以太坊简介

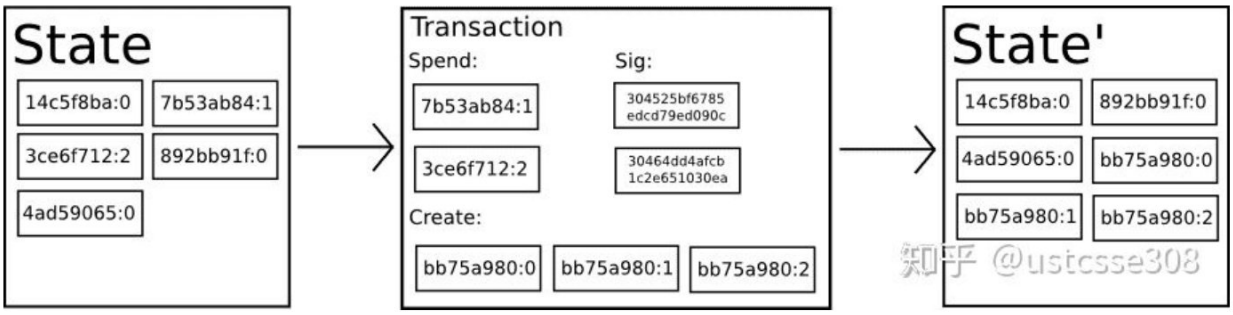
比特币系统的脚本语言存在的一些限制：

- 缺少图灵完备性
- 价值盲(value-blindness)
- 缺少状态
- 区块链盲(Blockchain-blindness)

以太坊的功能相对于比特币系统更加的强大，对比特币进行了改进。

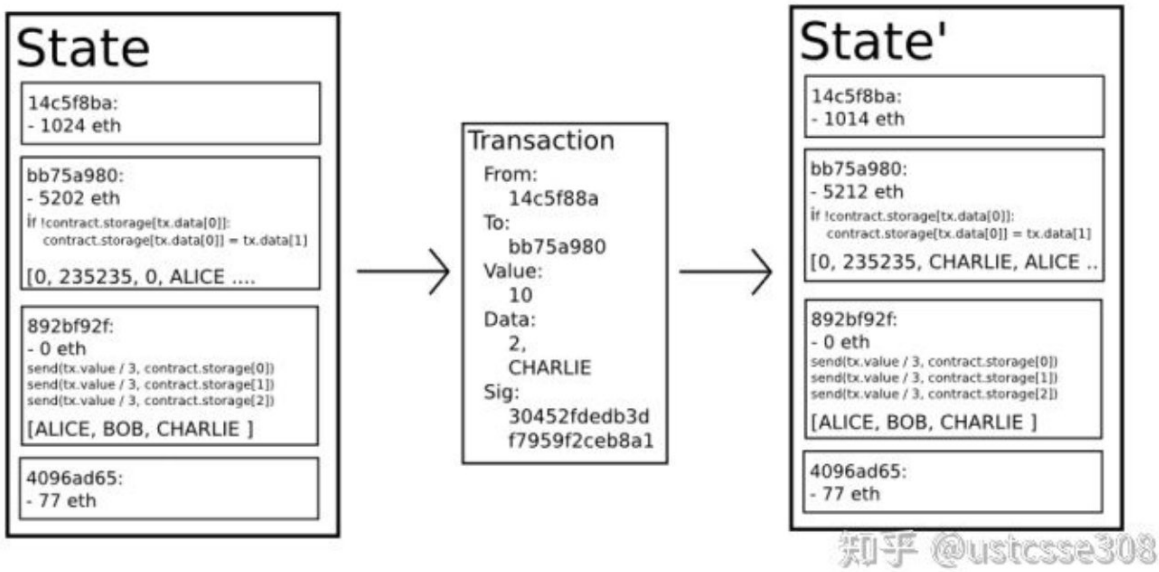
对比一下以太坊白皮书中列出的比特币和以太坊的状态转换图：

比特币作为状态转移系统：



在比特币的状态转移系统中，状态即UTXO，上面的图中新生成了三个state，删掉了两个state

以太坊的状态转换：



在上图中，起始状态保护了四个账户，分别是`14c5f8ba`、`bb75a980`、`892bf92f`、`4096ad65`。其中`14c5f8ba`和`4096ad65`仅维护了余额；`bb75a980`和`892bf92f`除了余额之外，还包括一部分的代码和数据。交易数据由`14c5f8ba`发送到`bb75a980`，转账10个以太币。同时发送了两个数据2和“charlie”。这个数据传递到账户bb75a980时，触发了bb75a980账户中的代码，代码大概意思是如果在contract.storage[tx.data[0]]的位置是0的话，就把这个地方的数据改成'charlie'

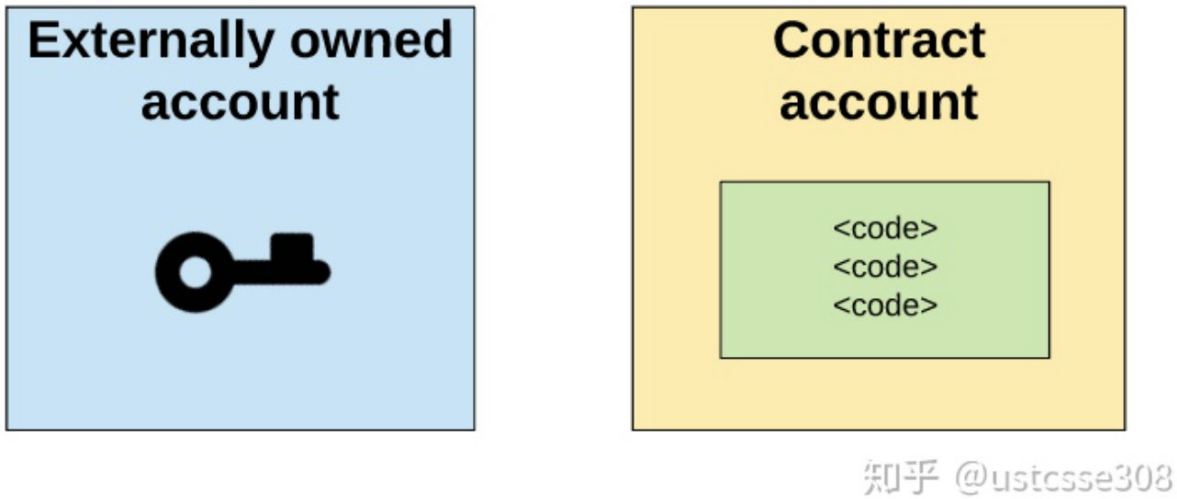
以太坊中重新引入了account(账户)这个概念

账户这一概念：

以太坊的全局“共享状态”是有很多对象（账户）来组成的，这些账户可以通过消息传递架构来与对方进行交互。每个账户都有一个与之关联的状态(state)和一个20字节的地址(address)。在以太坊中一个地址是160位的标识符，用来识别账户的。

有两种类型的账户：

1. 外部拥有的账户，被私钥控制且没有任何代码与之关联个简单来讲就是活人创建的账户
2. 合约账户，被它们的合约代码控制且有代码与之关联，可以理解为智能合约，这个账户可以由第1类账户创建，可以写一些代码创建，就比如说一个投票系统，只需要调用合约账户即可实现自动投票



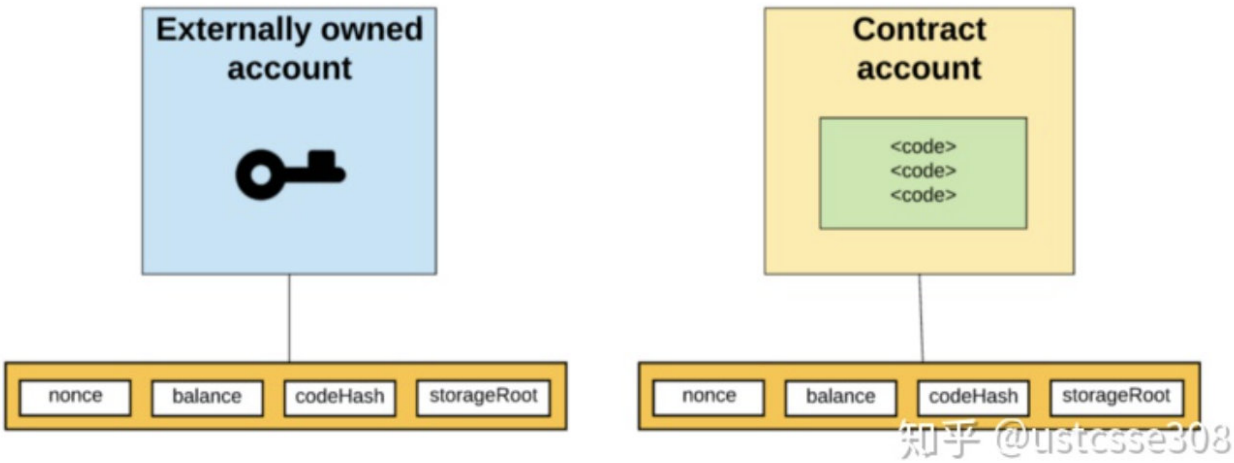
两种账户之间的比较

- 一个外部拥有的账户可以通过创建和用自己的私钥来对交易进行签名，来发送消息给另一个外部拥有账户或合约账户
- 在两个外部拥有账户之间传送的消息只是一个简单的价值转移。
- 但是从外部拥有的账户到合约账户的消息会激活合约账户的代码，允许它执行各种动作。（比如转移代币，写入内部存储，挖出一个新代币，执行一些运算，创建一个新的合约等等）。
- 不像外部拥有账户，合约账户不可以自己发起一个交易。
- 合约账户只有在接收到一个交易之后(从一个外部拥有账户或另一个合约账户接)，为了响应此交易而触发一个交易。我们将会在“交易和消息”部分来了解关于合约与合约之间的通信。

因此，**在以太坊上任何的动作，总是被外部控制账户触发的交易所发动的。**

账户状态(四部分组成)

1. nonce：如果账户是一个外部拥有账户，nonce代表从此账户地址发送的交易序号。如果账户是一个合约账户，nonce代表此账户创建的合约序号
2. balance(钱)： 此地址拥有Wei的数量。1 Ether=10^18 Wei
3. storageRoot： Merkle Patricia树的根节点Hash值。Merkle树会将此账户存储内容的Hash值进行编码，默认是空值
4. codeHash：此账户EVM（以太坊虚拟机，后面细说）代码的hash值。对于合约账户，就是被Hash的代码并作为codeHash保存。对于外部拥有账户，codeHash域是一个空字符串的Hash值



> 可以简单理解EOA就是实际的用户；而合约账户就是EOA用户部署的合约，外部所有的账户（EOA externally owned account）（由私钥控制的）和合约账户（由合约代码控制）。外部所有的账户没有代码，人们可以通过创建和签名一笔交易从一个外部账户发送消息。每当合约账户收到一条消息，合约内部的代码就会被激活，允许它对内部存储进行读取和写入，和发送其它消息或者创建合约。

图灵完备性

以太坊是图灵完备的，这就意味着在脚本里可以加入循环代码，但是对于矿工来讲可能比较危险，因为可能会陷入死循环，所以以太坊加入了一个机制来防止这种情况发生

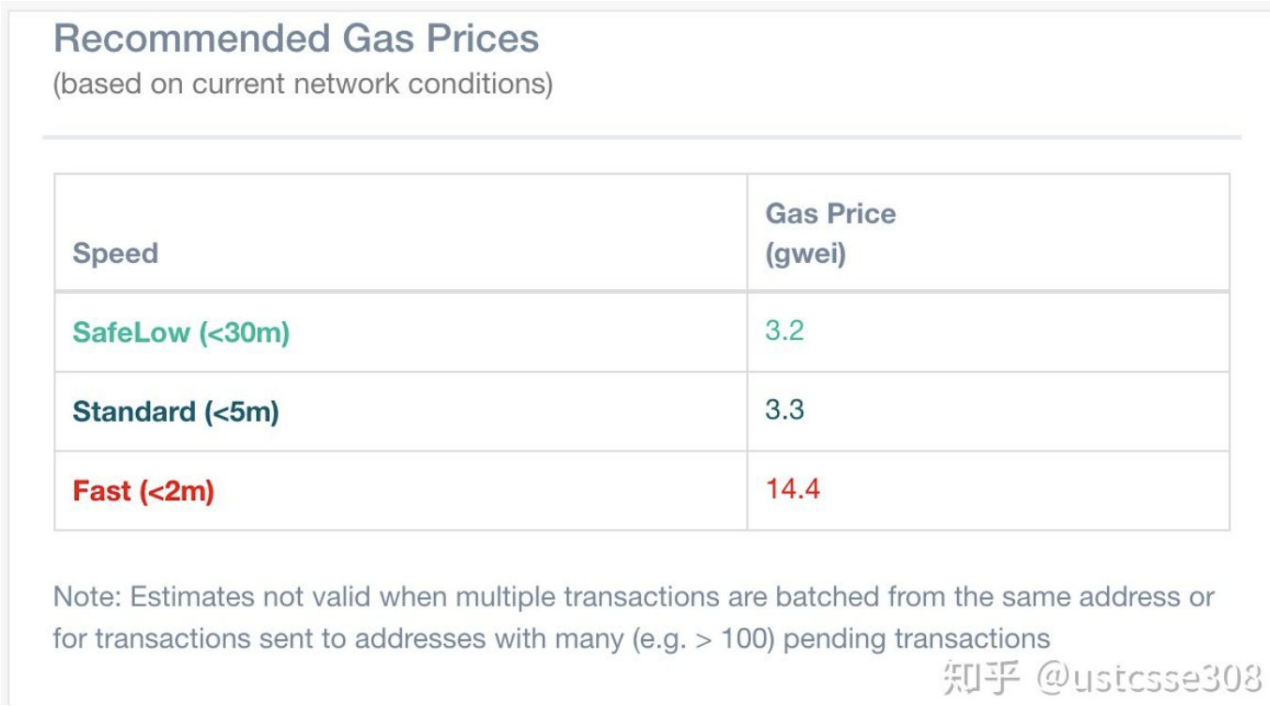
> 以太坊中引入了gas（瓦斯、油价等中文翻译）的概念。以太坊在区块链上实现了一个运行环境，被称为以太坊虚拟机（EVM），参与到网络的节点都会运行EVM，验证区块中的每个交易并在EVM中运行交易所触发的代码。合约可以利用的每个命令都会有一个相应的费用值，费用使用gas作为单位计数，也即用户付给矿工的佣金。这里列了一些命令的gas消耗。例：`PUSH操作`需要消耗`3个gas`，`一次转账`一般要消耗`21000 gas`，gas使用`ether`（以太币）来支付。

简单来讲，就是要花钱来请矿工进行计算，做的操作越复杂，花的钱越多，因此应该没人愿意写死循环

Gas常用的单位是wei，wei和ether的关系如下所示：

单位	wei值	Wei
wei	1	1 wei
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

每笔交易都被要求包括gas limit（startGas，限制）和Gas Price（价格）。Gas Limit 是用户愿意为执行某个操作或确认交易支付的最大Gas量（最少21,000）。Gas Price 是 Gwei 的数量，用户愿意花费于每个 Gas 单位的价钱。发送者支付的Gas Price越高，则其交易的优先级越重要，因为矿工打包该交易获得的报酬会更高，这样这个交易会较快地被打包到区块中，更早地获得确认。



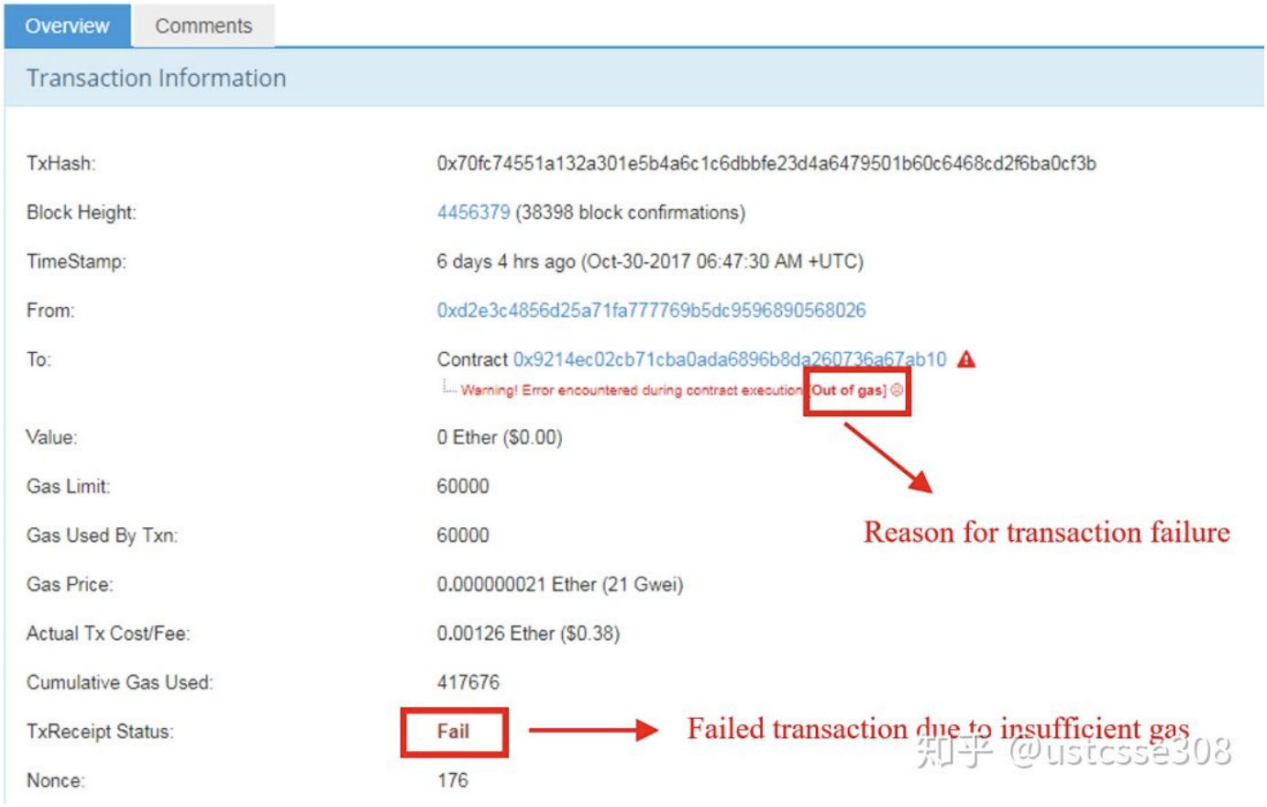
Recommended Gas Prices

(based on current network conditions)

Speed	Gas Price (gwei)
SafeLow (<30m)	3.2
Standard (<5m)	3.3
Fast (<2m)	14.4

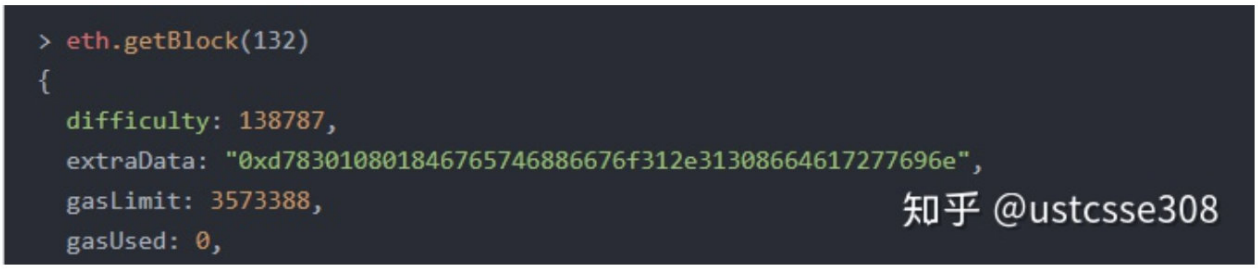
Note: Estimates not valid when multiple transactions are batched from the same address or for transactions sent to addresses with many (e.g. > 100) pending transactions

如果该交易需要使用的gas数量小于或等于所设置的gas limit，那么这个交易会被成功处理。如果gas总消耗超过gas limit，那么所有的操作都会被复原（回滚），但是交易费仍然会被矿工收取。区块链会显示这笔交易完成尝试，但因为没有提供足够的gas导致所有的合约命令都被复原。(**简单来讲就是钱没了但是事情还没有办成**)



如果交易里gas没有被消耗完毕，剩下的gas都会以以太币（ether）的形式打回给交易发起者

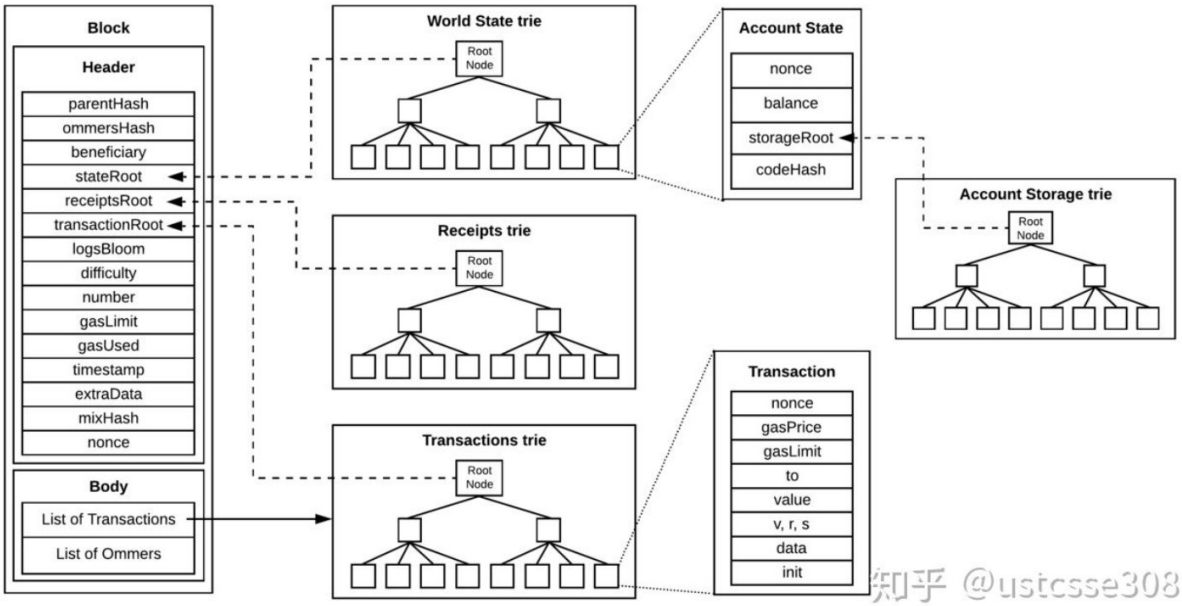
每个块还有Block gas limit，这个值在创世区块的配置文件中可以指定，譬如查看一个区块的信息如下，也即区块的gas limit是3573388。block gas limit的用意是限制一个区块中能够包含的交易的数量。



Value-blindness

为了实现账户方式，以太坊的做法是采用状态(state)的概念存储一系列账户，每个账户都有自己的余额，以及以太坊特有的数据（代码或内部存储器）。

为了维护这个状态，和比特币不同，比特币在区块中只包括了交易的merkle树根，而以太坊在区块头部中会包括三棵树，分别是世界状态树（world state trie），交易树（transaction trie）以及receipt树。



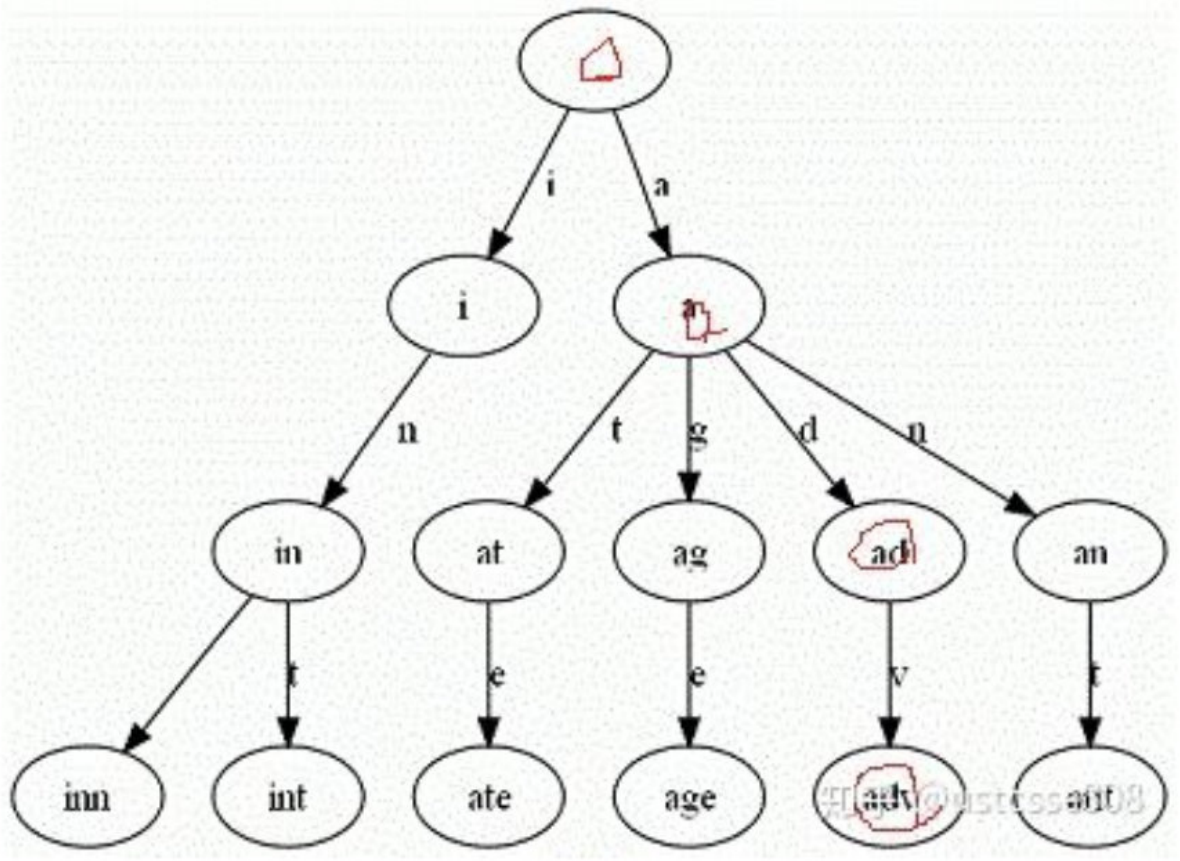
考试会考MPT树

帕特里夏树（Patricia Trees）（压缩前缀树）

Merkle Patricia Tree（又称为Merkle Patricia Trie）是一种经过改良的、融合了默克尔树和前缀树两种树结构优点的数据结构，是以太坊中用来组织管理账户数据、生成交易集合哈希的重要数据结构。

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。

下面这棵trie包含这样一组单词，inn, int, at, age, adv, ant 每个节点存储的是字符串中的部分字符，每个从根到某个节点的路径（不一定到叶子节点）代表了一个存储的字符串，如果想查找adv是否存在，只需要走红圈这样的路径即可。



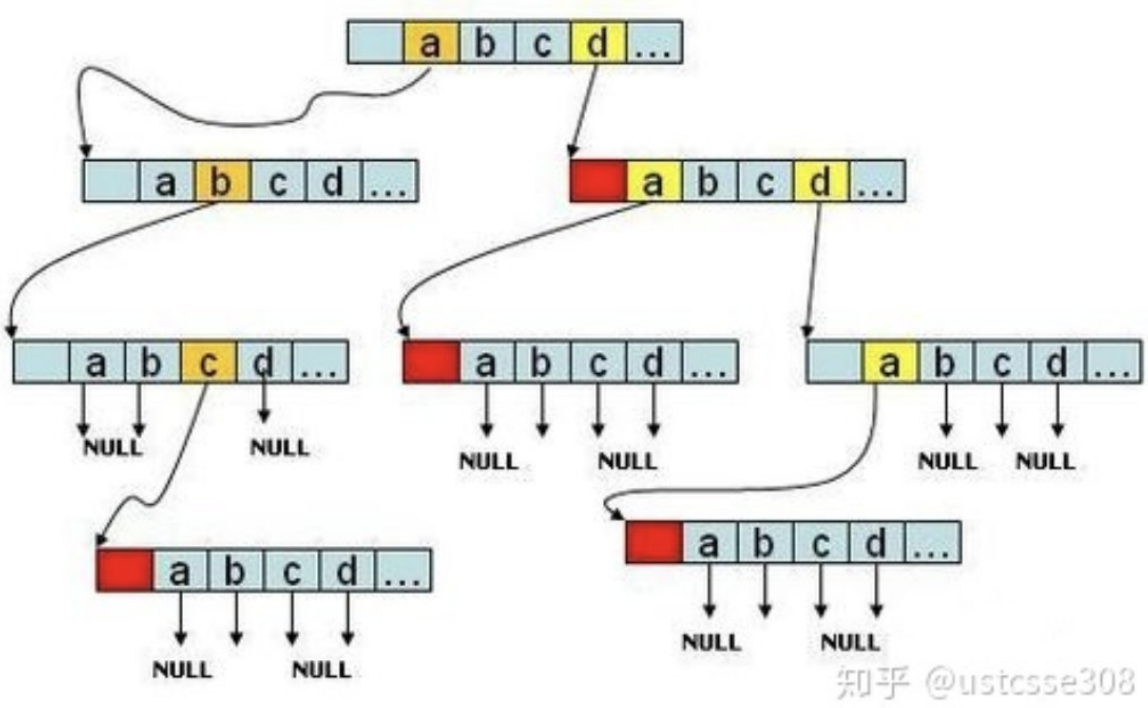
Trie的核心思想是**空间换时间**。利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。它有3个基本性质：

1. 根节点不包含字符，除根节点外每一个节点都只包含一个字符。
2. 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串
3. 每个节点的所有子节点包含的字符都不相同

典型应用是用于统计和排序大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：最大限度地减少无谓的字符串比较。举个具体的例子：

> 给你100000个长度不超过10的单词。对于每一个单词，我们要判断它出没出现过，如果出现了，求第一次出现在第几个位置。
> 分析：这题当然可以用hash来解决，但是比如说对于某一个单词，我们要询问它的前缀是否出现过。这样hash就不太适用，而用trie还是很简单。
> 现在回到例子中，如果我们用最朴实的方法，对于每一个单词，我们都要去查找它前面的单词中是否有它。那么这个算法的复杂度就是O(n^2)。显然对于100000的范围难以接受。现在我们换个思路想。假设我要查询的单词是abcd，那么在前面的单词中，以b，c，d，f之类开头的显然不必考虑。而只要找以a开头的中是否存在abcd就可以了。同样的，在以a开头中的单词中，我们只要考虑以b作为第二个字母的，一次次缩小范围和提高针对性，这样一个树的模型就渐渐清晰了。

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：



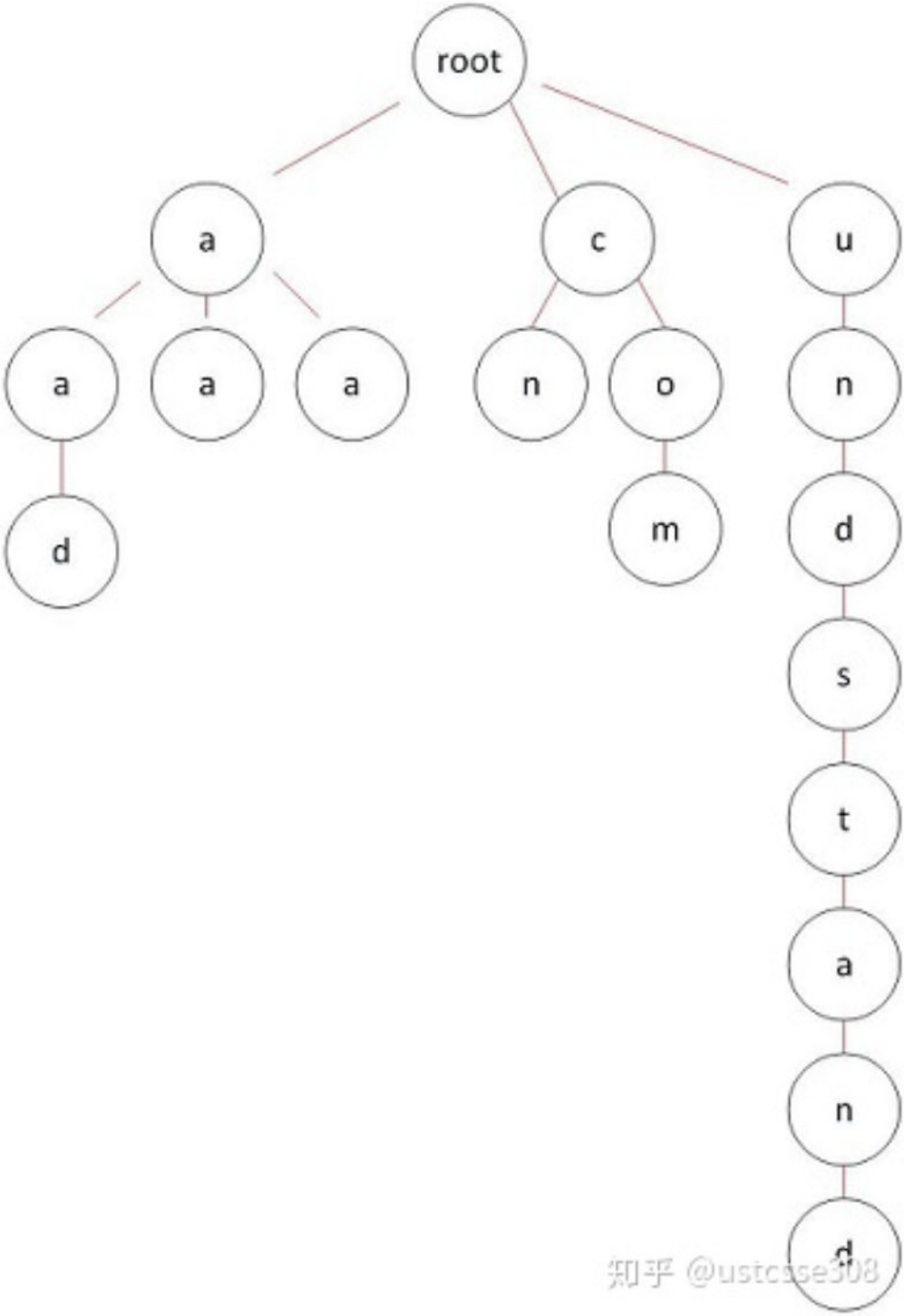
缺点：

- 直接查找效率较低

前缀树的查找效率是O(m)，m为所查找节点的key长度，而哈希表的查找效率为O(1)。且一次查找会有m次IO开销，相比于直接查找，无论是速率、还是对磁盘的压力都比较大。

- 可能会造成空间浪费

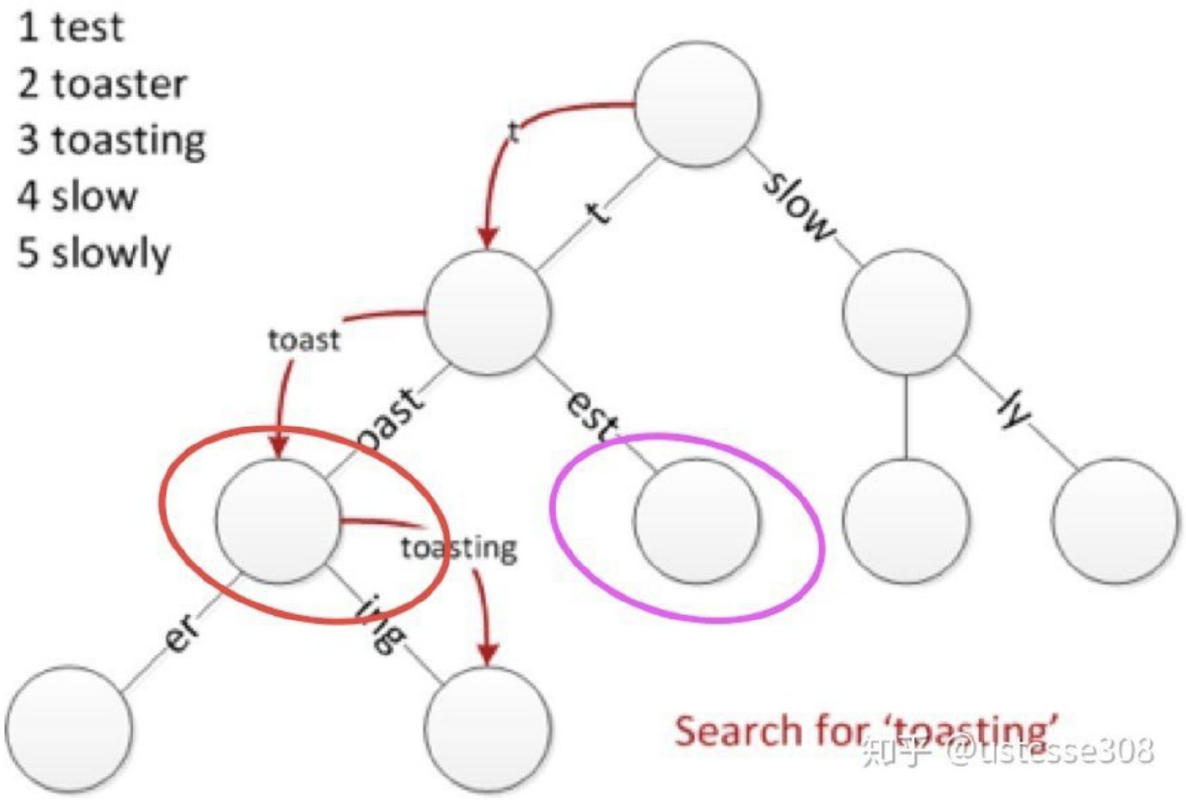
当存在一个节点，其key值内容很长（如一串很长的字符串），当树中没有与他相同前缀的分支时，为了存储该节点，需要创建许多非叶子节点来构建根节点到该节点间的路径，造成了存储空间的浪费。



对前缀树的改进：Patricia树

针对这种情况，MPT树对此进行了优化：当MPT试图插入一个节点，插入过程中发现目前没有与该节点Key拥有相同前缀的路径。此时MPT把*剩余的Key*存储在叶子 / 扩展节点的Key字段中，充当一个”Shortcut“。

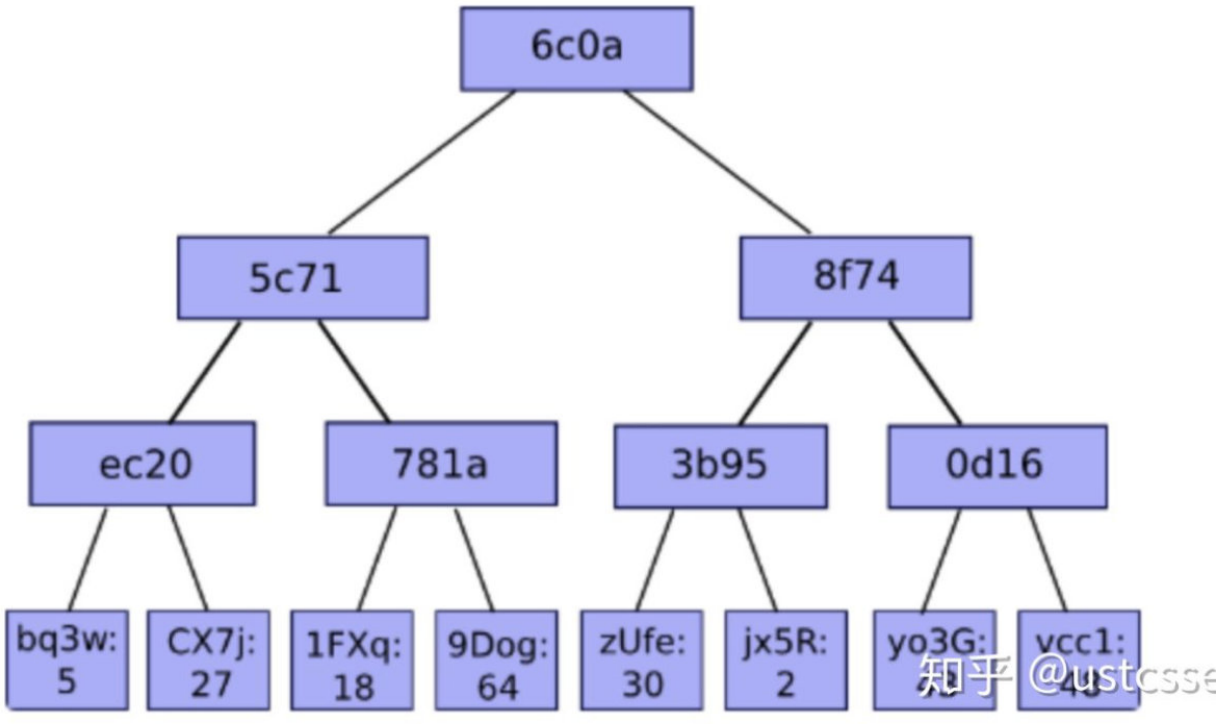
下图中对应着将5个单词test、toaster、toasting、slow、slowly插入到树中，其中有三个单词共享t前缀，两个单词共享s前缀，t和s不同，所以在根节点上有两个分支。在左边的分支（t）上，有两个分支，而且est是没有和其他任何单词共享前缀，也即父节点只有一个孩子，则est可以被压缩。类似地oast和slow以及ly可以进行压缩。



这样的做法有以下几点优势：

- 提高节点的查找效率，避免过多的磁盘访问；
- 减少存储空间浪费，避免存储无用的节点；

另一个例子，这种树可以用来存键值对，key就是路径上的字符串，value的值存储在叶子节点：

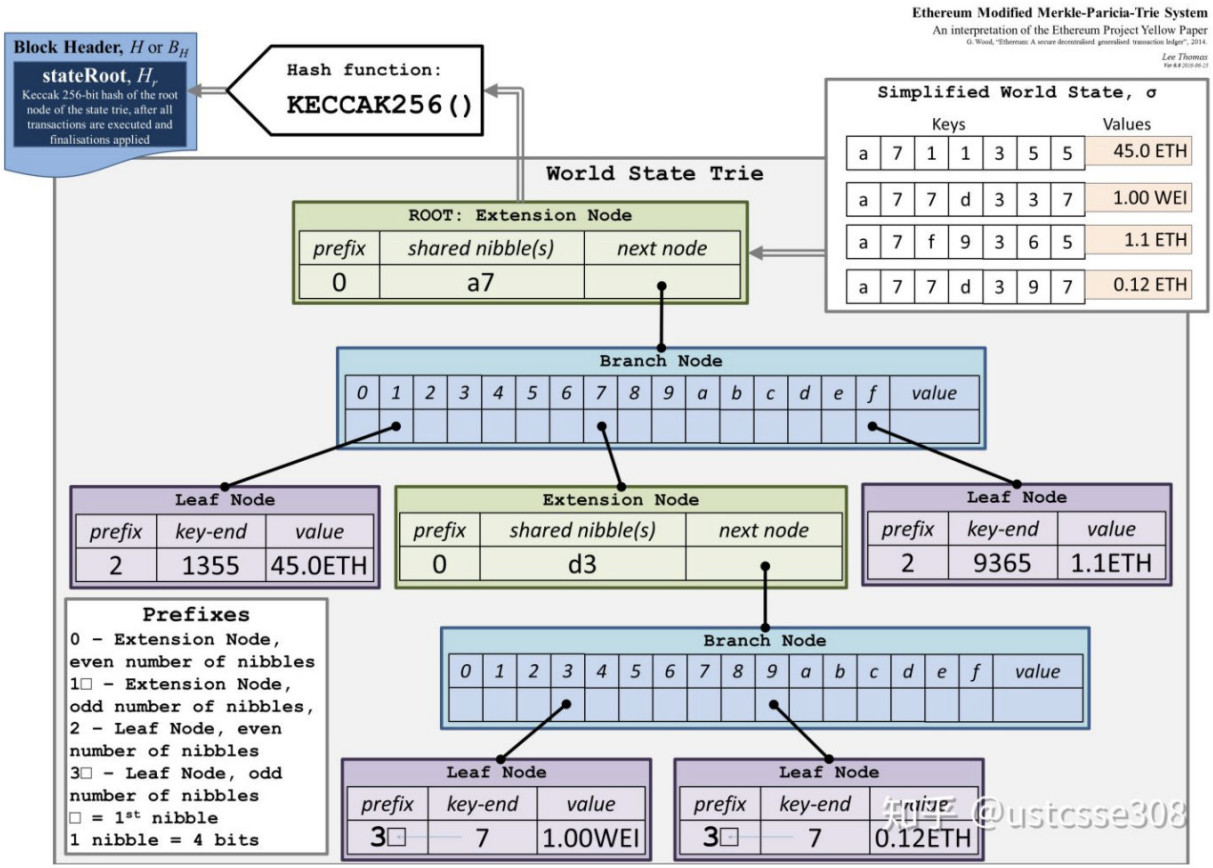


树中所存储的键值对：

- 6c0a5c71ec20bq3w => 5
- 6c0a5c71ec20CX7j => 27
- 6c0a5c71781a1FXq => 18
- 6c0a5c71781a9Dog => 64
- 6c0a8f743b95zUfe => 30
- 6c0a8f743b95jx5R => 2
- 6c0a8f740d16y03G => 43
- 6c0a8f740d16vcc1 => 48

就以太坊而言，状态树的键 / 值映射是地址与相关账户之间的映射，即指向每个账户的 balance、nonce、codeHash 和 storageRoot。

官方示例



与前缀树相同，MPT同样是把key-value数据项的key编码在树的路径中，但是key的每一个字节值的范围太大（[0-127]），因此在以太坊中，在进行树操作之前，首先会进行一个key编码的转换（下节会详述），将一个字节的高低四位内容分拆成两个字节存储。通过编码转换，`key'`的每一位的值范围都在[0, 15]内。因此，一个分支节点的孩子至多只有16个。以太坊通过这种方式，减小了每个分支节点的容量，但是增加了树高。

以太坊的MPT树种，树节点可以分为以下四类：

- 空节点：(represented as the empty string)

  简单的表示空，在代码中是一个空串。

- 分支节点：A 17-item node `[ v0 ... v15, vt ]`

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。

- 叶子节点：A 2-item node `[ encodedPath, value ]`

  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：A 2-item node `[ encodedPath, key ]`

  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。

另一个重要的概念：key值编码

在以太坊中，MPT树的key值共有三种不同的编码方式，以满足不同场景的不同需求：

1. Raw编码（原生的字符）
2. Hex编码（扩展的16进制编码）；
3. Hex-Prefix编码（16进制前缀编码）；

Raw编码

Raw编码就是原生的key值，不做任何改变。这种编码方式的key，是MPT对外提供接口的默认编码方式。

Hex编码

在介绍分支节点的时候，我们介绍了，为了减少分支节点孩子的个数，需要将key的编码进行转换，将原key的高低四位分拆成两个字节进行存储。这种转换后的key的编码方式，就是Hex编码。

从Raw编码向Hex编码的转换规则是：

- 将Raw编码的每个字符，根据高4位低4位(nibble)拆成两个字节；
- **若该Key对应的节点存储的是真实的数据项内容（即该节点是叶子节点），则在末位添加一个ASCII值为16的字符作为终止标志符**；
- 若该key对应的节点存储的是另外一个节点的哈希索引（即该节点是扩展节点），则不加任何字符；

Hex编码用于对内存中MPT树节点key进行编码

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。因为有两种[key,value]节点(叶节点和扩展节点)，所以需要对它们进行区分。此时，引进一种特殊的标识（一个bit即可）来标识key所对应的是值是叶子，还是其他节点的hash。如果标识符是1，那么key对应的是叶节点，反之则是扩展节点。

另外需要注意的一点是，在某个节点处，当前路径的长度可能是奇数。此时会面临的一个问题是，因为路径本身是按照4位，也即一个nibble为单位的，但是存储的时候总是以字节为单位的。假设当前有两个路径分别是‘136’和‘0136’，在存储的时候是没有办法区分的，因为以字节为单位进行存储的时候，总是会转化为01 + 36两个字节。所以，在HP编码中，还必须有一个标识进行路径长度奇偶性的标识。【注：要结合下面具体的例子看，这里的路径不是从根到叶子的完整的路径（总是偶数），而是当前在叶子节点/扩展节点中存储的路径的长度】

所以在MPT树中，对每个路径（叶子节点首先移除末尾的16），总是要首先加上一个nibble，这个Nibble的最低位表示节点路径长度奇偶性，第二低位表示节点的性质。

如果key是偶数长度，那么因为又加了一个四个比特，所以需要加上另外一个值为0的nibble，使得整体长度为偶数。

HP编码的规则如下：

- 若原key的末尾字节的值为16（即该节点是叶子节点），去掉该字节；
- 在key之前增加一个半字节，其中最低位用来编码原本key长度的奇偶信息，key长度为奇数，则该位为1；低2位中编码一个特殊的终止标记符，若该节点为叶子节点，则该位为1；
- 若原本key的长度为偶数，则在key之前再增加一个值为0x0的**半字节**；
- 将原本key的内容作压缩，即将两个字符以高4位低4位进行划分，存储在一个字节中（Hex扩展的逆过程）；

> 若Hex编码为[6, 3, 6, 1, 7, 4, 10]，则HP编码的值为[20, 63, 61, 74]

一些练习

```
> [ 1, 2, 3, 4, 5, ...] 【扩展结点，路径长度为奇数，添加01（bit），也即1(nibble)】
'11 23 45'
> [ 0, 1, 2, 3, 4, 5, ...]【扩展结点，路径长度为偶数，添加00，也即0，然后补一个0000】
'00 01 23 45'
> [ 0, f, 1, c, b, 8, 10] 【叶子节点，长度为偶数（最后一个10也即16，补充位，需要移除），添加10，也即2，然后补0000，也即0】
'20 0f 1c b8'
> [ f, 1, c, b, 8, 10] 【叶子节点，长度为奇数，添加11，也即3】
'3f 1c b8'
```

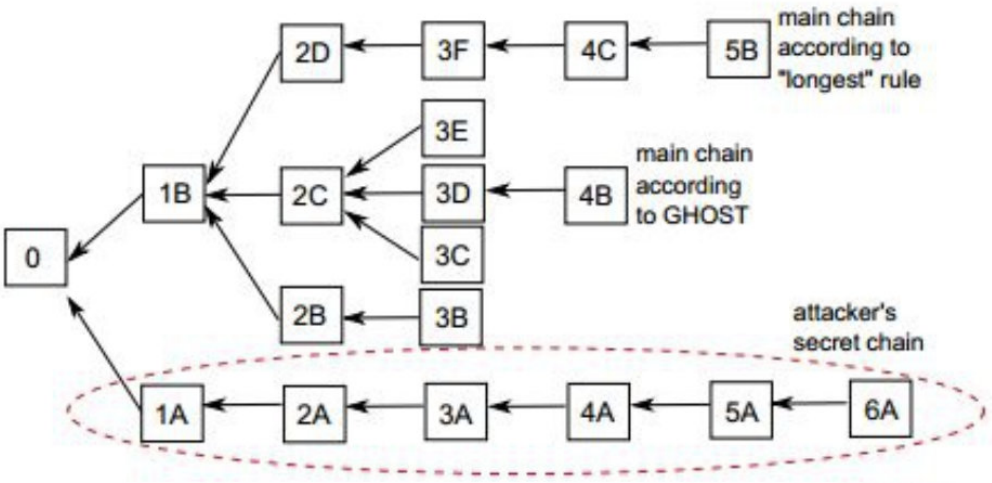
GHOST协议

GHOST（Greedy Heaviest Observed Subtree）**是一种主链（非侧链）选择协议**。举例来说：经典的Proof-of-Work（POW）是以取最长的主链为基本原则，GHOST协议则是以包含块数最多为基本原则。

以太坊认为孤块有价值

摒弃了单一的最长链原则, 取而代之的是最大子数原则；孤块奖励问题。

如下图所示



如果单纯的计算最长链原则, 主链应该是 0 -> 1B -> 2D -> 3F -> 3F -> 4C -> 5B.
如果采用了GHOST协议, 以前的"废块"也会被考虑到主链的计算量中.每一个节点下含一个子树, 兄弟节点之间子树节点最多的被选为主链. 这样一来 0 -> 1B -> 2C -> 3D -> 4B 成为主链,
如果采用GHOST协议, 一个攻击者仅仅提供一个1A到6A的长链并不能被认为是主链。

孤块奖励问题：

- 主链节点获得base reward；
- 一个节点最多引用两个叔块；
- 叔块必须是区块的前2层~前7层的祖先的直接子块；
- 被引用过的叔块不能重复引用；
- 引用叔块的区块，可以获得挖矿报酬的1/32，也就是5*1/32=0.15625 Ether，最多获得2*0.15625=0.3125 Ether；
- 主链节点的兄弟（非主链节点）获得 ((叔块高度+8-当前块的高度)/8)*base reward，如下图表格所示；
- 交易费用(transaction fee)不会分配给叔块