 读取，效率低下；

(2) 系统易遭受 Dos 攻击，攻击者可以通过在合约中存储特

定的数据，“构造”一棵拥有一条很长路径的树，然后不断地调用 SLOAD 指令读取该树节点的内容，造成系统执行效率极度下降；

(3) 所有的 key 其实是一种明文的形式进行存储；

为了解决以上问题，在以太坊中对 MPT 再进行了一次封装，对数据项的 key 进行了一次哈希计算，因此最终作为参数传入到 MPT 接口的数据项其实是 (sha3(key), value). 其存在传入 MPT 接口的 key 是固定长度的 (32 字节)，可以避免出现树中出现长度很长的路径的优点，同样也存在每次树操作需要增加一次哈希计算以及需要在数据库中存储额外的 sha3(key) 与 key 之间的对应关系的缺点。