**作业2报告**

课程名称：区块链与数字货币

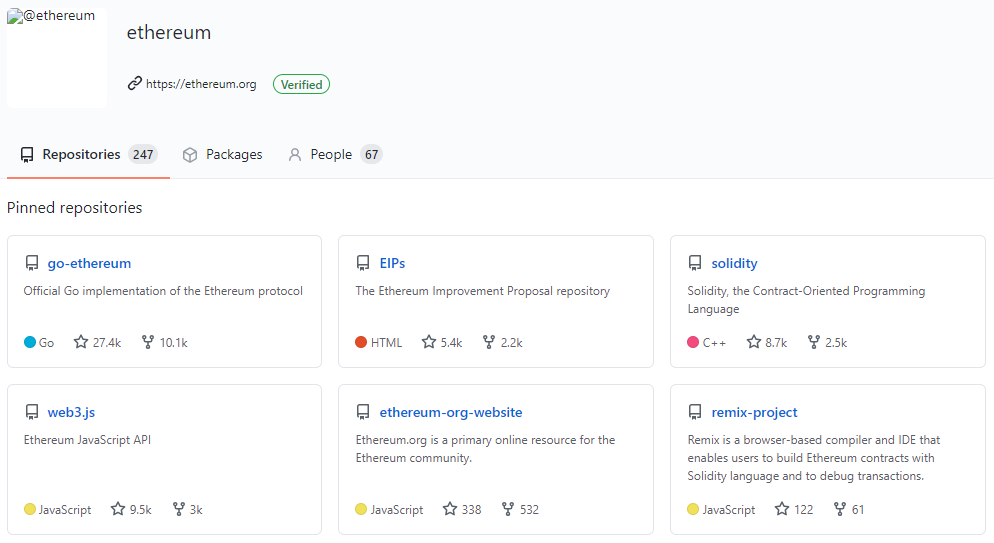
作业名称：分析以太坊源代码的Merkle Patricia Trie数据结构

学生姓名：臧一霄

学 号：3180103769

**查找以太坊源代码**

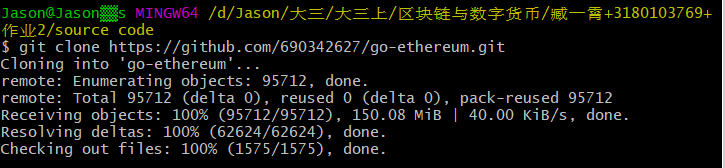
查阅资料找到以太坊（ethereum）在github上的GO语言源代码



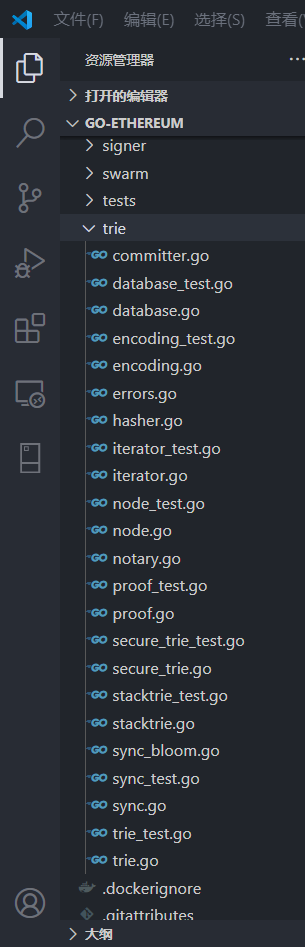
在Git Bash中通过命令

$ git clone https://github.com/690342627/go-ethereum.git

将以太坊源代码克隆到本地



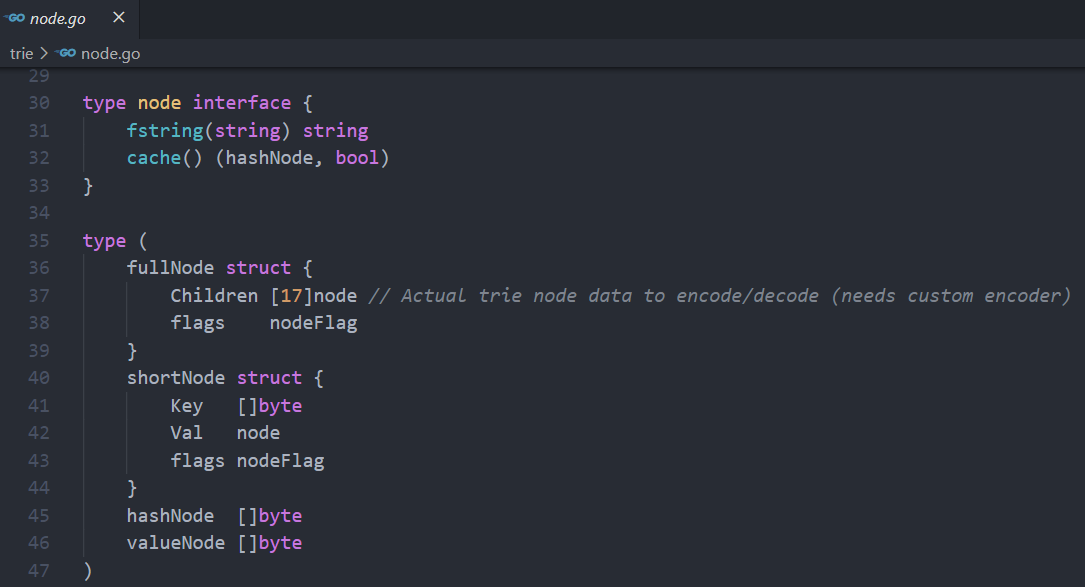
用vscode管理查看以太坊源代码



**分析源代码中的Merkle Patricia Trie**

查阅资料了解到，在以太坊的源码中，MPT树的源码主要在trie目录下，因此主要对trie目录下的源代码进行分析。通过阅读trie目录下的源代码，发现MPT树的数据结构定义主要在node.go和trie.go中。

**--node.go中**



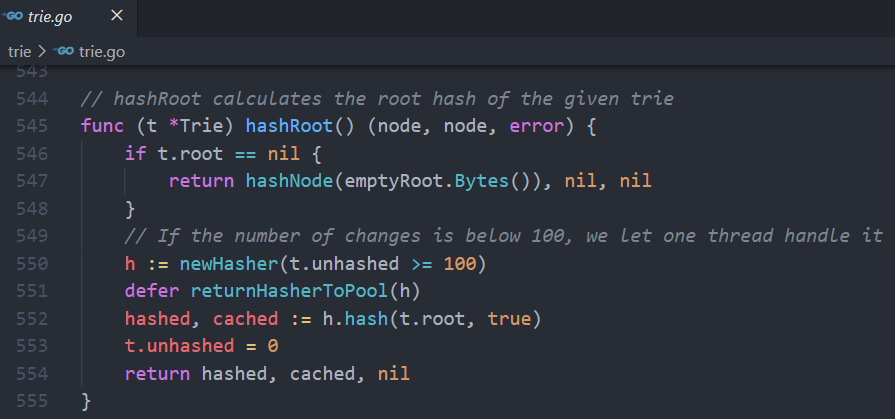
在上面的代码可以看到node是一个接口定义。包括: fullNode，shortNode，hashNode，valueNode，需要注意的是只有 fullNode 和 shortNode 能够包含子节点，因为fullnode和shortnode为结构性节点，而hashNode和valueNode为数据节点。

fullNode结构中定义了一个容量为 17 的 node 数组成员变量 Children，其中前 16 个数组成员分别对应 16 进制 (hex) 下的 0-9和a-f，通过HEX的KEY来存储对应的成员，用于子节点索引。并且定义了一个node标志flags，用来标识节点的一些特征。

shortNode结构中只定义了一个node节点，而成员 Key 是一个字节数组[]byte，成员变量 Val 指向一个子节点，成员flags是一个节点的标志变量。

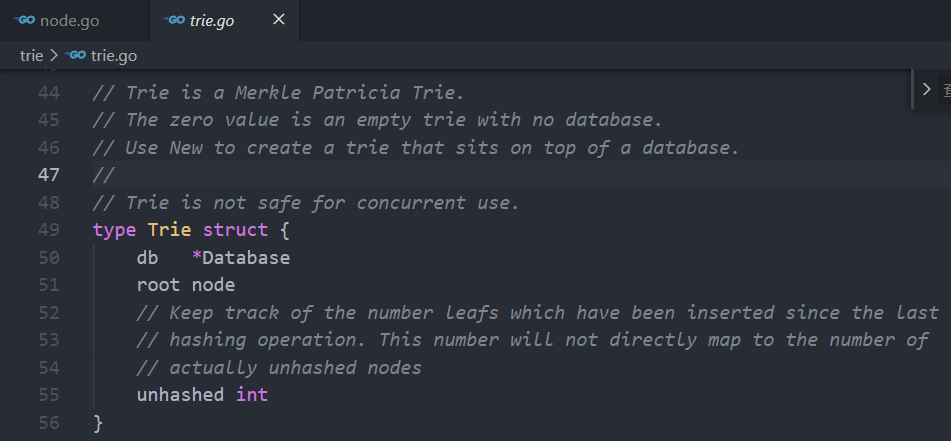
valueNode 是MPT数据结构中存储数据部分的节点。它的数据定义和hashNode一样就是一个比特数组，它存储的是数据的32字节长度RLP 哈希值，最终会被映射到数据库中。valuenode很简单，只是用于存储value。

hashNode 存储了fullNode 或者 shortNode对象的RLP哈希值，这是它与上面的valueNode不同之处。通过trie.go中的hashRoot返回代码可以知道，hashNode在 MPT 中很难单独存在，基本上在nodeFlag中以nodeFlag.hash为fullNode 和 shortNode间接持有。而在实际应用的场景中，这二者一旦改变，hashNode就一定会被更新。其实Block 的成员变量 Root、TxHash、ReceiptHash都与此有关。



这样，fullNode、shortNode、hashNode、valueNode这四个变量形成了一个完整的MPT树。

**--trie.go中**

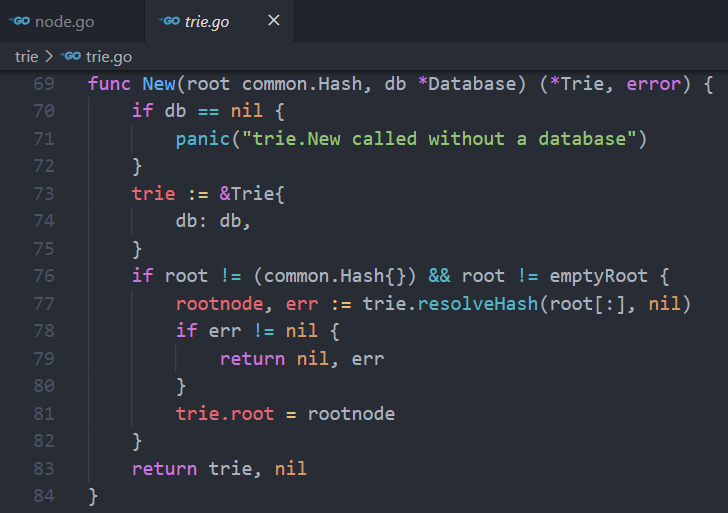


db是做为的KV存储的指针，在最终commit时，提交到leveldb中。

root是根节点。

unhashed用于跟踪自上次哈希操作以来插入的叶数。此数字不会直接映射到实际未剪切节点的数量。

New

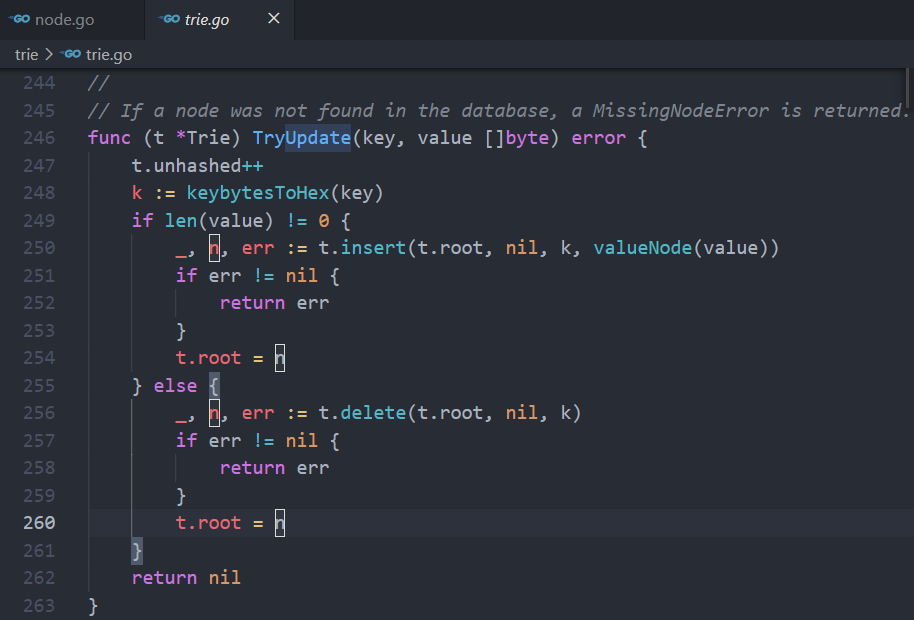


New(root common.Hash, db \*Database)创建一个新的Trie。这里的流程是这样的

73-75：创建一个结构体

76-82：当root为nil的时候，说明创建一个空的Trie。当root不为nil的时候，说明为加载一个已经存在的Trie。

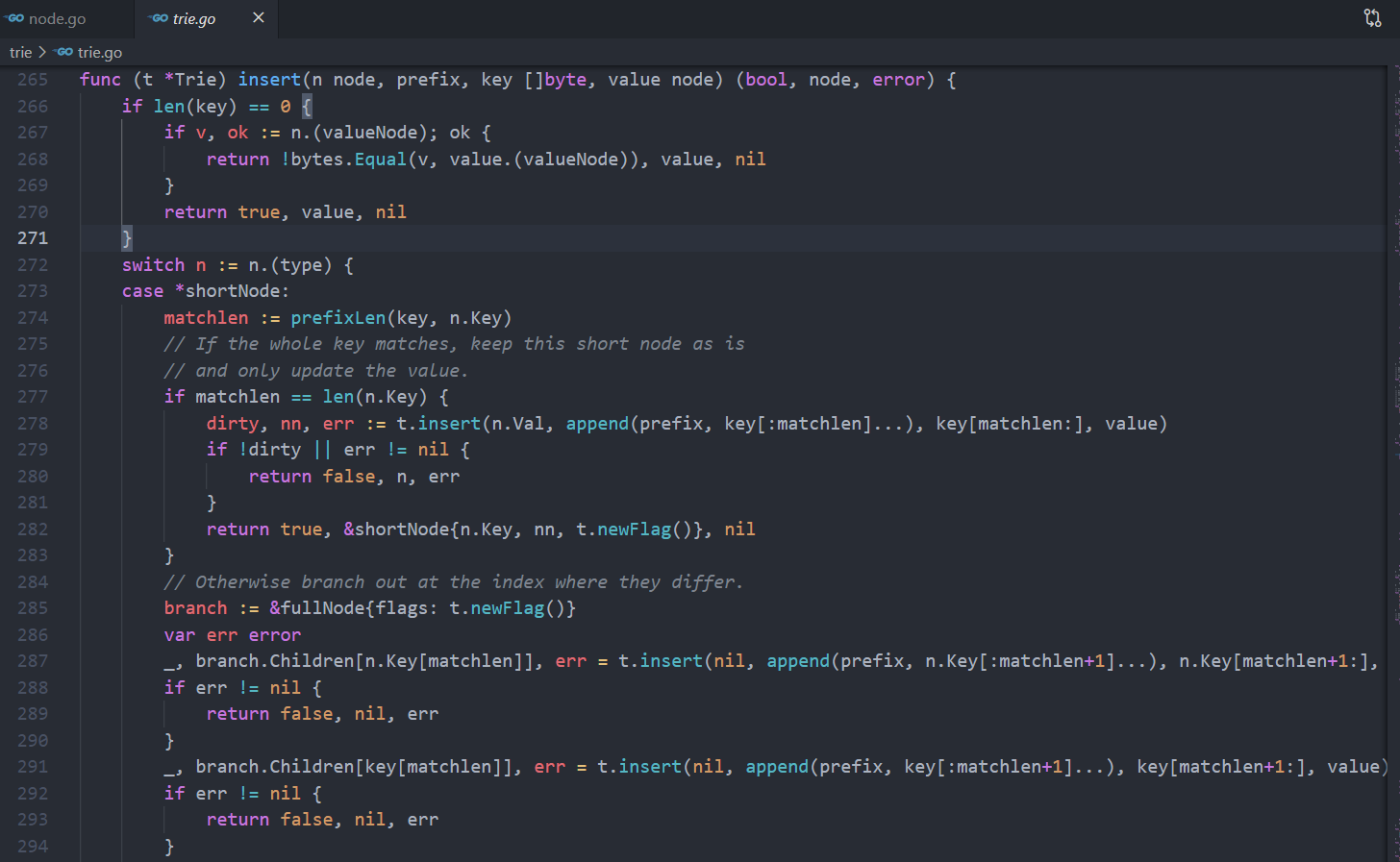
TryUpdate

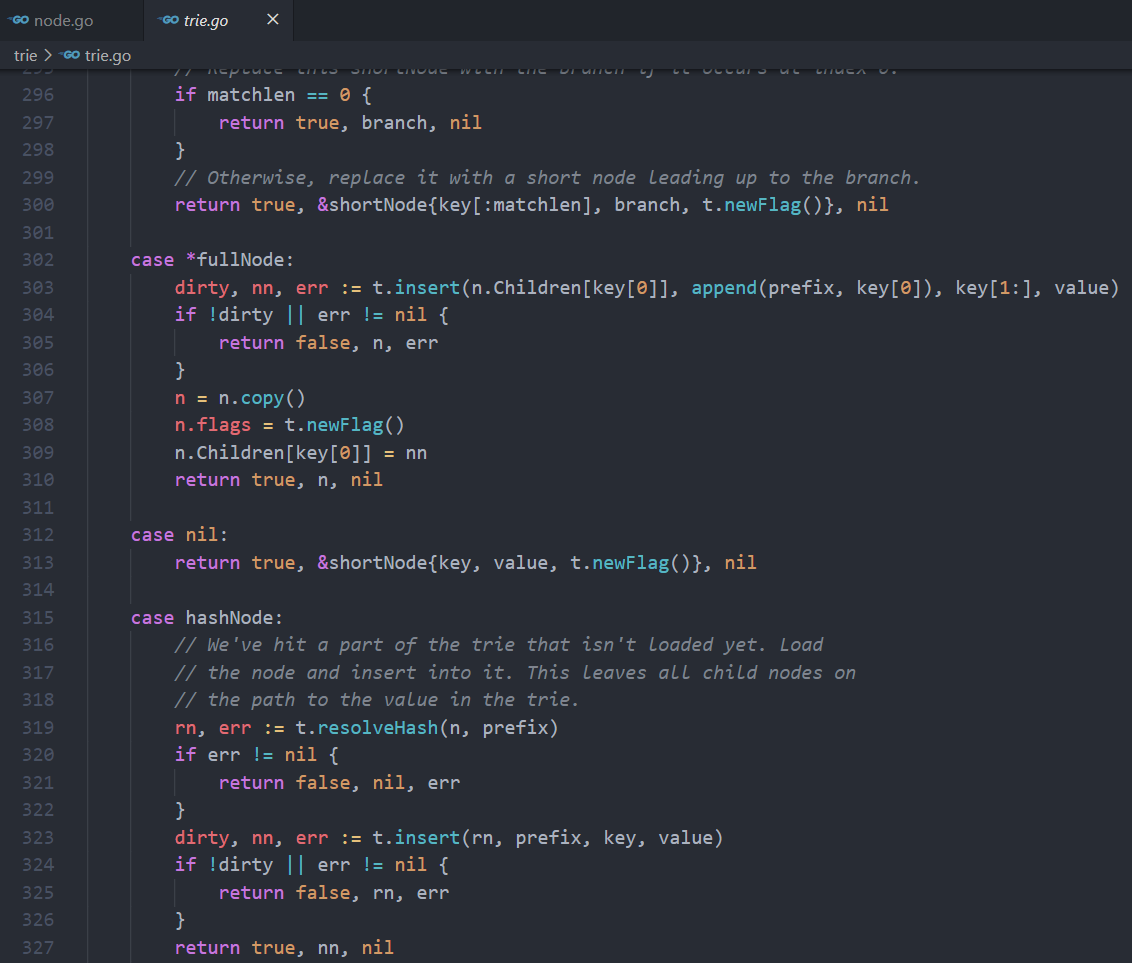


249-254：当value不为空的时候，说明是要刷新节点或者插入一个新的节点。创建一个valuenode，并调用insert。

255-261：当value为空的时候，说明是要将节点删除，调用了delete。

insert





274：变量n.key为插入节点node的node\_key（为了区分变量），变量key为要被插入节点的node\_key。计算其n.key和key的重合度。如过重合度相同则直接插入。

277-283：对key进行判断，如果key与要插入的node的key是一致的话，只需要构建一个新的shortnode，进行替换即可。

286-294：说明k和key的长度不一样，那么需要将shortnode拆分裂变为两个。pnode，nnode（这里面还有递归插入后续的动作）而当前的node则替换成fullnode，并将两个子节点set到这个fullnode下。

296-298：这里判断匹配长度是不是为0，是则，此次递归就完成了。

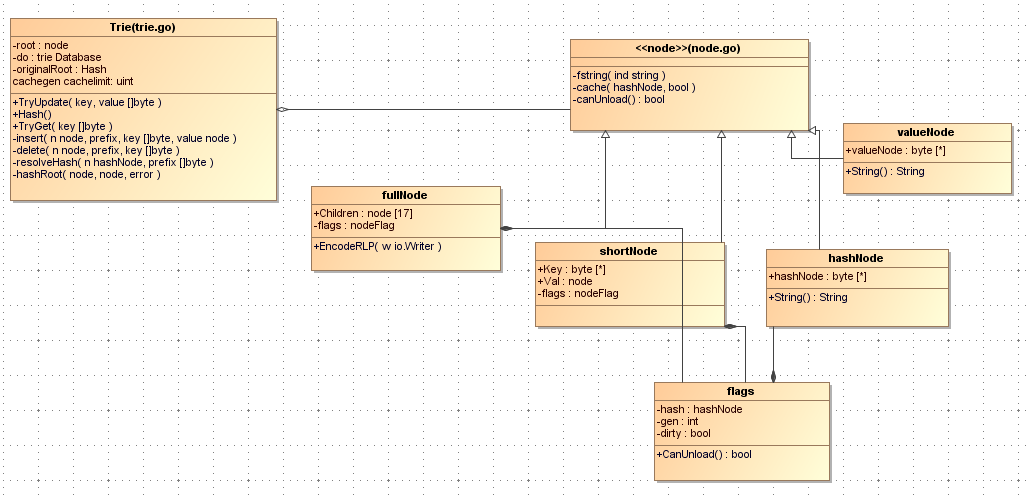
300：如果递归还没未完成，说明还有遗留的部分key是无法匹配的，那么就创建一个shortnode来包容剩下的。

302-310：当插入节点处为fullnode的时候，插入就比较简单。直接将其插入到fullnode中，但也需要继续递归插入了剩下的部分。

312-313：当node为nil的时候。当我们的trie为空的时候，root就是为nil。那么第一次插入的时候，走的就是这个流程。创建了shortnode。并将其返回了。在Update函数中，trie的root会被赋值为这个shortnode。那么第一次的插入就完成了。再继续插入的话，执行流程就走到这里了。第一次插入的root肯定是一个shortnode。

315-327：当node为hashNode时，说明hit到了一个还未load的trie，所以加载节点并插入其中，这会将所有子节点保留在trie中value的路径上。

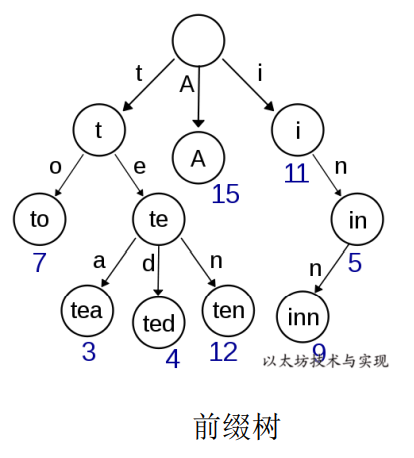
MRT相关数据结构图



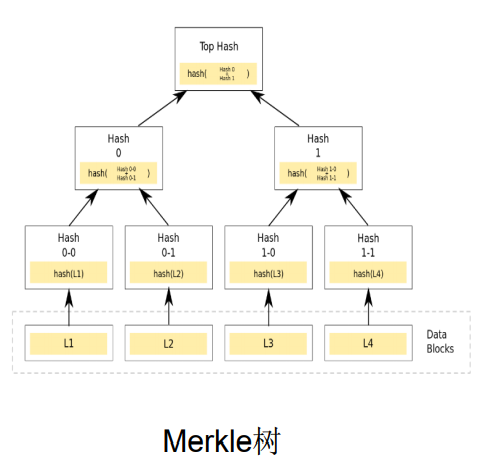
**MRT的作用**

为了说明Merkle Patricia Trie的作用，首先需要了解Trie树、Patricia Trie树、Merkle树。而这三个数据结构老师上课都介绍过了，因此在这里不再赘述。以下图片取自课件

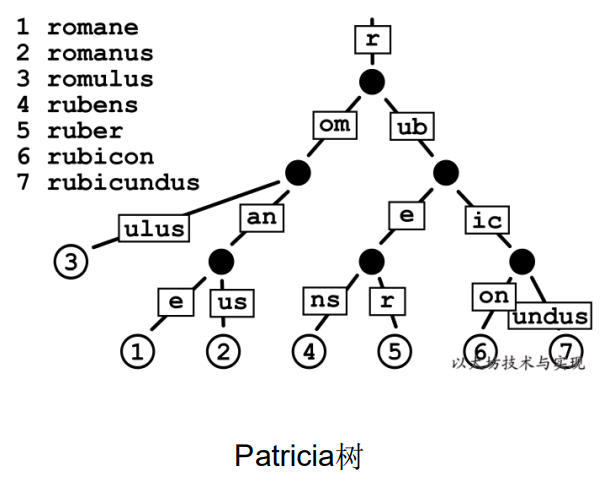
Trie树



Merkel树



Patricia树



在以太坊中，区块的区块头包含三颗MPT树，分别是：

* 交易树：记录交易的状态和变化。缓存到MTP
* 收据树(交易收据)：交易收据的存储
* 状态树(账户信息)：帐户中各种状态的保存。如余额等。

在相关分支中，其又增加了几条树，目的不外乎是存储更多的状态，不过，可能为了安全和更小的改动，新增的树存储在了块内。

通过查阅资料了解到比特币中为了SPV验证使用了默克尔树，但merkle树又有一定的信息量的不足，针对于此，以太坊将Trie树， Patricia Trie， 和Merkle树融合在一起，组成了一个新的数据结构Merkle Patricia Trie。Merkle Patricia Trie是以太坊中自定义的数据结构，综合了Merkle Tree与Patricia Trie两个的特点。