9区块链

9.1介绍

The blockchain data structure is an ordered, back-linked list of blocks of transactions. The blockchain can be stored as a flat file, or in a simple database. The Bitcoin Core client stores the blockchain metadata using Google's LevelDB database. Blocks are linked "back," each referring to the previous block in the chain. The blockchain is often visualized as a vertical stack, with blocks layered on top of each other and the first block serving as the foundation of the stack. The visualization of blocks stacked on top of each other results in the use of terms such as "height" to refer to the distance from the first block, and "top" or "tip" to refer to the most recently added block.
区块链数据结构是交易的区块的一个有序链表。
区块链可以存储为扁平文件，或存储在一个简单数据库中。
Bitcoin Core客户端使用Google LevelDB数据库存储区块链元数据。
每个区块链接到前一个区块。
区块链经常被看做一个垂直的栈，最下面的区块是栈的基础，每个区块在另一个区块的上面。
使用栈的“高度”表示某区块与首区块之间的距离，用“顶端”表示最新添加的区块。

Each block within the blockchain is identified by a hash, generated using the SHA256 cryptographic hash algorithm on the header of the block. Each block also references a previous block, known as the *parent* block, through the "previous block hash" field in the block header. In other words, each block contains the hash of its parent inside its own header. The sequence of hashes linking each block to its parent creates a chain going back all the way to the first block ever created, known as the *genesis block*.
区块链中的每个区块都用一个哈希识别，使用SHA256加密哈希算法对区块头进行计算，得到它的哈希。
每个区块还指向前一个区块（父区块），使用的是区块头中的“父区块哈希”字段。
也就是说，每个区块在头中包含父区块的哈希。
这样，哈希的序列将每个区块链接到它的父区块，从而创建一个链，一直能回到第一个区块（创世区块）。

Although a block has just one parent, it can temporarily have multiple children. Each of the children refers to the same block as its parent and contains the same (parent) hash in the "previous block hash" field. Multiple children arise during a blockchain "fork," a temporary situation that occurs when different blocks are discovered almost simultaneously by different miners (see [forks]). Eventually, only one child block becomes part of the blockchain and the "fork" is resolved. Even though a block may have more than one child, each block can have only one parent. This is because a block has one single "previous block hash" field referencing its single parent.
虽然每个区块只有一个父区块，但可以暂时拥有多个子区块。
每个子区块都将这个区块作为其父区块，并且在“父区块哈希”字段中有相同的哈希。
一个区块有多个子区块的情况被称为“区块链分叉”。
区块链分叉只是暂时状态，只有当多个不同区块几乎同时被不同的矿工发现时才会发生。
最终，只有一个子区块会成为区块链的一部分，也就解决了“区块链分叉”的问题。
尽管一个区块可能会有不止一个子区块，但每个区块只有一个父区块，这是因为一个区块只有一个“父区块希值”字段。

The "previous block hash" field is inside the block header and thereby affects the *current* block's hash. The child's own identity changes if the parent's identity changes. When the parent is modified in any way, the parent's hash changes. The parent's changed hash necessitates a change in the "previous block hash" pointer of the child. This in turn

causes the child's hash to change, which requires a change in the pointer of the grandchild, which in turn changes the grandchild, and so on. This cascade effect ensures that once a block has many generations following it, it cannot be changed without forcing a recalculation of all subsequent blocks. Because such a recalculation would require enormous computation (and therefore energy consumption), the existence of a long chain of blocks makes the blockchain's deep history immutable, which is a key feature of bitcoin's security.
由于区块头里面包含"父区块哈希"字段，所以当前区块的哈希也受到该字段的影响。
如果父区块的标识发生变化，子区块的标识也会跟着变化。
当父区块有任何改动时，父区块的哈希也发生变化。这将迫使子区块的"父区块哈希"字段发生改变，从而又将导致子区块的哈希发生改变。而子区块的哈希发生改变又将迫使孙区块的"父区块哈希"字段发生改变，又因此改变了孙区块哈希，以此类推。
一旦一个区块有很多后代以后，这种瀑布效应将保证该区块不会被改变，除非强制重新计算该区块的所有后续区块。
因为重新计算需要耗费巨大的计算量，所以，存在一个长区块链能让区块链的历史不可改变，这是比特币安全性的一个关键特征。

One way to think about the blockchain is like layers in a geological formation, or glacier core sample. The surface layers might change with the seasons, or even be blown away before they have time to settle. But once you go a few inches deep, geological layers become more and more stable. By the time you look a few hundred feet down, you are looking at a snapshot of the past that has remained undisturbed for millions of years. In the blockchain, the most recent few blocks might be revised if there is a chain recalculation due to a fork. The top six blocks are like a few inches of topsoil. But once you go more deeply into the blockchain, beyond six blocks, blocks are less and less likely to change. After 100 blocks back there is so much stability that the coinbase transaction—the transaction containing newly mined bitcoin—can be spent. A few thousand blocks back (a month) and the blockchain is settled history, for all practical purposes. While the protocol always allows a chain to be undone by a longer chain and while the possibility of any block being reversed always exists, the probability of such an event decreases as time passes until it becomes infinitesimal.
你可以把区块链想象成地质构造中的地质层或冰川岩芯样品。
表层可能会随着季节而变化，甚至在沉积之前就被风吹走了。
但是越往深处，地质层就变得越稳定。到了几百英尺深的地方，你看到的将是保存了数百万年但依然保持历史原状的岩层。
在区块链里，最近的几个区块可能会由于区块链分叉所引发的重新计算而被修改。
最新的六个区块就像几英寸深的表土层。但是，超过六个区块后，区块在区块链中的位置越深，被改变的可能性就越小。在100个区块以后，区块链已经足够稳定，这时可以花费币基交易。几千个区块（一个月）后的区块链将变成确定的历史，可用于所有实际的目的。
虽然协议总是允许一个链被更长的链替代，并且任何区块被修改的可能性都总是存在，但是，发生的概率会随时间的推移而降低，直到概率变成无穷小。

9.2 区块的结构

A block is a container data structure that aggregates transactions for inclusion in the public ledger, the blockchain. The block is made of a header, containing metadata, followed by a long list of transactions that make up the bulk of its size.
区块是一个容器数据结构，它聚合了一些交易，以包含在区块链中。
区块的构成：

- 区块头：包含元数据
- 交易列表：构成了区块的主体

The block header is 80 bytes, whereas the average transaction is at least 400 bytes and the average block contains more than 1900 transactions. A complete block, with all transactions, is therefore 10,000 times larger than the block header. The structure of a block describes the structure of a block.

区块头是80字节，每个交易平均是400字节，每个区块平均包含超过1900个交易。
因此，一个完整的区块是区块头的1000倍。
图1描述了区块的结构。

Table 1. The structure of a block
表1：区块的结构。

Size	Field	描述
4字节	区块大小	The size of the block, in bytes, following this field 区块的字节数，指这个字段之后的字节数
80字节	区块头	Several fields form the block header 几个字段形成了区块头
1~9字节（变长）	交易个数	How many transactions follow 后面跟了多少个交易
变长	交易	The transactions recorded in this block 记录在这个区块中的交易

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1–9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

9.3 区块头

The block header consists of three sets of block metadata. First, there is a reference to a previous block hash, which connects this block to the previous block in the blockchain. The second set of metadata, namely the *difficulty*, *timestamp*, and *nonce*, relate to the mining competition, as detailed in [mining]. The third piece of metadata is the merkle tree root, a data structure used to efficiently summarize all the transactions in the block. The structure of the block header describes the structure of a block header.
区块头由三组区块元数据组成。

- 父区块哈希
- merkle树根：用于有效地总结区块中所有交易
- 难度、时间戳、nonce：它们与挖矿竞争相关

Table 2. The structure of the block header
表2：区块头的数据结构

字节数	Field	描述
4	版本	A version number to track software/protocol upgrades 版本号，用于跟踪软件/协议升级
32	父区块哈希	A reference to the hash of the previous (parent) block in the chain

32	Merkle根	A hash of the root of the merkle tree of this block's transactions merkle数的根的哈希
4	时间戳	The approximate creation time of this block (seconds from Unix Epoch) 这个区块的近似创建时间（Unix时间）
4	难度目标	The Proof-of-Work algorithm difficulty target for this block 这个区块的工作量证明算法的难度目标
4	Nonce	A counter used for the Proof-of-Work algorithm 一个计数器，用于工作量证明算法

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

The nonce, difficulty target, and timestamp are used in the mining process and will be discussed in more detail in [mining].
Nonce、难度目标和时间戳用于挖矿过程，更多细节将在挖矿章节讨论。

9.4 区块标识：区块头哈希和区块高度

The primary identifier of a block is its cryptographic hash, a digital fingerprint, made by hashing the block header twice through the SHA256 algorithm. The resulting 32-byte hash is called the *block hash* but is more accurately the *block header hash*, because only the block header is used to compute it.区块的主要标识是它的加密哈希，这是一个数字指纹，算法是：
通过SHA256算法对区块头做两次哈希。
产生的32字节哈希被称为“区块哈希”，但更确切地说是“区块头哈希”，因为计算只用到了区块头。

For example, 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f is the block hash of the first bitcoin block ever created. The block hash identifies a block uniquely and unambiguously and can be independently derived by any node by simply hashing the block header.
例如：下面这个值是比特币的第一个区块哈希
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
区块哈希可以明确地标识一个区块，任何节点只用对区块头计算哈希都可以得出这个值。

Note that the block hash is not actually included inside the block's data structure, neither when the block is transmitted on the network, nor when it is stored on a node's persistence storage as part of the blockchain. Instead, the block's hash is computed by each node as the block is received from the network. The block hash might be stored in a separate database table as part of the block's metadata, to facilitate indexing and faster retrieval of blocks from disk.
注意，区块哈希实际上并不包含在区块的数据结构里，不会在网络上传输，也不存储在区块链中。
当每个节点收到一个区块时，会计算这个区块的哈希。

区块哈希可以存储在一个单独的数据库表中，作为这个区块的元数据的一部分，以便于索引和更快从磁盘获得区块。

A second way to identify a block is by its position in the blockchain, called the *block height*. The first block ever created is at block height 0 (zero) and is the same block that was previously referenced by the following block hash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f. A block can thus be identified in two ways: by referencing the block hash or by referencing the block height. Each subsequent block added "on top" of that first block is one position "higher" in the blockchain, like boxes stacked one on top of the other. The block height on January 1, 2017 was approximately 446,000, meaning there were 446,000 blocks stacked on top of the first block created in January 2009.
识别区块的另一个方法是“区块高度”，即区块在区块链中的位置。
第一个区块的高度为0，它的区块哈希就是前面说的那个值。
因此，区块可以用两种方式识别：区块哈希、区块高度。
2017-1-1的区块高度大约是 446,000，说明从2009年1月创建第一个区块以后，已经产生了446,000个区块。

Unlike the block hash, the block height is not a unique identifier. Although a single block will always have a specific and invariant block height, the reverse is not true—the block height does not always identify a single block. Two or more blocks might have the same block height, competing for the same position in the blockchain. This scenario is discussed in detail in the section [forks].
和区块哈希不同的是，区块高度并不是唯一标识。
虽然某个区块总是会有一个明确、固定的区块高度， 但一个区块高度并不总是对应一个区块。
多个区块可能有相同的区块高度，它们在区块链里争夺同一位置。这种情况就是“区块链分叉”。

The block height is also not a part of the block's data structure; it is not stored within the block. Each node dynamically identifies a block's position (height) in the blockchain when it is received from the bitcoin network. The block height might also be stored as metadata in an indexed database table for faster retrieval.
区块高度也不是区块数据结构的一部分，并不存储在区块里。
当节点接收来自比特币网络的区块时，会动态识别该区块在区块链里的位置（区块高度）。
区块高度也可作为元数据存储在一个索引数据库表中，以便于更快获取。

Tip: A block's *block hash* always identifies a single block uniquely. A block also always has a specific *block height*. However, it is not always the case that a specific block height can identify a single block. Rather, two or more blocks might compete for a single position in the blockchain.
提示：一个区块的区块哈希总能唯一地标识一个区块。一个区块也总是有一个特定的区块高度。
但是，一个特定的区块高度并不一定总能唯一地标识一个特定区块。
更确切地说，多个区块可能会竞争区块链中的一个位置。

9.5 创世区块

The first block in the blockchain is called the genesis block and was created in 2009. It is the common ancestor of all the blocks in the blockchain, meaning that if you start at any block and follow the chain backward in time, you will eventually arrive at the genesis block.
区块链中的第一个区块创建于2009年，被称为创世区块。
它是区块链中所有区块的共同祖先。

Every node always starts with a blockchain of at least one block because the genesis block is statically encoded within the bitcoin client software, such that it cannot be altered. Every node always "knows" the genesis block's hash and structure, the fixed time it was created, and even the single transaction within. Thus, every node has the starting point for the blockchain, a secure "root" from which to build a trusted blockchain.
每个节点都是从至少一个区块开始的，因为创世区块被写入到比特币客户端软件里，这样就不能改它。
每个节点总是知道创世区块的哈希、结构、创建时间，已经它包含的单个交易。
因此，每个节点都把该区块作为区块链的首区块，从而构建了一个安全、可信的区块链。

See the statically encoded genesis block inside the Bitcoin Core client, in *chainparams.cpp*.
在chainparams.cpp中，可以看到创世区块被写入到Bitcoin Core客户端中。

The following identifier hash belongs to the genesis block:
创世区块的哈希是：

```
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
```

You can search for that block hash in any block explorer website, such as *blockchain.info*, and you will find a page describing the contents of this block, with a URL containing that hash:
你可以在区块浏览网站搜索这个区块哈希，如blockchain.info，
你会找到描述这一区块内容的页面，该页面的URL包含了这个区块哈希：

https://blockchain.info/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Using the Bitcoin Core reference client on the command line:
在命令行上使用Bitcoin Core参考客户端：

```
$ bitcoin-cli getblock
000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

{
    "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
    "confirmations"          : 308321,
    "size"         : 285,
    "height"       : 0,
    "version"      : 1,
    "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
    "tx" : [
        "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"
    ],
    "time"         : 1231006505,
    "nonce"        : 2083236893,
    "bits"         : "1d00ffff",
    "difficulty"   : 1.00000000,
    "nextblockhash" :
"00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"
}
```

The genesis block contains a hidden message within it. The coinbase transaction input contains the text "The Times 03/Jan/2009 Chancellor on brink of second bailout for

banks." This message was intended to offer proof of the earliest date this block was created, by referencing the headline of the British newspaper *The Times*. It also serves as a tongue-in-cheek reminder of the importance of an independent monetary system, with bitcoin's launch occurring at the same time as an unprecedented worldwide monetary crisis. The message was embedded in the first block by Satoshi Nakamoto, bitcoin's creator.

创世区块包含了一个隐藏的信息。

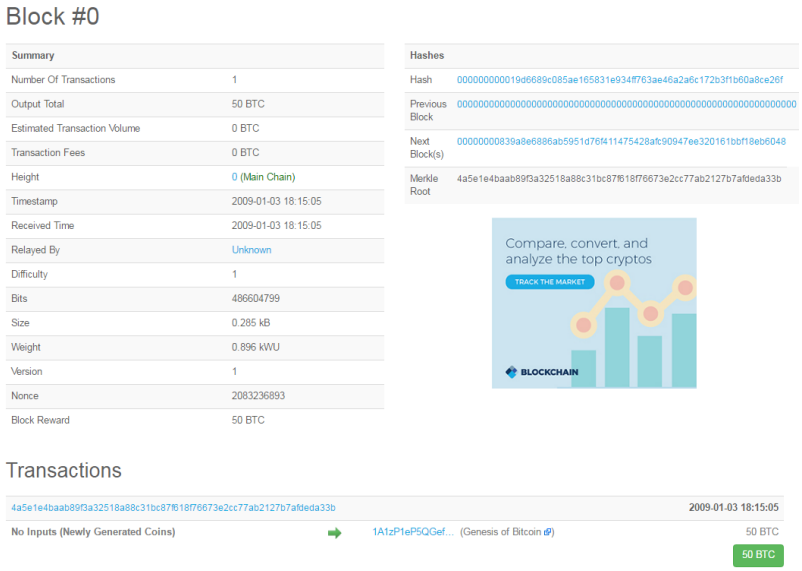
在币基交易输入中包含一句话：`The Times 03/Jan/2009 Chancellor on brink of second bailout for banks.`

这句话是泰晤士报当天的头版文章标题。

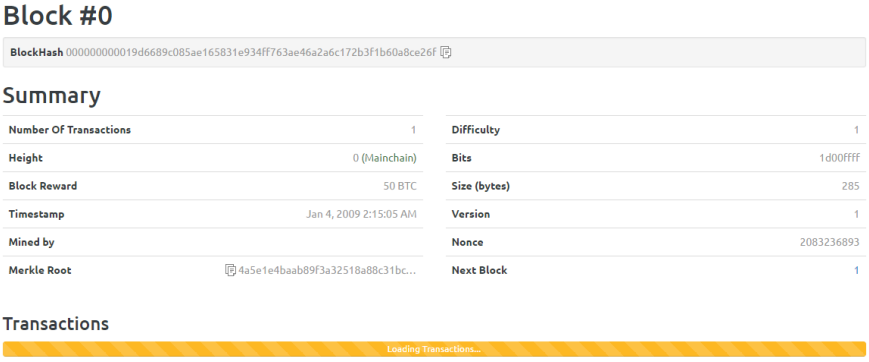
引用这句话，是对该区块产生最早时间的证明。

同时也提醒人们注意独立货币体系的重要性，比特币的推出与空前的全球货币危机同时发生。

比特币的创造者中本聪将这个消息写入到第一个区块。



https://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f



9.6 把“区块”链接到“区块链”

Bitcoin full nodes maintain a local copy of the blockchain, starting at the genesis block. The local copy of the blockchain is constantly updated as new blocks are found and used to extend the chain. As a node receives incoming blocks from the network, it will validate these blocks and then link them to the existing blockchain. To establish a link, a node will examine the incoming block header and look for the "previous block hash."

比特币全节点在本地维护一个区块链，从创世区块开始。

当新的区块被发现和用于扩展这个链时，本地区块链就不断更新。
当节点收到区块时，它会验证这些区块，然后链接到现有的区块链上。
为了建立链接，节点将检查区块头，寻找这个“父区块哈希”。

Let’s assume, for example, that a node has 277,314 blocks in the local copy of the blockchain. The last block the node knows about is block 277,314, with a block header hash of:
例如，一个节点的本地区块链有277,314个区块。
该节点知道最后一个区块是区块277,314，它的区块头哈希是：

```
0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249
```

The bitcoin node then receives a new block from the network, which it parses as follows:
之后，节点收到一个新的区块，该区块描述如下：

```
{
    "size"                      : 43560,
    "version"       : 2,
    "previousblockhash": "0000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
    "merkleroot"    : "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
    "time"                      : 1388185038,
    "difficulty"    : 1180923195.25802612,
    "nonce"         : 4215469401,
    "tx" : [
        "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
        #[... many more transactions omitted ...]
        "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"
    ]
}
```

Looking at this new block, the node finds the previousblockhash field, which contains the hash of its parent block. It is a hash known to the node, that of the last block on the chain at height 277,314. Therefore, this new block is a child of the last block on the chain and extends the existing blockchain. The node adds this new block to the end of the chain, making the blockchain longer with a new height of 277,315. Blocks linked in a chain by reference to the previous block header hash shows the chain of three blocks, linked by references in the previousblockhash field.
对于这个新区块，节点会在“previousblockhash”字段得到它的父区块的哈希。
这是节点已知的哈希，也就是区块 277314的区块哈希。
所以，这个新区块是这个链中的最后一个区块的子区块，因此这个区块链得到扩展。
节点将新区块添加至区块链的末端，使区块链的高度变为277,315。
图1显示了三个区块的链。

Figure 1. Blocks linked in a chain by reference to the previous block header hash
图1：通过引用父区块哈希把“区块”链接到一个“区块链”

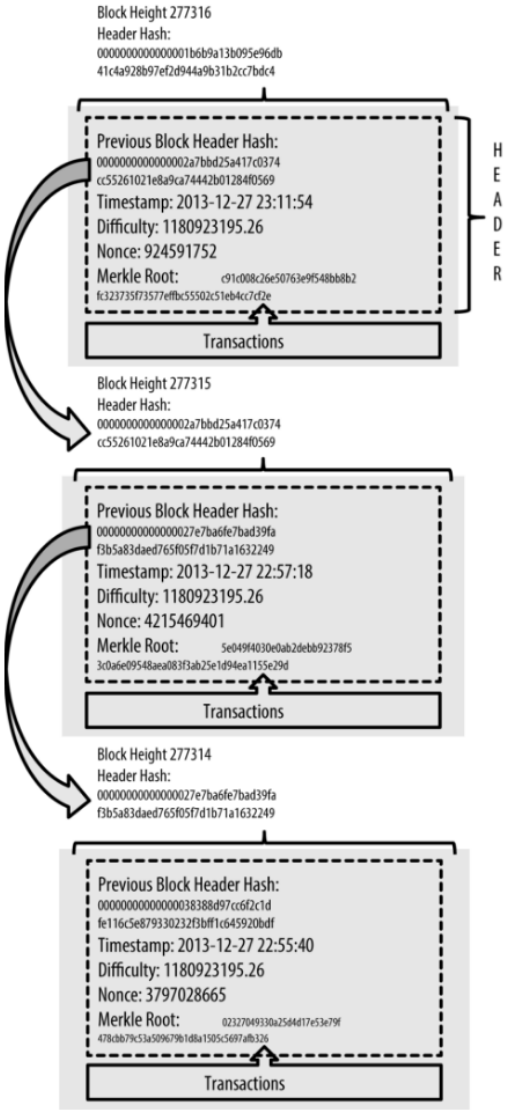


Figure 1. Blocks linked in a chain by reference to the previous block header hash

9.7 Merkle 树

Each block in the bitcoin blockchain contains a summary of all the transactions in the block using a *merkle tree*.
比特币区块链中的每个区块使用一个Merkle树来对这个区块中所有交易做一个总结，包含在区块中。

A *merkle tree*, also known as a *binary hash tree*, is a data structure used for efficiently summarizing and verifying the integrity of large sets of data. Merkle trees are binary trees containing cryptographic hashes. The term "tree" is used in computer science to describe a branching data structure, but these trees are usually displayed upside down with the "root" at the top and the "leaves" at the bottom of a diagram, as you will see in the examples that follow.
Merkle树是一种二叉哈希树，用于快速归纳和验证大规模数据集的完整性。
Merkle树是包含加密哈希的二叉树。
“树”在计算机学科中常用的一种数据结构。

Merkle trees are used in bitcoin to summarize all the transactions in a block, producing an overall digital fingerprint of the entire set of transactions, providing a very efficient process to verify whether a transaction is included in a block. A merkle tree is constructed by recursively hashing pairs of nodes until there is only one hash, called the *root*, or *merkle root*. The cryptographic hash algorithm used in bitcoin's merkle trees is SHA256 applied twice, also known as double-SHA256.
比特币中使用的Merkle树归纳了一个区块中的所有交易，对整个交易集生成了一个数字指纹，并且提供了一种高效的过程来验证一个交易是否包含在一个区块中。
Merkle数的构造方法是：递归地计算成对的节点的哈希，直到只剩下一个哈希，即merkle根。
比特币的Merkle树中使用的加密哈希算法是应用了两次SHA256，也称为“双SHA-256”。

When N data elements are hashed and summarized in a merkle tree, you can check to see if any one data element is included in the tree with at most 2*log~2~(N) calculations, making this a very efficient data structure.
当N个数据元素被哈希和归纳到一个merkle树中时，你就能检查任何一个数据元素是否包含在这个树中，只需要最多2*log2(N)个计算，所以这是一个非常高效的数据结构。

The merkle tree is constructed bottom-up. In the following example, we start with four transactions, A, B, C, and D, which form the *leaves* of the merkle tree, as shown in Calculating the nodes in a merkle tree. The transactions are not stored in the merkle tree; rather, their data is hashed and the resulting hash is stored in each leaf node as H_A, H_B, H_C, and H_D:
Merkle树是自底向上构建。
见图2，我们从四个交易（ABCD）开始，它们形成了merkle树的叶子。
这些交易不存在这个merkle树中，而是，它们的数据被哈希，生成的哈希存储在每个叶节点中（H_A, H_B, H_C, H_D）。

```
H_A = SHA256(SHA256(Transaction A))
```

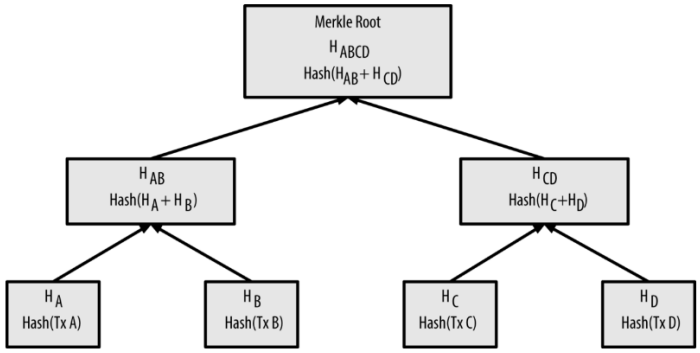


图9-2计算默克树中的节点

Figure 2. Calculating the nodes in a merkle tree
图2：计算merkle树中的节点

Consecutive pairs of leaf nodes are then summarized in a parent node, by concatenating the two hashes and hashing them together. For example, to construct the parent node H_{AB}, the two 32-byte hashes of the children are concatenated to create a 64-byte string. That string is then double-hashed to produce the parent node's hash:
然后，连续的节点对呗总结到一个父节点中，方法是：把这两个哈希连在一起，然后一起计算哈希。
例如，为了构建父节点H_{AB}，子节点 A和子节点B的两个32字节的哈希将被连接成64字节的字符串。
然后，对这个字符串进行两次哈希，产生了这个父节点的哈希：

$$H_{AB} = SHA256(SHA256(H_A + H_B))$$

The process continues until there is only one node at the top, the node known as the merkle root. That 32-byte hash is stored in the block header and summarizes all the data in all four transactions. Calculating the nodes in a merkle tree shows how the root is calculated by pair-wise hashes of the nodes.
继续这个过程，直到顶部只有一个节点，这个节点就称为Merkle根。
这个32字节哈希存储在区块头中，它归纳了四个交易的所有数据。
图2显示了如何计算Merkle根。

Because the merkle tree is a binary tree, it needs an even number of leaf nodes. If there is an odd number of transactions to summarize, the last transaction hash will be duplicated to create an even number of leaf nodes, also known as a *balanced tree*. This is shown in Duplicating one data element achieves an even number of data elements, where transaction C is duplicated.
因为Merkle树是二叉树，所以它需要偶数个叶节点。
如果只有奇数个交易需要归纳，则最后那个交易就被复制一份，以构成偶数个叶节点，这种偶数个叶节点的树也被称为平衡树。
如图3所示，C节点被复制了一份。

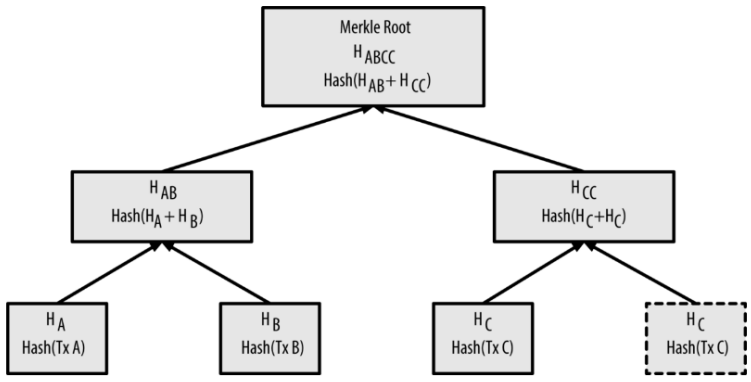


图9-3复制一个数据元素可以实现偶数个数据元素

Figure 3. Duplicating one data element achieves an even number of data elements
图3：复制一个数据元素，以实现偶数个数据元素

The same method for constructing a tree from four transactions can be generalized to construct trees of any size. In bitcoin it is common to have several hundred to more than a thousand transactions in a single block, which are summarized in exactly the same way, producing just 32 bytes of data as the single merkle root.
这种merkle树的构造方法可以推广到任意个元素的树。
在比特币中，一个区块中包含成百上千的交易是很常见的，使用同样的方法归纳这些交易，产生一个32字节的Merkle根。

In A merkle tree summarizing many data elements, you will see a tree built from 16 transactions. Note that although the root looks bigger than the leaf nodes in the diagram, it is the exact same size, just 32 bytes. Whether there is one transaction or a hundred thousand transactions in the block, the merkle root always summarizes them into 32 bytes.
在图4中，是用16个交易构造的一个树。
注意，虽然图中的根看起来比叶节点大，但实际上都是32字节。
无论区块中有一个交易或上万个交易，Merkle根都是把所有交易归纳为32字节。

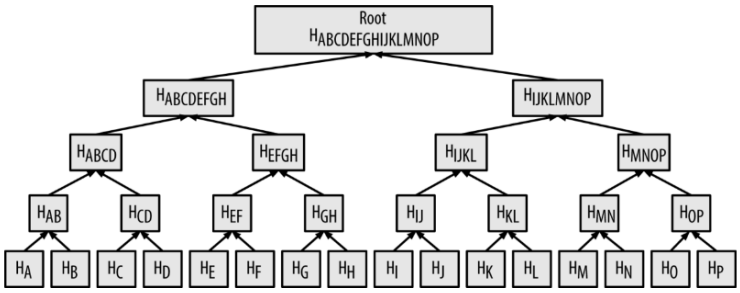


图9-4Merkle树汇总了许多数据元素

Figure 4. A merkle tree summarizing many data elements
图4：一个merkle树，归纳了许多数据元素

To prove that a specific transaction is included in a block, a node only needs to produce log~2~(N) 32-byte hashes, constituting an *authentication path* or *merkle path* connecting the specific transaction to the root of the tree. This is especially important as the number of transactions increases, because the base-2 logarithm of the number of transactions increases much more slowly. This allows bitcoin nodes to efficiently produce paths of 10 or 12 hashes (320–384 bytes), which can provide proof of a single transaction out of more than a thousand transactions in a megabyte-sized block.
为了证明一个区块中包含某个特定的交易，一个节点只需要生成log2(N)个32字节的哈希，组成一个认证路径（merkle路径），它把特定交易与merkle根连接起来。
随着交易数量的增加，这个就特别重要，因为log2(N)增长要慢得多。
这使得比特币节点高效地产生10或12个哈希（320-384字节）的路径，它能证明一个交易是在一个1M字节的区块中（有上千个交易）。

In A merkle path used to prove inclusion of a data element, a node can prove that a transaction K is included in the block by producing a merkle path that is only four 32-byte hashes long (128 bytes total). The path consists of the four hashes (shown with a shaded background in A merkle path used to prove inclusion of a data element) H_L, H_{IJ}, H_{MNOP}, and $H_{ABCDEFGH}$. With those four hashes provided as an authentication path, any node can prove that H_K (with a black background at the bottom of the diagram) is included in the merkle root by computing four additional pair-wise hashes H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$, and the merkle tree root (outlined in a dashed line in the diagram).
在图5中，节点可以证明交易K包含在这个区块中，方法是：生成一个merkle路径，它只有4个32字节哈希（总128字节）。
这个路径包含4个哈希（在图5中由蓝色表示）：H_L, H_{IJ}, H_{MNOP}, $H_{ABCDEFGH}$。
这4个哈希提供了一个认证路径，任何节点都可以证明H_K包含在这个merkle树中，方法是计算四个哈希：H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$，merkle root（在图中用虚线表示）。

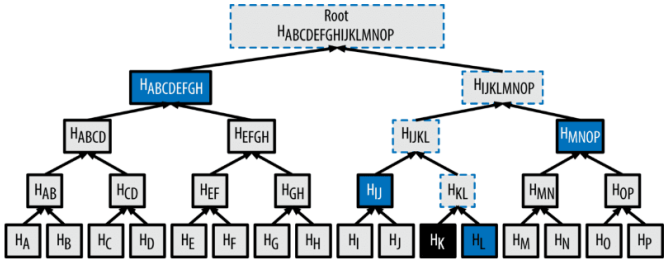


图9-5用于证明包含数据元素的merkle路径

Figure 5. A merkle path used to prove inclusion of a data element
图5：一个merkle路径，用于证明包含一个数据元素

The code in Building a merkle tree demonstrates the process of creating a merkle tree from the leaf-node hashes up to the root, using the libbitcoin library for some helper functions.
例1中的代码演示了创建merkle树的过程，它使用了libbitcoin库中的一些辅助程序。

Example 1. Building a merkle tree
```
link:code/merkle.cpp[]
```

Compiling and running the merkle example code shows the result of compiling and running the merkle code.
例2显示了编译和运行这个merkle代码的结果。

Example 2. Compiling and running the merkle example code
例2：编译和运行这个merkle例子代码

```
$ # Compile the merkle.cpp code
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)

$ # Run the merkle executable
$ ./merkle
Current merkle hash list:
  32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006
  30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4

Current merkle hash list:
  d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3
```

The efficiency of merkle trees becomes obvious as the scale increases. Merkle tree efficiency shows the amount of data that needs to be exchanged as a merkle path to prove that a transaction is part of a block.
随着规模的增加，Merkle树的效率变得明显。
表3展示了，为了证明区块中包含某个交易，而需要交换的一个Merkle路径的数据量。

Table 3. Merkle tree efficiency

Number of transactions	Approx. size of block	Path size (hashes)	Path size (bytes)
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65,535 transactions	16 megabytes	16 hashes	512 bytes

As you can see from the table, while the block size increases rapidly, from 4 KB with 16 transactions to a block size of 16 MB to fit 65,535 transactions, the merkle path required

to prove the inclusion of a transaction increases much more slowly, from 128 bytes to only 512 bytes.
从表中可以看出，虽然区块大小快速增加，从16个交易（4KB）到65,535个交易（16MB），但Merkle路径增长得很慢，仅仅从128字节增加到512字节。

With merkle trees, a node can download just the block headers (80 bytes per block) and still be able to identify a transaction's inclusion in a block by retrieving a small merkle path from a full node, without storing or transmitting the vast majority of the blockchain, which might be several gigabytes in size. Nodes that do not maintain a full blockchain, called simplified payment verification (SPV) nodes, use merkle paths to verify transactions without downloading full blocks.
有了Merkle树，一个节点可以只下载区块头（每个区块头是80字节），仍然能够识别一个交易在一个区块中，方法是：从一个全节点那里获取一个merkle路径。这样就不需要存储和传输区块链的主要部分（它可能有几个G字节）。
这种不需要维持完整区块链的节点，称为SPV节点，它使用merkle路径验证交易，不需要下载完整区块。

9.8 Merkle树和SPV

Merkle trees are used extensively by SPV nodes. SPV nodes don't have all transactions and do not download full blocks, just block headers. In order to verify that a transaction is included in a block, without having to download all the transactions in the block, they use an authentication path, or merkle path.
SPV广泛地使用了Merkle树。
SPV节点没有所有交易，没有下载完整区块，它只下载区块头。
为了验证一个交易包含在一个区块中，SPV不是下载这区块中的所有交易，而是使用一个认证路径（merkle路径）。

Consider, for example, an SPV node that is interested in incoming payments to an address contained in its wallet. The SPV node will establish a bloom filter (see [bloom_filters]) on its connections to peers to limit the transactions received to only those containing addresses of interest.
例如，一个SPV节点对钱包中某个地址收到的钱干兴趣。
这个SPV节点就在与peer的连接上建立一个bloom过滤器，用来限制接收的交易，只接收包含感兴趣的地址的交易。

When a peer sees a transaction that matches the bloom filter, it will send that block using a merkleblock message. The merkleblock message contains the block header as well as a merkle path that links the transaction of interest to the merkle root in the block. The SPV node can use this merkle path to connect the transaction to the block and verify that the transaction is included in the block. The SPV node also uses the block header to link the block to the rest of the blockchain. The combination of these two links, between the transaction and block, and between the block and blockchain, proves that the transaction is recorded in the blockchain.
当peer收到一个区块，看到一个交易与bloom过滤器匹配时，就用merkleblock消息发送那个区块。
这个merkleblock消息包含：区块头、一个merkle路径（连接交易和merkle根）。
SPV节点使用这个merkle路径来连接这个交易到这个区块，并验证这个交易包含在这个区块中。
SPV节点还会用区块头来连接这个区块到区块链的其它部分。
把这两个联系合并（交易与区块的联系、区块与区块链的联系），就能证明这个交易记录在区块链中。

All in all, the SPV node will have received less than a kilobyte of data for the block header and merkle path, an amount of data that is more than a thousand times less than a full block (about 1 megabyte currently).
总而言之，SPV节点将接收少于1K字节的数据（区块头和Melkle路径），而一个完整区块大约是1M字节。

9.9比特币的测试区块链

You might be surprised to learn that there is more than one bitcoin blockchain. The "main" bitcoin blockchain, the one created by Satoshi Nakamoto on January 3rd, 2009, the one with the genesis block we studied in this chapter, is called *mainnet*. There are other bitcoin blockchains that are used for testing purposes: at this time *testnet*, *segnet*, and *regtest*. Let's look at each in turn.
你可能会惊讶地发现：有多个比特币区块链。
2009-1-3，中本聪创建了"主比特币块链"，即本章研究的创世区块所在的区块链，称为mainnet。
还有其它用于测试的比特币区块链，目前有：testnet、segnet、regtest。
我们依次看一看。

9.9.1 Testnet：比特币的试验场

Testnet is the name of the test blockchain, network, and currency that is used for testing purposes. The testnet is a fully featured live P2P network, with wallets, test bitcoins (testnet coins), mining, and all the other features of mainnet. There are really only two differences: testnet coins are meant to be worthless and mining difficulty should be low enough that anyone can mine testnet coins relatively easily (keeping them worthless).
testnet是用于测试目的，是测试区块链、网络和货币的名称。
testnet是一个全功能活的P2P网络，包括：钱包、测试比特币、挖矿，以及mainnet的所有其它功能。

它与mainnet只有两个区别：testnet币毫无价值，挖掘难度很低，任何人都可以容易挖到testnet币。

Any software development that is intended for production use on bitcoin's mainnet should first be tested on testnet with test coins. This protects both the developers from monetary losses due to bugs and the network from unintended behavior due to bugs.
任何打算在比特币主干网上应用软件开发都应该先在testnet上用测试币进行测试。
这样可以防止由于软件错误而导致的金钱损失，也可以防止网络由于软件错误出现意外行为。

Keeping the coins worthless and the mining easy, however, is not easy. Despite pleas from developers, some people use advanced mining equipment (GPUs and ASICs) to mine on testnet. This increases the difficulty, makes it impossible to mine with a CPU, and eventually makes it difficult enough to get test coins that people start valuing them, so they're not worthless. As a result, every now and then, the testnet has to be scrapped and restarted from a new genesis block, resetting the difficulty.
但是，保持测试币的无价值和易挖掘并不容易。
尽管有来自开发者的呼吁，但还是有人使用先进的设备（GPU和ASIC）在testnet上挖矿。
这就增加了难度，导致使用CPU挖矿不能挖到矿，使获取测试币非常困难，以致于人们开始赋予测试币一定的价值，所以测试币并不是毫无价值。
结果，时不时地，testnet必须被报废，并重新从创始区块启动，重新进行难度设置。

The current testnet is called *testnet3*, the third iteration of testnet, restarted in February 2011 to reset the difficulty from the previous testnet.
目前的testnet被称为testnet3，是testnet的第三次迭代，于2011年2月重启，重置了之前的testnet网络的难度。

Keep in mind that testnet3 is a large blockchain, in excess of 20 GB in early 2017. It will take a day or so to sync fully and use up resources on your computer. Not as much as mainnet, but not exactly "lightweight" either. One good way to run a testnet node is as a virtual machine image (e.g., VirtualBox, Docker, Cloud Server, etc.) dedicated for that purpose.
记住，testnet3是一个大区块链，在2017年初超过20 GB。

完全同步需要一天左右的时间，并占用你的计算机资源。
它不像主干网，也不是轻量级。
运行testnet节点的一个好方法是，将其运行为一个专用的虚拟机镜像，例如VirtualBox、Docker、Cloud Server等。

9.9.1.1使用testnet

Bitcoin Core, like almost all other bitcoin software, has full support for operation on testnet instead of mainnet. All of Bitcoin Core's functions work on testnet, including the wallet, mining testnet coins, and syncing a full testnet node.
像几乎所有其它比特币软件一样，Bitcoin Core完全支持在testnet网络运行，而不是只能在主干网上运行，还允许你进行测试币挖矿，并运行一个testnet全节点。

To start Bitcoin Core on testnet instead of mainnet you use the testnet switch:
如果要在testnet上启动Bitcoin Core，而不是在主干网启动，可以使用testnet开关：

```
$ bitcoind -testnet
```

In the logs you should see that bitcoind is building a new blockchain in the testnet3 subdirectory of the default bitcoind directory:
在日志中，应该会看到，bitcoind在testnet3子目录中构建一个新的区块链：

```
bitcoind: Using data directory /home/username/.bitcoin/testnet3
```

To connect to bitcoind, you use the bitcoin-cli command-line tool, but you must also switch it to testnet mode:
要连接bitcoind，可以使用bitcoin-cli，但是要记得切换到testnet模式：

```
$ bitcoin-cli -testnet getblockchaininfo
{
  "chain"          : "test",
  "blocks"                  : 1088,
  "headers"        : 139999,
  "bestblockhash": "0000000063d29909d475a1c4ba26da64b368e56cce5d925097bf3a2084370128",
  "difficulty"     : 1,
  "mediantime"     : 1337966158,
  "verificationprogress": 0.001644065914099759,
  "chainwork": "0000000000000000000000000000000000000000000000000044104410441",
  "pruned"                  : false,
  "softforks"      : [
  [...]
```

You can also run on testnet3 with other full-node implementations, such as btcd (written in Go) and bcoin (written in JavaScript), to experiment and learn in other programming languages and frameworks.
你也可以用其它全节点运行在testnet3上，例如btcd（Go编写）和bcion（JavaScript编写），以体验和学习其它语言和框架。

In early 2017, testnet3 supports all the features of mainnet, including Segregated Witness (see [segwit]). Therefore, testnet3 can also be used to test Segregated Witness features.
在2017年初，testnet3支持主干网的所有功能，包括隔离见证。
因此，testnet3也可用于测试隔离见证功能。

9.9.2 Segnet：隔离见证测试网络

In 2016, a special-purpose testnet was launched to aid in development and testing of Segregated Witness (aka segwit; see [segwit]). This test blockchain is called segnet and can be joined by running a special version (branch) of Bitcoin Core.
2016年，启动了一个特殊用途的测试网络，帮助开发和测试隔离见证（segwit）。
这个测试区块链称为segnet，可以通过运行Bitcoin Core的一个特殊版本来连接。

Since segwit was added to testnet3, it is no longer necessary to use segnet for testing of segwit features.
由于已经将segwit添加到testnet3中，因此后来不再使用segnet来测试segwit功能。

In the future it is likely we will see other testnet blockchains that are specifically designed to test a single feature or major architectural change, like segnet.
在将来，我们可能会看到其它专门用于测试某个功能或主要架构修改的测试网络区块链。

9.9.3 Regtest：本地区块链

Regtest, which stands for "Regression Testing," is a Bitcoin Core feature that allows you to create a local blockchain for testing purposes. Unlike testnet3, which is a public and shared test blockchain, the regtest blockchains are intended to be run as closed systems for local testing. You launch a regtest blockchain from scratch, creating a local genesis block. You may add other nodes to the network, or run it with a single node only to test the Bitcoin Core software.
Regtest代表“回归测试”，是一种Bitcoin Core功能，允许你创建一个本地区块链来进行测试。
与testnet3（一个公共和共享的测试区块链）不同，regtest区块链只作为本地测试的封闭系统运行。
你从零开始启动regtest区块链，创建一个本地的创世区块。
你可以将其它节点添加到这个网络中，或者运行单个节点来测试Bitcoin Core软件。

To start Bitcoin Core in regtest mode, you use the regtest flag:
要在regtest模式下启动Bitcoin Core，您可以使用regtest标志：

```
$ bitcoind -regtest
```

Just like with testnet, Bitcoin Core will initialize a new blockchain under the *regtest* subdirectory of your bitcoind default directory:
像testnet一样，Bitcoin Core将在bitcoind默认目录的regtest子目录下初始化一个新的区块链：

```
bitcoind: Using data directory /home/username/.bitcoin/regtest
```

To use the command-line tool, you need to specify the regtest flag too. Let’s try the getblockchaininfo command to inspect the regtest blockchain:
要使用命令行工具，还需要指定regtest标志。
我们用getblockchaininfo命令来检查regtest区块链：

```
$ bitcoin-cli -regtest getblockchaininfo
{
  "chain"           : "regtest",
  "blocks"                    : 0,
  "headers"         : 0,
  "bestblockhash"   :
"0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206",
  "difficulty"      : 4.656542373906925e-10,
  "mediantime"      : 1296688602,
  "verificationprogress": 1,
```

```
  "chainwork"      :
"0000000000000000000000000000000000000000000000000000000000000002",
  "pruned"                  : false,
  [...]
```

As you can see, there are no blocks yet. Let's mine some (500 blocks) and earn the reward:
你可以看到，还没有任何区块。
我们挖一些（500个区块），赚取奖励：

```
$ bitcoin-cli -regtest generate 500
[
  "7afed70259f22c2bf11e406cb12ed5c0657b6e16a6477a9f8b28e2046b5ba1ca",
  "1aca2f154a80a9863a9aac4c72047a6d3f385c4eec5441a4aafa6acaa1dada14",
  "4334ecf6fb022f30fbd764c3ee778fabbd53b4a4d1950eae8a91f1f5158ed2d1",
  "5f951d34065efeaf64e54e91d00b260294fcdfc7f05dbb5599aec84b957a7766",
  "43744b5e77c1dfece9d05ab5f0e6796ebe627303163547e69e27f55d0f2b9353",
   [...]
  "6c31585a48d4fc2b3fd25521f4515b18aefb59d0def82bd9c2185c4ecb754327"
]
```

It will only take a few seconds to mine all these blocks, which certainly makes it easy for testing. If you check your wallet balance, you will see that you earned reward for the first 400 blocks (coinbase rewards must be 100 blocks deep before you can spend them):
挖掘这些区块只需要几秒钟，这样就可以很容易地进行测试。
如果你检查钱包余额，会看到获得了前400个区块的奖励（币基奖励必须挖满100个区块后才能花费）：

```
$ bitcoin-cli -regtest getbalance
12462.50000000
```

9.10使用测试区块链进行开发

Bitcoin's various blockchains (regtest, segnet, testnet3, mainnet) offer a range of testing environments for bitcoin development. Use the test blockchains whether you are developing for Bitcoin Core, or another full-node consensus client; an application such as a wallet, exchange, ecommerce site; or even developing novel smart contracts and complex scripts.
Bitcoin的各种区块链（mainnet，regtest，segnet，testnet3）为比特币开发提供了一系列测试环境。
无论你是为下面哪个做开发，请使用测试区块链：

- Bitcoin Core，或其它全节点共识客户端
- 应用程序，例如：钱包、交易所、电子商务网站
- 新颖的智能合约和复杂脚本

You can use the test blockchains to establish a development pipeline. Test your code locally on a regtest as you develop it. Once you are ready to try it on a public network, switch to testnet to expose your code to a more dynamic environment with more diversity of code and applications. Finally, once you are confident your code works as expected, switch to mainnet to deploy it in production.
你可以使用测试区块链来建立一个开发管道。

- 开发时，在regtest上测试你的代码。
- 一旦准备在公网上尝试，就切换到testnet，将你的代码放在更加动态的环境中，这里有各种代码和应用。
- 最后，一旦确信你的代码能正常工作，就切换到mainnet，以实现实际应用。

As you make changes, improvements, bug fixes, etc., start the pipeline again, deploying each change first on regtest, then on testnet, and finally into production.
当你进行变更时、改进、修改bug时，再次启动这个开发管道，
首先在regtest上部署变更，然后是testnet，最后实际应用。