

10.1 比特币区块链的交易

第10章 交易

比特币区块链上的交易包括输入（input）、输出（output）以及其他关联数据。

10.1.1 比特币中交易数据的结构

在比特币中创建和管理交易数据是区块链技术非常重要的部分。交易是描述多少金额以及如何汇款的数据。交易数据本身不包含任何用于识别个人的特定的数据，例如由谁汇款给谁这样的数据，而是通过使用私密密钥或公开密钥的数字签名的方式来证明所有权的。

区块链是以串珠状连接各个区块，以形成前后依赖关系并保持其整体的数据一致性。交易数据也具有相同的链状结构，主要由输入和输出组成，如图 10.1 所示。

交易是很多区块链中最重要的机制之一。在这里，我们以所有区块链的原型——比特币的交易机制为中心来进行讲解。

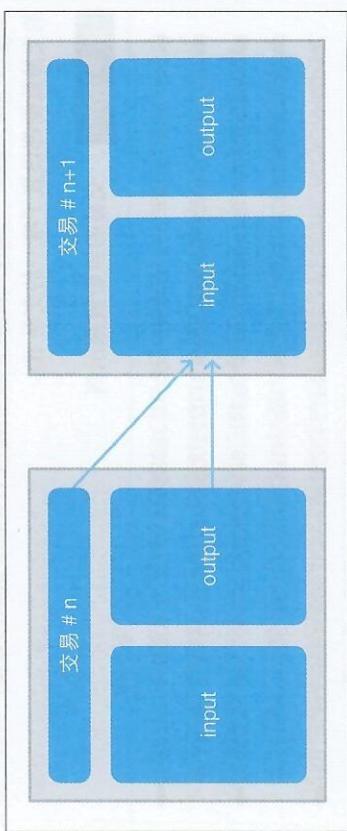


图 10.1 交易的构成

交易数据的基本构造如表 10.1 所示。

表 10.1 交易数据的构造

数据项	长 度 (字节)	内 容
Version	4	指定要遵循的规则。通常为 1。限制交易使用期限时指定为 2

(续表)

数据项	长 度 (字节)	内 容
Input counter	1 ~ 9	该交易数据中包含的输入数据的数量
Inputs	可变长	数据输入
Output counter	1 ~ 9	该交易数据中包含的输出数据的数量
Outputs	可变长	数据输出
Locktime	4	在此处设置值以指定存储在区块中的时间

从图 10.1 和表 10.1 中可以看到，交易数据由输入、输出以及其他相关数据组成，如同一种链状的形式。

10.1.2 查看实际的交易数据

让我们看看实际的交易数据。如清单 10.1 所示的数据是原始交易数据的示例。这是包含在区块高度为 111111 的区块中的交易 ID 为 0e942bb178dbf7ae40d36d238d59427429641689a379fc43929f15275a75fa6 的数据。同时，该数据是由使用全节点客户端的 Bitcoin Core 创建的全节点所获取的数据。而且，所有的交易数据都设置了唯一的交易 ID。

清单 10.1 交易的原始数据示例

```
01000000245d50a313eb89a71bb501380334e58973b7b3c0fb4614f8645↑
d07d728b8574aa010000008c493046022100b3c7f1c56384c6145673b94f↑
a226af8d02a263a1ed9f3505fd0e94cc681193e022100d677eb3ctb9b82da961626cc↑
da961626ccfdc31a6041195123ae236b5ed1a34250c4163e73[ALL] 04545e7c6d2acc↑
7c6d2acc1c161567860039bc3068d4af5432b1afe34d2909bf8996b95c10a6↑
445692bc3f458a0503c45084f99329d5e90d84f21a52a5d11add62eb26340db2",↑
0db2fffffff49de1cd30c20d1f22a2e966b0d195d20b0f133e8de426f7d↑
1610525a64a09daf000000008b483045022100d4ea6ce46296548d38f02d↑
a337daebef00f443134f3b68ff4ad8946a961728b0402207573dc96e32403↑
7d1934fa5105a517608364a2b273e694dce53714745fa135f4014104e896↑
67c1c2cf4eb91324debcc7742b8eb22406c59d6033794124efb7cce37b78↑
dd5e3c3474aeb9dbf7689c9d36d776b7d21debe4d75142b43a2529fb3a↑
6da4fffff0100286bee000000001976a914c1ac9042b50a9d2e7a6a1b↑
42bad6e61a9ec3f6b88ac00000000
```

请注意，该数据是十六进制序列化数据。猛一看，无法理解是什么含义，但是转换为 JSON 格式，则如清单 10.2 所示。

清单 10.2 转换为 JSON 格式

```
{
  "txid": "0e942bb178dbf7ae40d36d238d59427429641689a379fc43929f15275a75fa6",
  "hash": "0e942bb178dbf7ae40d36d238d59427429641689a379fc43929f15275a75fa6",
  "version": 1,
  "size": 405,
  "vsize": 405,
  "weight": 1620,
  "locktime": 0,
  "vin": [
    {
      "txid": "aa74858b727dd045864f61b40f3c7b3b97584e3380135",
      "vout": 1,
      "scriptSig": {
        "asm": "3046022100b3c7f1c56384c6145673b94fa226af8d02a263a1ed9f3505fd0e94cc681193e022100d677eb3ctb9b82da961626ccfdc31a6041195123ae236b5ed1a34250c4163e73[ALL] 04545e7c6d2acc161567860039bc3068d4af5432b1afe34d2909bf8996b95c10a6445692bc3f458a0503c45084f99329d5e90d84f21a52a5d11add62eb26340db2",
        "hex": "493046022100b3c7f1c56384c6145673b94fa226af8d02a263a1ed9f3505fd0e94cc681193e022100d677eb3ctb9b82da961626ccfdc31a6041195123ae236b5ed1a34250c4163e73014104545e7c6d2acc161567860039bc3068d4af5432b1afe34d2909bf8996b95c10a6445692bc3f458a0503c45084f99329d5e90d84f21a52a5d11add62eb26340db2"
      },
      "sequence": 4294967295
    }
  ],
  "outputs": [
    {
      "txid": "af9da0645a5210167d6f42de833f1b0205d190d6b962"
    }
  ]
}
```

```

e2af2d1200cd31cde49",
  "vout": 0,
    "scriptSig": {
      "asm": "3045022100d4ea6ce46296548d38f02da337daebef00",
      "hex": "443134f3b68f4ad8946a961728b0402207573dc96e324037d1934fa5105",
      "n": 0,
      "sequence": 4294967295
    },
    "vout": [
      {
        "txid": "aa7485b7727dd045864f61b40f3c7b3b975b4e33801350bb719ab83e310ad545",
        "vout": 1,
        "scriptSig": {
          "asm": "3046022100b3c7f1c56384c6145673b94fb226af8d02a263a1ed9f3505fd0e..",
          "hex": "493046022100b3c7f1c56384c6145673b94fb226af8d02a263a1ed9f3505fd..",
          "n": 1,
          "sequence": 4294967295
        }
      },
      {
        "txid": "af9da0645a5210167d6f42ddee833f1b0205d19d6b962e2af2d1200cd31cd49",
        "vout": 0,
        "scriptSig": {
          "asm": "3045022100d4ea6ce46296548d38f02da337ddebef00f443134f3b68f4ad89..",
          "hex": "483045022100d4ea6ce46296548d38f02da337ddebef00f443134f3b68f4ad..",
          "n": 0,
          "sequence": 4294967295
        }
      }
    ],
    "vout": [
      {
        "txid": "742b8eb22406c59d6033794124efb7cce37b78dd5e3c347",
        "vout": 1,
        "scriptSig": {
          "asm": "OP_DUP OP_HASH160clac9042b50a9d2e7a6a1b42bad66e61a9ec3f",
          "hex": "66e61a9ec3f6b OP_EQUALVERIFY OP_CHECKSIG",
          "n": 1,
          "sequence": 4294967295
        }
      }
    ]
  }
}

```

开头的 0e942bb178lbf7ae4036d2380559427420641689a379fc43929ff5275a75fa6 是该交易的 ID。每笔交易都设置有唯一的 ID。另外，vin 和 vout 分别指交易的输入数据和输出数据。

vin 的数组部分是该交易的输入部分。在该示例中，可以看到如图 10.2 所示的两个输入数据的存储。由 txid 指定引用的交易 ID，并在 vout 部分中指定使用第几个输出数据。通过以这种方式使用 txid 和 vout，可以唯一确定正在引用哪个交易的哪个输出数据。在这里，可以看到引用了第一个输入数据的 txid 为 aa7485b727dd045864f61b40f3c7b3b975b4e33801350bb719ab83e310ad545 的交易的第二个输出数据。

```

① {
  "txid": "aa7485b7727dd045864f61b40f3c7b3b975b4e33801350bb719ab83e310ad545",
  "vout": 1,
  "scriptSig": {
    "asm": "3046022100b3c7f1c56384c6145673b94fb226af8d02a263a1ed9f3505fd0e..",
    "hex": "493046022100b3c7f1c56384c6145673b94fb226af8d02a263a1ed9f3505fd..",
    "n": 1,
    "sequence": 4294967295
  }
},
② {
  "txid": "af9da0645a5210167d6f42ddee833f1b0205d19d6b962e2af2d1200cd31cd49",
  "vout": 0,
  "scriptSig": {
    "asm": "3045022100d4ea6ce46296548d38f02da337ddebef00f443134f3b68f4ad89..",
    "hex": "483045022100d4ea6ce46296548d38f02da337ddebef00f443134f3b68f4ad..",
    "n": 0,
    "sequence": 4294967295
  }
}

```

图 10.2 交易的输入部分

vout 的数组部分是该交易的输出部分。在此示例中，仅存储一个。其中包括金额和收款人的地址等。asm 部分包含一个脚本，稍后将对其进行说明，定义了使用该输出数据的条件。

10.1.3 尝试获得交易数据

前面已经介绍的交易数据是通过使用 BitcoinCore 设置全节点获得的数据，但是也可以无须专门设置全节点而是使用公共 API 来获取数据。本节主要这种方法。首先，本次使用的开源 API 是称为 blockchain.info 的 Web 服务。还可以使用其他诸如 smartbit 之类的服务。如果使用开源 API，则无须自己拥有所有数据即可获取必要的数据。在 Python 中使用 API 时，要用到称为 requests

的第三方软件包。该软件包可以轻松地实现 HTTP 通信，并且可以通过指定开源 API 的 URL 来执行诸如提供和获取数据之类的操作。其次，在使用它之前，需要事先使用 pip install 命令安装它。

● [终端]

```
$ pip install requests
```

继续进行交易数据的获取处理。首先，本次获取的交易是区块高度为 11111 的区块中包含的数据，其交易 ID 为 0e942bb178dbf7ae40d36d238d59427429641689a379f c43929f15275a75fa6。有了这些信息，可以通过运行如清单 10.3 所示的程序来获取原始交易数据。

清单 10.3 从开源 API 获取原始交易数据的代码

```
输入
import requests

APIURL = "https://blockchain.info/rawtx/"

TXID = "0e942bb178dbf7ae40d36d238d59427429641689a379fc"
43929f15275a75fa6"

r = requests.get( APIURL + TXID + "?format=hex" )
print(r.text)
```

需要使用已安装的 requests 软件包，因此在代码最开始部分使用了 import requests 语句。接下来定义的 APIURL 和 TXID 是使用该 API 所需的基本信息。在 APIURL 中，设定了本次使用的开源 API 的 URL。同时，在 TXID 中，设定本次处理对象的交易数据的 ID。像这样预先定义基本信息，可以提高代码的可读性。例如，当要改变交易 ID 或使用其他的开源 API 时，仅需要改变事先定义的部分，这样做还可以防止出现不必要的错误。

在 requests.get(APIURL + TXID + "?format=hex") 语句中，将 URL 作为参数来向 blockchain.info API 服务器发送 GET 请求。由于 GET 方法是一种请求文件和数据的方法，因此，客户端可以获取所需的数据。在 URL 之后附加 "?format=hex" 的原因为了获取以十六进制形式编写的交易数据。另外，在 print(r.text) 语句中，.text 用以将响应正文部分转换为文本格式。顺便说一句，发出请求后返回的数据称为响应数据，分为几个部分，请求数据的内容也

包含在主体 (body) 部分中。因此，可以通过以文本格式输出响应数据的正文部分来获得如清单 10.4 所示的结果。

执行如清单 10.3 所示的代码，可以得到如清单 10.4 所示结果。

清单 10.4 使用开源 API 获取的交易数据

输出	01000000245d50a313eb89a71bb501300334e59973b7b3c0fb461 4f8645dd07d728b8574aa010000008c493046022100b3c7f1c56384 c6145673b94fa226af8d02a263a1ed9f3505fdf0e94cc681193e02 2100d67feb3cb69b8d2da961c26ccfdc3:a6041195123ae236b5ed a34250c4163e73014104545e7c6d2acc161567860039bc3068d4af 5432b1afe34d2909bf8996b95c10a6445692bc3f458a0503c45084 f99329d5e90d84f21a52a5d1add62eb26340db2fffffff49de1c d30c20d1f22a2e966b0d195d20b0f133e8de426f7d1610525a64a0 9daf00000008b483045022100d4ea6ce46296548d38f02da337da ebef00f443134f3b68f4ad8946a961728b0402207573dc9632403 7d1934fa5105a517608364a2b273e694dc53714745fa135f40141 04e89667c1c2cf4eb91324d8bc7742b8eb22406c59d6033794124 efb7cce37b78ddd5e3c3474aeb9dbf7689c9d36d776b7d21dbe4d 75142cb43a2529fb3a6da4fffff0100286bee00000001976a9 14c1ac9042b50a9d2e7a6a1b42bad66e61a9ec3f6b88ac00000000
----	---

让我们看看清单 10.4 的结果和前面介绍的十六进制的交易数据（清单 10.1）是否相匹配。但是，只看这样的输出结果，是很难读懂的。因此，让我们改变一下数据的形式。转换数据形式，可以通过运行如清单 10.5 所示中的代码来实现。与清单 10.3 的唯一区别是参数 URL 中没有 "?format = hex"。

清单 10.5 获得转换交易数据格式的代码

输入	import requests APIURL = "https://blockchain.info/rawtx/ TXID = "0e942bb178dbf7ae40d36d238d59427429641689a379fc" 43929f15275a75fa6" r = requests.get(APIURL + TXID) print(r.text)
----	--

```
APIURL = "https://blockchain.info/rawtx/"
TXID = "0e942bb178dbf7ae40d36d238d59427429641689a379fc"
43929f15275a75fa6"

r = requests.get( APIURL + TXID )
print(r.text)
```

```

{
  "sequence": "4294967295",
  "witness": "",
  "prev_out": {
    "spent": true,
    "spending_outputs": [
      {
        "tx_index": 323055,
        "type": 0,
        "addr": "17vCttx3McuR3Z7ch2Nz83EoJN9J9p1dvy",
        "value": 3500000000,
        "n": 0,
        "script": "76a9144be0a14399e3aad60a761f2e3f6"
      },
      {
        "tx_index": 323218,
        "type": 0,
        "addr": "483045022100d4ea6cce46296548d38f02da337daebe",
        "value": 324037d19,
        "n": 0,
        "script": "443134f3b68f4ad8946a9617280402207573dc96e324037d19"
      },
      {
        "tx_index": 321153,
        "type": 0,
        "addr": "1BNZJx7pM4GTrqe8MgEZoJi2FUS3fKnJH2i",
        "value": 500000000,
        "n": 1,
        "script": "76a91471c46b0556fef3cbdbae6f2976"
      },
      {
        "tx_index": 323055,
        "type": 0,
        "addr": "493046022100b3c7f1c56384cc6145673b94f",
        "value": 324037d19,
        "n": 0,
        "script": "493046022100b3c7f1c56384cc6145673b94f"
      }
    ],
    "block_height": 1111111,
    "relayed_by": "0.0.0.0",
    "out": [
      {
        "addr_tag_link": "https://bitcointalk.org/index.php?action=profile;u=4904",
        "addr_tag": "sanjay",
        "weight": 1620
      }
    ]
  }
}

```

如果运行清单 10.5 的代码，将获得如清单 10.6 所示的格式化数据。尽管格式与先前介绍的交易数据（清单 10.2）略有不同，但是可以看到内容几乎相同。特别是，让我们看看是否存在用于 input 和 output 的数据项，并且看看数据是否相对详细。

清单 10.6 获得的格式交易数据

```

    "spent": false,
    "tx_index": 323218,
    "type": 0,
    "addr": "1DF48wufCdpPQ2EEjX1jUUULuTHVzDhgBSF",
    "value": 4000000000,
    "n": 0,
    "script": "76a914c1ac9042b50a9d2e7a6a1b42bad66e",
    "61a9ec3f6b88ac"
  ],
  "lock_time": 0,
  "size": 405,
  "double_spend": false,
  "block_index": 125961,
  "time": 1298920129,
  "tx_index": 323218,
  "vin_sz": 2,
  "hash": "0e942bb178dbf7ae40d36d238d559427429641689a37",
  "9fc43929f15275a75fa6",
  "vout_sz": 1
}

```

交易输出，这样就生成了交易数据。

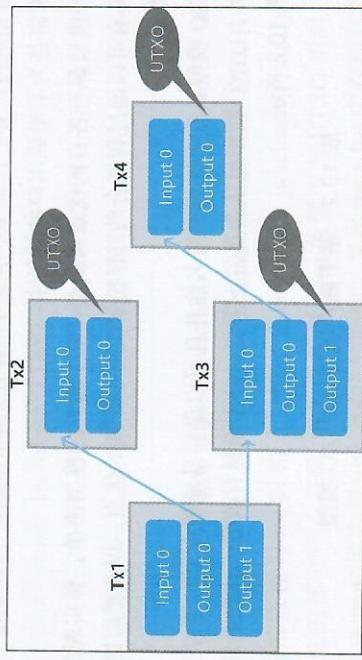


图 10.3 UTXO 的示意图

根据 10.1 节介绍的交易数据，如果存储在 vout 中的输出数据没有被利用，则该数据为 UTXO。我们可以看到，存储在 vin 中的两个输入数据在生成该交易时是 UTXO。

10.2.2 基于账户模型和 UTXO 模型

在区块链上管理虚拟货币和代币的余额主要有两种方法：基于账户的模型和 UTXO 模型，如图 10.4 所示。基于账户的模型管理每个账户有多少金额以及进出多少金额。想象为一个普通的银行账户，将很容易理解。以太坊使用基于账户的模型。

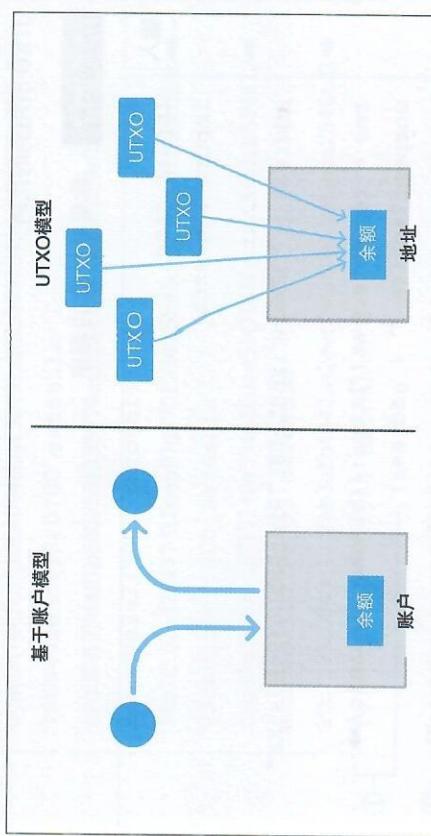


图 10.4 余额管理模型的比较

10.2 UTXO

UTXO 是用于管理比特币交易的重要概念。

10.2.1 UTXO 概述

UTXO 是 Unspent Transaction Output 的缩写，表示未花费的交易输出。在比特币的交易中使用 UTXO 模型管理交易，如图 10.3 所示。UTXO 的特征在于它不能分割为多个部分。这就如同即使将 1000 日元纸币撕成两半也不可能得到 500 日元纸币。当用户汇款生成交易时，收集到的 UTXO 不应该超出想要支付的金额。如果超出付款金额，余额部分通过将新的 UTXO 链接到新地址来管理。此时，消耗掉的 UTXO 是作为交易输入，而生成的 UTXO 是作为

比特币的区块链使用 UTXO 模型。在计算余额时，从网络中获取未使用的输出数据进行计算。比特币中本身并不存在余额的概念，而在钱包中使用余额这个术语是为了增强用户的便利性。

每种模型都有各自的优点和缺点。基于账户的模型简单、易实现的特点，因此它具有可以通过智能合约而轻松解决复杂处理的优点，以及难以提高匿名性的缺点。

UTXO 模型具有可以提高匿名性并且可以很容易地确保可扩展性的优点。除此之外，UTXO 模型被认为在防止重复付款方面是有效的。其缺点是难以实现，如表 10.2 所示。

表 10.2 基于账户模型和 UTXO 模型

UTXO 模型	
优点	缺点
<ul style="list-style-type: none"> 易于实现 易于解决复杂处理问题 	<ul style="list-style-type: none"> 易于提高匿名性 易于提高可扩展性 易于防止重复付款

10.2.3 尝试获取 UTXO

让我们尝试获取实际的 UTXO。这次，我们将使用 blockchain.info 提供的 API。由于 UTXO 是和地址相关的，因此必须指定一个特定的地址，在这里使用的是 3FkencCiXpSLqD8L79intRNXUgjRoH9sjXa。该地址是由比特币客户端 Bitcoin Core 的开发者 Bitcoin.org 发布。如清单 10.7 所示显示了收集 UTXO 所用到的代码。

清单 10.7 收集 UTXO 的程序

```
输入
import json
import requests

address = "3FkencCiXpSLqD8L79intRNXUgjRoH9sjXa"

res = requests.get("https://blockchain.info/unspent?active=" + address + " - 1")
utxo_list = json.loads(res.text)["unspent_outputs"] - 2
```

```
print("发现了" + str(len(utxo_list)) + "个UTXO!")
for utxo in utxo_list:
    print(utxo["tx_hash"] + ":" + str(utxo["value"]) + " satoshis") - 3
```

在 res = requests.get("https://blockchain.info/unspent?active=" + address) 语句中（**清单 10.7 ①**），数据是通过 blockchain.info 的相应 URL 获得的。因为获取的数据为 JSON 格式，所以在 utxo_list = json.loads(res.text)["unspent_outputs"] 的部分中（**清单 10.7 ②**），将 json 数据中与 UTXO 相对应的部分 unspent_outputs 提取到 utxo_list 中。之后，显示 utxo_list 中存储的数据的数量。另外，通过使用 for 语句，可以为每个项目输出 tx_hash 和 value（**清单 10.7 ③**）。这里，satoshis 是比特币的货币单位，1 BTC = 1 亿 satoshis。执行该代码，将获得如 **清单 10.8** 中所示的输出结果。请注意，根据 UTXO 的状态不同，具体数据项的数量可能也会发生变化。

清单 10.8 清单 10.7 的输出结果

输出
发现了 158 个 UTXO !
8e3a965aac9e2594206c2f13d8254d2e1fa5b2f42b90bf2↑
6de5cf215e:1000 satoshis
f24ba491a636cd5f1f2dd52a1ccb01ea9fc94277418bdb3cc4faa↑
6befd2b2a2:42342 satoshis
ecae0c744fc1516ee864ebaf32191a81b3697d456c99c706c909ef↑
786949ae3d:1380700 satoshis
c6cd2917eaf1b58256f153774af0124f833b8d048f9a769922a5↑
e1bdd0e3e2:5000 satoshis
36c7f1526323477d12ee597d770f23425b92ce6f9d1cd1849cd28b↑
93cac5c6cf:800 satoshis
fab59e9889f1c3717148e394cf60ce590ff32acbad3db3fb586812↑
aadb6c7350:28035 satoshis
80b719320843c434feb3efb442aa2ed8e960e13e63aba2846cd1↑
f97ecfa63a:1101960 satoshis

是基于代币的交易。基于代币的交易是一种在采矿成功时接收到采矿奖励的交易，与其他交易数据不同，它没有输入值。也就是说，这是唯一不消耗UTXO的交易。另外，在网络中所有UTXO的总数就是迄今为止发行的比特币的总数。换句话说，基于代币的交易是在比特币网络中唯一的、增加总UTXO的交易。

让我们尝试查看基于代币的交易。如清单10.9所示显示了基于代币交易的原始数据，是区块高度为111111的、基于代币的交易数据。

清单 10.9 基于代币交易的原始数据

```
8d4dbb63eb502878337b85791b5ebc2960e3b1932d80f3adaa93ab↑
130d5f3895:23816 satoshis
c383cc5a1ed3666f85b13087804244a500a0d34a25ebd62bb2eфе5↑
83b28291c8:56678 satoshis
1dd49710dcfc62af956eee4130322e11f099abf6177c19bcb37ff↑
622736fdbaa9:29029 satoshis
dfa6b3ff4d010db594ede969ff412b93021e4842ced361a425cabef↑
b0b95fc90a:255617 satoshis
af37c4cc9c94749f6224bcff75f545c3da97969906dc75d863515b↑
5fb8fd2095:6340 satoshis
c24e29913b7f8c9defcf4adbedcf2a9a52398726deceb7e8a0bb6↑
61a6e8e1d5:1139417 satoshis
13c9f498c026b334375ff697774fe7299da660d2eb6023d7585537↑
bb734e906f:136977 satoshis
5e9768b2133e1f94d13bffd5c7a0130fe58a31cb06ede16b5/fe8c↑
4d131c1ffb:37600 satoshis
a1c52e63a1b9bf6074d73664f0ce467485dec72640147554e301cc↑
9de8d7eaba:1101 satoshis
db9a72cf96ac48c42cc0726bf4aaea2bd1ba577c9020556b7db380↑
8404b402de:672 satoshis
b9f8478c0e2bd819e448a921c592ff261109d59514fea95fcc0d52↑
f0a18746ac:10716 satoshis
(...略...)
```

该原始数据转换为JSON格式则如清单10.10所示。

清单 10.10 转换为 JSON 格式

```
{
  "txid": "e7c6a5c20318e99e7a2fe7e9c534fae52d402ef6544afdf85a↑
  0a1a22a8d09783a",
  "hash": "e7c6a5c20318e99e7a2fe7e9c534fae52d402ef6544afdf85a↑
  0a1a22a8d09783a",
  "version": 1,
  "size": 134,
  "vsize": 134,
  "weight": 536,
  "locktime": 0,
  "nvin": [
    {
      "coinbase": "04cd2d011b0112" ,
      "sequence": 4294967295
    }
  ]
}
```

这些是与清单10.7所示的地址3FkenCiXpSLqJgjRoH9sjXa关联的UTXO。由于比特币是公有链，因此可以看出，即使是完全第三方的信息，在某种程度上也是公开的。

10.3 基于代币的交易

成功采矿的矿工可以获得采矿奖励。获得奖励的交易称为基于代币的交易。基于代币的交易与我们迄今为止处理的交易不同。交易是由输入和输出链接而成。那么，链锁的起点是如何形成的呢？答案

10.4.1 脚本语言

脚本语言是一种使用称为堆栈的数据结构，并在堆栈中存储数据的语言。这种数据结构存储的数据是 LIFO（后进先出）形式，这就像将积木放在盒子中再拿出来一样。如图 10.5 所示，其具有堆叠的结构。

```

    ],
    "vout": [
        {
            "value": 50.00000000,
            "n": 0,
            "scriptPubKey": {
                "asm": "047b8d5e4b08e71b4e21edc71dc2b5040c0fc6cd1b84↑
46500a95e44fce027d95d7bf74ad3f94e6068f2b94dd4daadcfffc767304↑
4c71876b7e7061531b35b6a0a5 OP_CHECKSIG",
                "hex": "41047b8d5e4b08e71b4e21edc71dc2b5040c0fc6cd1b↑
8446500a95e44fce027d95d7bf74ad3f94e6068f2b94dd4daadcfffc7673↑
044c71876b7e7061531b35b6a0a5ac",
                "reqSigs": 1,
                "type": "pubkey",
                "addresses": [
                    "1Jyv1RNBCf1dQwDTRXNnsJ9Xxpqcr27"
                ]
            }
        }
    ]
}

```

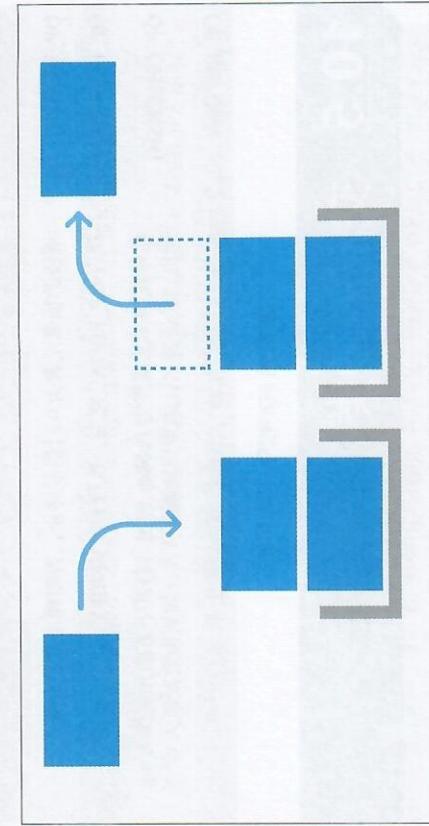


图 10.5 堆栈的示意图

10.4.2 OP_CODE

比特币使用称为 OP_CODE 的编码。OP_CODE 的特征是图灵非完备性和无状态验证。图灵非完备性是指无法进行复杂分支处理的属性，并且无法编写像 Python 那样的复杂程序。另外，**无状态验证**是指不具有任何状态的属性，即无论何人何时在何处执行同一代码，都将获得相同的结果。这些特征，在减少篡改交易数据的可能性的同时，有助于减少程序的漏洞。OP_CODE 定义了如表 10.3 所示的指令及其内容。这些编码用于交易签名和验证。

表 10.3 OP_CODE 典型示例

命 令	内 容
OP_0、OP_FALSE	压入 0
OP_1、OP_TRUE	压入 1
OP_2 ~ OP_n	压入 OP_n 的 n 部分

10.4 脚本语言与 OP_CODE

比特币的交易使用一种称为脚本语言的简单语言进行签名和验证。

命 令	内 容
OP_DUP	压入与堆栈顶部元素相同的数据
OP_HASH160	弹出堆栈的顶部元素，并压入该 Hash160 的哈希值
OP_EQUAL	弹出堆栈的顶部元素及其下方的元素，如果相同则压入 1，否则压入 0
OP_VERIFY	堆栈的顶部元素为 0 时强制终止，否则弹出顶部元素
OP_EQUALVERIFY	执行 OP_EQUAL 之后，执行 OP_VERIFY
OP_CHECKSIG	将堆栈顶部元素作为 Pubkey 弹出，将其下方的元素作为 sig 弹出，验证签名，如果成功则压入 1，如果失败则压入 0
OP_RETURN	将与支付无关的 80 字节数据添加到输出中

10.5 交易的种类

类 型		说 明
P2PKH (Pay-to-Public-Key-Hash)		锁定公开密钥的哈希值
P2PK (Pay-to-Public-Key)		在 Locking Script 配置公开密钥
P2SH (Pay-to-Script-Hash)		使用哈希值简化复杂的脚本

10.5.3 P2PKH

截至 2019 年 9 月，经常可以看到 P2PKH (Pay-to-Public-Key-Hash) 方法将公开密钥的哈希值写在输出中。Locking Script 和 Unlocking Script 如表 10.5 所示。

表 10.5 P2PKH 的脚本

脚本类型	脚本组成
Unlocking Script	<Signature><Public Key>
Locking Script	OP_DUP OP_HASH160 <Public Key Hash> OP_EQUAL OP_CHECKSIG

比特币中交易有几种类型。下面将介绍三个具有代表性的交易类型。

10.5.1 Locking Script 和 Unlocking Script

交易由两种类型的脚本组成：Locking Script (锁定脚本) 和 Unlocking Script (解锁脚本)。Locking Script 是写在输出中的，描述了消耗 UTXO 时必须满足的条件。Unlocking Script 是为了输出可用的脚本。重复进行这种锁定和解锁操作，交易的链接形式便形成了。

实际的脚本是按 Unlocking Script → Locking Script 的顺序执行。执行 Unlocking Script 中指定的解锁条件后，便实现了满足 Locking Script 所指定条件的过程。如果在此过程中某处发生错误，则无法使用 UTXO。

10.5.2 交易的种类概述

在比特币的交易中，根据脚本形式的不同，交易类型也不同。这里介绍的是 P2PKH、P2PK、P2SH 三种类型，如表 10.4 所示。交易存在多种类型的原因有很多。例如，它们的设计目的不同，有的是提高交易的安全性，有的是提高便利性等。

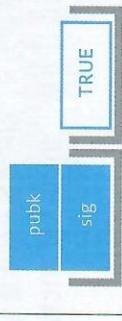
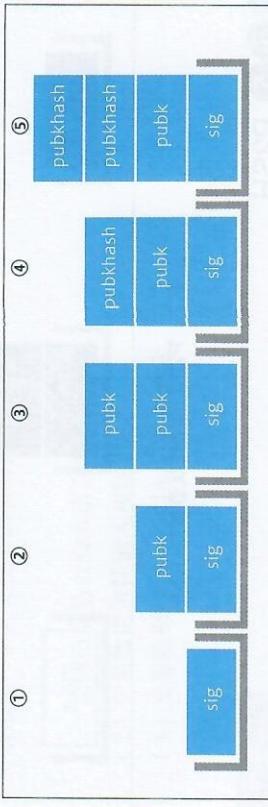


图 10.6 P2PKH 的堆栈状态转换

比起直接将公开密钥配置在 Locking Script 上，这种方式的优点是可以抑制数

的情况下，使用如表 10.7 所示的脚本。

10.5.4 P2PK

P2PK（Pay-to-Public-Key）方式是在 Locking Script 上配置公开密钥的方式，是初期使用的一种古老的方式。Locking Script 和 Unlocking Script 如表 10.6 所示。

表 10.6 P2PK 的脚本

脚本类型	脚本组成
Unlocking Script	<Signature>
Locking Script	<Public Key> OP_CHECKSIG

在该脚本中堆栈的状态转换如图 10.7 所示。

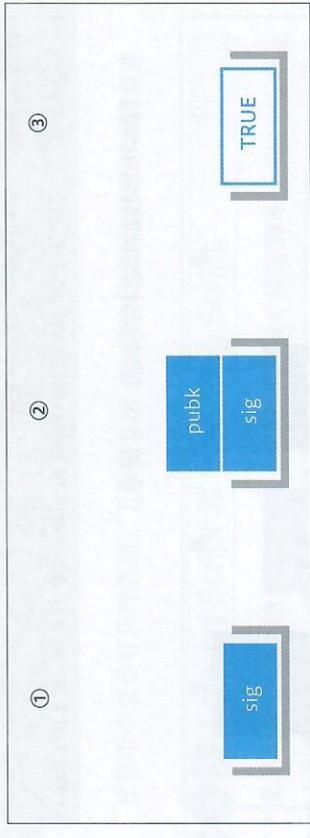


图 10.7 P2PK 的堆栈状态转换

10.5.5 P2SH

P2SH（Pay-to-Script-Hash）方法是一种新方法，可以处理更复杂的交易。要了解 P2SH 方法，需要了解多重签名的概念。多重签名是指使用多个密钥来分别锁定和解锁交易。如果注册的公开密钥的数量为 N，而所需签名的数量为 M，则表示为 M-of-N。多重签名是一种特殊的设计方法，当由多人管理比特币时，可以用更现实的方式对其进行安全管理。

现在，当处理诸如多重签名之类的复杂交易时，脚本将变得非常复杂。如果所需的公开密钥的数量很多，则锁定脚本中的公开密钥及其哈希值将很长。因此，P2SH 通过使用加密哈希函数使复杂的脚本变得简单。例如，在 2-of-3

表 10.7 P2SH 的脚本

脚本类型	脚本组成
Unlocking Script	OP_0 <signature A> <signature B> <Redeem Script>
Locking Script	OP_HASH160 <redeemScriptHash> OP_EQUAL
Redeem Script	2 pubkey A pubkey B pubkey C 3 OP_CHECKMULTISIG

对于 P2SH，首先验证 Redeem Script 和 Locking Script。在堆栈中按从 Redeem Script 到 Locking Script 的顺序执行。如果执行结果正确，则会运行 Unlocking Script 以解锁 Redeem Script。与 P2PKH 和 P2PK 相比，P2SH 中新出现了 Redeem Script，哈希值被配置在 Locking Script 中是它的特征之一。通过删去复杂和多余的部分并对其进行哈希处理，可以使交易过程更加简单。

本章习题

请选择一个有关交易说明的错误选项。

- 问题一
- (1) 比特币交易以脚本语言签名和验证。
 - (2) 交易数据包括输入和输出。
 - (3) 交易数据经过一段时间后将被删除。

问题二

请选择一个关于 UTXO 描述的正确选项。

- (1) 网络上的 UTXO 总数与当时发行的代币总数相同。
- (2) 从安全性的角度来看，所有区块链都使用 UTXO 模型。
- (3) UTXO 可以自由分割。

问题三

请获取地址为 1RCJPNCX9xwlogz5XfKWDjpXF1qmSVxx 的 UTXO。