读书笔记

第 3 篇

区块链的机制

从这里开始，我们将通过执行 Python 程序来了解区块链的具体机制。目前为止我们介绍了一些区块链以及 Python 语言的基础知识，让我们一边巩固一边继续深入学习吧。

第7章 区块链的结构

当已成功学会 Python 语言时，就可以开始了解区块链的结构了。首先，我们将学习理解区块链结构所必需的哈希函数知识，之后学习区块链本身的知识点。

7.1 哈希函数

由于哈希函数具有使用非常方便的特点，所以频繁地被应用在区块链的各个方面。下面我们将详细介绍哈希函数。

7.1.1 什么是哈希函数

哈希函数是根据某些规则将输入数据转换为不同数据的函数。区块链中哈希函数的特点主要有如下四点。

（1）只能在一个方向上进行计算，而不能逆向进行计算。（不可逆性）

（2）如果输入数据发生哪怕仅仅是很小的变化，也会使得输出数据发生很大的变化。（机密性）

（3）不论输入数据长度如何，输出数据都将是相同长度的数据。（固定长度）

（4）可以从输入数据轻松、便捷地计算输出数据。（处理速度）

哈希函数所具有的不可逆向运算的这一特点（不可逆性），可以用于隐藏不希望第三方知道的信息。另外，哈希函数还具有即使输入数据发生很细微的变化，输出数据也会发生显著变化的特点（机密性），这一特点可用于检测原始数据是否已被篡改，如图 7.1 所示。此外，无论输入数据的长度如何，输出数据的长度都将完全相同（固定长度），这个特点可以用于汇总大量数据并压缩数据的大小。

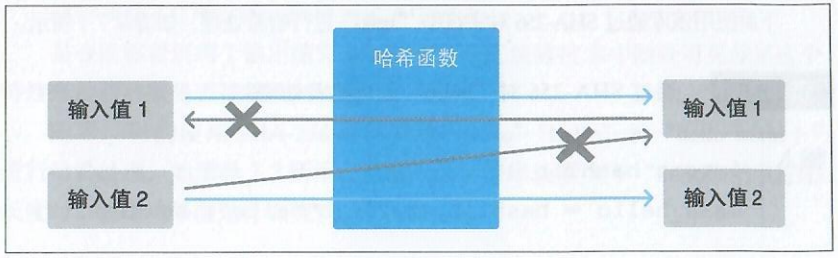

图 7.1 哈希函数的特点

7.1.2 哈希函数的种类

目前已经开发出了几种类型的哈希函数，除区块链以外，还被应用在各个

领域。表 7.1 列出了区块链技术中使用的主要哈希函数。

表 7.1　主要的哈希函数

哈希函数	说　明
SHA-256	Secure Hash Algorism 256 bit 的简称，是一种无论输入数据的大小如何，都会生成 256 位哈希值的哈希函数。该技术在整个区块链技术中被广泛应用，经常可以见到连续两次使用 SHA-256 进行运算的做法
RIPEMD-160	RACE Integrity Primitives Evaluation Message Digest 的简称，生成一个 160 位的哈希值。由于可以生成一个小于 SHA-256 的哈希值，应用于需要节省数据容量的情况
HMAC-SHA516	HMAC-SHA516 是一种通过使用键值对作为输入值来获得 516 位哈希值的哈希函数。用于分层确定性钱包（HD 钱包）

SHA 系列是由 NIST（National Institute of Standards and Technology，美国国家标准技术研究院）进行标准化的，另外还有其他诸如 SHA-384、SHA-516 等。曾经存在 SHA-1 标准并被广泛使用，但是由 CWI Amsterdam（URL https://www.cwi.nl/）和 Google Research 的联合研究小组发现该标准存在一个安全漏洞，目前已经不推荐使用。尽管不能称为绝对安全，但上述哈希函数目前被认为是安全的。

7.1.3 尝试应用哈希函数

由于已经准备了 Python 的标准库 hashlib，因此可以方便地使用它来计算哈希函数。

下面使用该库通过 SHA-256 对字符串“hello”进行哈希处理，如清单 7.1 所示。

清单 7.1　通过 SHA-256 对“hello”进行哈希处理

输入

```
import hashlib
hash_hello = hashlib.sha256(b"hello").hexdigest()

print(hash_hello)
```

输出

```
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa742➥
5e73043362938b9824
```


执行清单 7.1 所示的代码得到输出中的值。

清单 7.1 中的 hashlib.sha256（b"hello"）.hexdigest() 部分，是使用 SHA-256 进行哈希处理的。进行哈希处理的数据必须为字节类型，因此写为 b"hello"。在 .hexdigest() 部分中，是将其转换为十六进制格式的字符串。

由于哈希值是将输入值和输出值以一一对应的方式进行关联，因此，如果对完全相同的字符串进行哈希处理，将输出相同的哈希值。但是，如果输入值略有变化，则输出值将会有很大不同。因此，我们尝试使用 SHA-256 对与“hello”稍有不同的字符串“hallo”进行哈希处理。如清单 7.2 所示。

清单 7.2　通过 SHA-256 对“hallo”进行哈希处理

输入

```
import hashlib

hash_hello = hashlib.sha256(b"hello").hexdigest()
hash_hallo = hashlib.sha256(b"hallo").hexdigest()
print(hash_hello)
print(hash_hallo)
```

输出

```
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e730433➥
62938b9824
d3751d33f9cd5049c4af2b462735457e4d3baf130bcbb87f389e34➥
9fbaeb20b9
```

执行清单 7.2 所示的代码得到输出中的值。

是否能够看到两个输出值完全不同呢？在区块链技术中随处可见使用这个特性来检测数据是否被篡改的机制。

随后，我们使用 SHA-256 对字符串“hello”“hallo”和“hello world！”进行哈希处理，如清单 7.3 所示。当输入的字符串长度大于“hello”的字符串长度时，输出结果会是什么呢？

清单 7.3　通过 SHA-256 对各种字符串进行哈希处理

输入

```
import hashlib

hash_hello = hashlib.sha256(b"hel lo").hexdigest()
```

```
hash_hallo = hashlib.sha256(b"hallo").hexdigest()
hash_helloworld = hashlib.sha256(b"hello world!").hexdigest()

print(hash_hello)
print(hash_hallo)
print(hash_helloworld)
```

输出

```
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e730433
62938b9824
d3751d33f9cd5049c4af2b462735457e4d3baf130bcbb87f389e34
9fbaeb20b9
7509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df16
3be08e6ca9
```

执行清单 7.3 所示的代码得到输出中的值。

当然，由于输入值发生了变化，所以输出值也会发生很大变化。尽管输入值的长度增加了，但是从输出结果可以看到输出的哈希值长度并没有变化。哈希函数的这个特点非常重要，并且可广泛应用于各种场合，因此，请结合代码的编写方法来掌握它的这一特性。

7.2 区块的内容

正如已经看到的那样，区块链是将被称为区块的数据整理成锁链状的结构。下面，我们将详细分析一下区块链的结构。

7.2.1 区块内部结构

区块的内部如图 7.2 所示。其中，最重要的信息是 blockheader（以下称为区块头）、Txsvi 和 Txs（以下称为交易）。如表 7.2 所示总结了各个构成元素的特点。

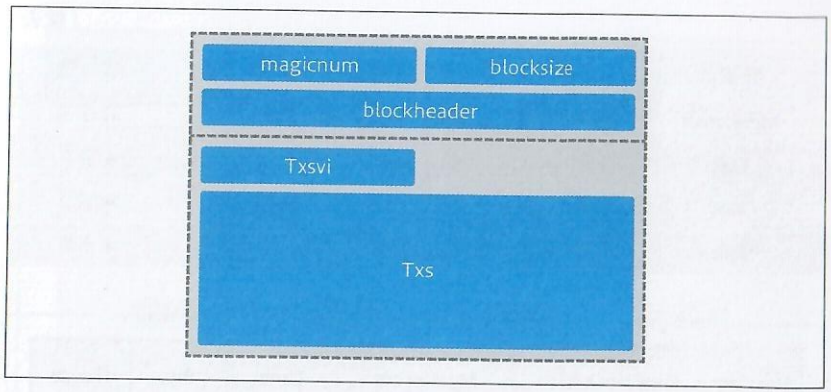

图 7.2 区块内部

表 7.2 区块内容的构成元素

数据项	说 明
magicnum	uint32（32 位无符号整数数组）
blocksize	uint32（32 位无符号整数数组）
blockheader	关于区块头的信息
Txsvi	交易数量
Txs	交易列表

区块头中包含有关于区块的重要数据。交易部分包含经过采矿过程而存储在区块中的交易以及相关信息。

对于比特币来说，一个区块的数据容量限制为 1MB。虽然上述的数据，假设其合计最多为 1 MB，但是根据区块链类型的不同，也存在有更大的区块容量的可能性，这一数据的容量会极大地影响到每种区块链的设计理念。

7.2.2 区块头

区块头中包含如表 7.3 所示的 80 个字节的数据，如图 7.3 所示。

表 7.3 区块头的结构

数据项	说 明	容 量
Version	版本	4 字节
Prev block hash	前面一个区块的哈希值	32 字节

（续表）

数据项	说　明	容　量
Merkleroot	使用哈希函数生成的所有交易的哈希值	32 字节
Time	指示生成区块时间的时间戳	4 字节
Bits	采矿的难易度	4 字节
Nonce	在采矿中满足条件的随机数	4 字节

图 7.3　区块头的示意图

Prev block hash 是将前一个区块头通过哈希函数来生成的。换句话说，前一个区块头的哈希值被带入下一个区块的区块头中，并且它们像锁链一样连接。因此，使用哈希函数将区块像锁链一样一一进行连接的结构也称为哈希链。

7.2.3 交易

经过采矿可以获得的最大交易数量为 3000 个。这些交易以列表的形式存储，并且根据交易数量的不同可能存在有较大的数据容量发生的情况。虽然通常情况下，仅仅存储交易列表。但是也存在某些约束，即如果交易之间具有依赖关系时，则必须先将父交易放在其中。但是，如果需要考虑交易数据的先后顺序并将交易数据存储在区块中，即当一定的矿工在采矿时，甚至需要互相交换关于区块顺序的信息，这将降低操作的效率。因此，采用一种名为 CTOR（Canonical Transaction Ordering Rule 交易规范排序规则）的方法，即将交易的哈希值升序（字母顺序）排列。

使用称为 Merkletree（默克尔树）的数据结构对交易进行哈希处理，并汇总为 Merkleroot（默克尔根）。默克尔根存储在该区块的区块头中，用于验证交易是否正确，如图 7.4 所示。第 14 章将详细讨论默克尔根。

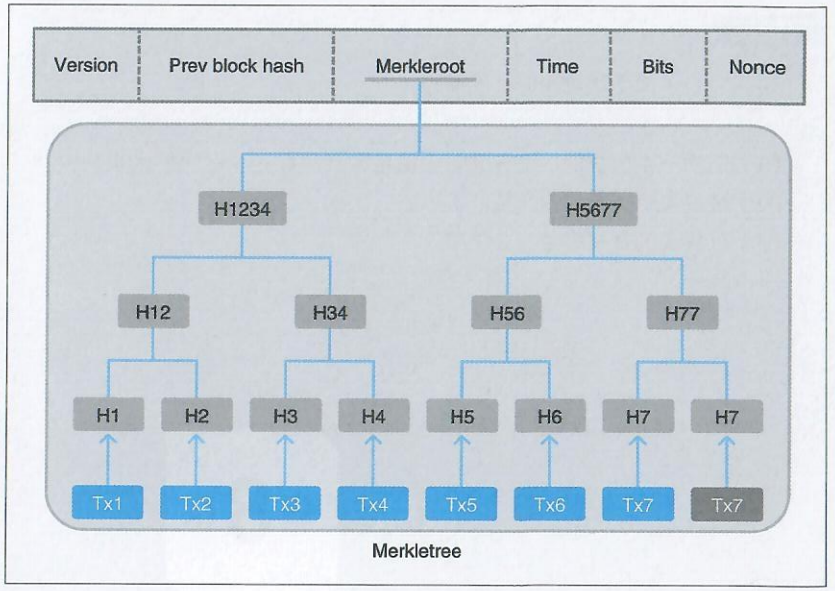

图 7.4　默克尔根和默克尔树

本章习题

问题一

请选择一个错误的关于哈希函数特征的描述。

（1）输入数据稍有变化，输出数据就会发生很大变化。
（2）基本上不能从输出数据逆向运算出输入数据。
（3）根据输入数据的长度可以调整输出数据的长度。

问题二

通过 SHA-256 计算 Bonjour 的哈希值。