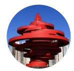
区块链笔记（3）比特币交易的数据和流程

 sept08 发布于 2019-05-25

> 区块链技术只能用来做关于金融交易的应用么？或许先去了解它有关交易的细节，才能看到是否有其它应用的可能。

1 交易的数据模型

1.1 起因

在此之前，我们关于`Bitcoin Core`介绍了许多，以及把它当作工具如何使用，现在我们将进一步来研究下区块链中的数据模型。

> 为什么说将区块和交易当作**数据模型**来理解非常重要?

我的答案是：为了知道如何使用数据。

我们使用区块链应用与网络中的其它节点进行通信、交互以及协作时，可能更关注的是协议。但如果直接去看协议，可能会不容易看得通透，例如在面对一些问题：通过协议传输的数据长什么样？开发自己的区块链应用时，数据是主角，那如何组织和使用它呢？要搞清楚，数据模型这座大山势必要推倒。

另外谈到数据这个话题，开发者可以通过操作码（`Op-code`）的方式向区块中嵌入额外的数据，对此目前社区反应出两种不同的声音，以比特币平台为例，一些人认为比特币区块链如此便包含了许多非金融数据，当区块链不断延展的同时，会对那些不在意这些数据的人的存储空间带来了沉重的负担；另一些人则认为这些非金融数据的存在，可能使区块链在金融领域之外，产生更多的应用可能。

> **Op-code**：来自比特币脚本语言的一些操作码，用于在公钥脚本和签名脚本中推送数据或执行函数。

其实在社区中看到类似的争论还是蛮有意思的，早期时候，人们为了给比特币交易添加备注信息，或其他和交易本身无关的非金融数据，是通过刻录比特币的方式来进行的，就是在不同的交易中，将output里的验证脚本换成其他数据，这会使得`UTXO`数据集不断变大，因为这么做会导致这笔交易里的比特币不能再被花费，又因为整个比特币系统出于速度的考虑，会把所有未被花费的交易（`UTXO`）都存储在内存中，这必然使得网络各节点中包含大量的冗余信息，造成跨节点分类账的维护成本变高。而现在，随着新的改进方案已纳入区块链和操作码中，如`Op-return`。如此协议已经渐趋成熟，`UTXO`数据集就不会夸张的膨胀.

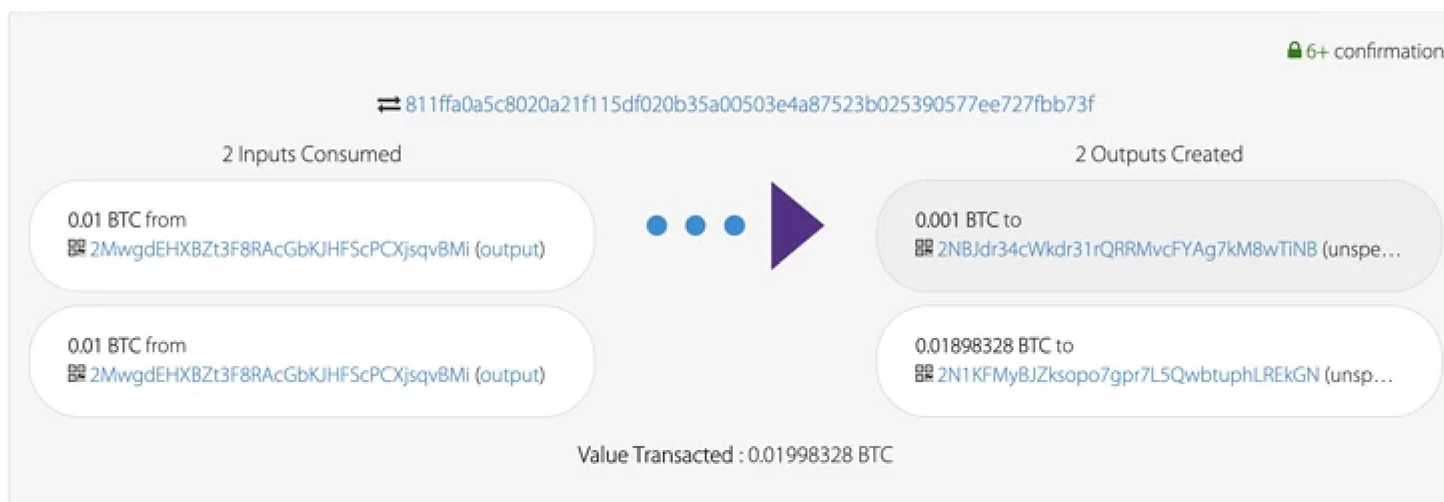
> **UTXO**：即未花费的交易输出（Unspent Transaction Outputs），它是比特币交易生成及验证的一个核心概念。交易构成了一组链式结构，所有合法的比特币交易都可以追溯到前向一个或多个交易的输出，这些链条的源头都是挖矿奖励，末尾则是当前未花费的交易输出。另外值得提的一点是，在比特币钱包当中，我们都可以看到账户余额，但在这个账户余额的概念与我们所熟知的银行账户余额有着巨大的不同，其实站在`UTXO`交易模型上看，并没有什么所谓一个一个的比特币，有的只是`UTXO`。当我们说张三拥有10个比特币的时候，我们实际上是在说，当前区块链账本中，有若干笔交易的`UTXO`项的收款人写的是张三的地址，而这些`UTXO`项的数额总和是10。比特币钱包中所看到的账户余额，实际上则是钱包通过扫描区块链并聚合所有属于该用户的`UTXO`计算得来的。
> **Op-return**：本质上讲，OP_RETURN是一个脚本操作码，是专门被设计出来承载额外的交易信息的。它的作用就像我们在日常转账过程中的备注信息。通过它发送的数据会和我们进行的比特币交易一样，永久保存在比特币区块链的区块中。

1.2 交易的输入和输出

不论你面对的是哪种区块链应用，交易都是区块链系统中最重要的部分。你可以把交易理解为组成区块链宇宙的原子，正如原子是组成所有生命的基础，交易则是组成数据块的单位。你可能已经注意到了，比特币区块链上所做的任何事情都是，为了确保一笔交易能否被创建，并在网络中传播和验证，以及最终添加到区块链上。当然搞清楚这些具体细节，还是为了以后能够创建自己的区块链应用。所以现在还是一步一步来，先回顾下交易是如何运作的，以及它的输入和输出，这对后面讨论交易的数据模型来说很重要。

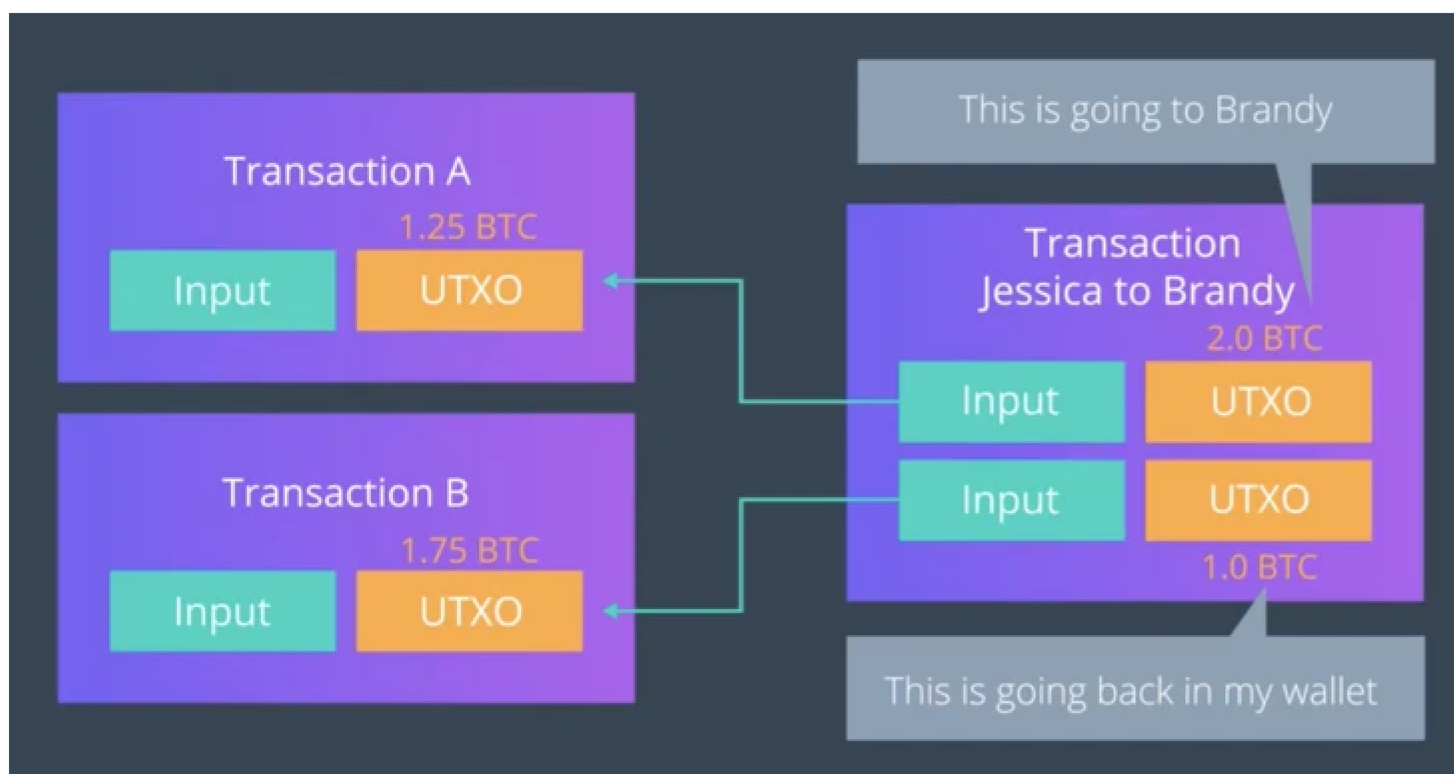
> **交易**描述的是一笔资金从它的原始所有者（`input`）向即将所有者（`output`）价值转化的数据结构

以下交易详情是使用之前我们介绍过的站点，查看比特币测试链上的一笔交易：



从图中显而易见的是，有两笔为0.01BTC的输入，参与了一次0.001BTC的转账后，又退回给原所有者0.019BTC，基于此我想问的是：这些输入从何而来，产生的新输出又去向何处？

> 一个交易的输入，都来自与另一个交易的未花费输出（**UTXO**）。



在交易发生时有获取账户余额的需求，都是通过统计整个区块链上，该钱包地址关联的所有**UTXO**（未花费交易输出）上的比特币数量来完成的。所以并不存在存储一个账户余额的字段，或者一个比特币的地址。

1.3 数据模型

这一小节我们来看交易的信息在数据模型中是如何存储的。如果要求网络返回一个原始交易信息给我们，所得到的可能是像下面这样的信息：

```
0100000001f3f6a909f8521adb57d898d2985834e632374e770fd9e2b98656f1bf1fdfd427010000006b48304502203a776322ebf
8eb8b58cc6ced4f2574f4c73aa664edce0b0022690f2f6f47c521022100b82353305988cb0ebd443089a173ceec93fe4dbfe98d74
419ecc84a6a698e31d012103c5c1bc61f60ce3d6223a63cedbece03b12ef9f0068f2f3c4a7e7f06c523c3664ffffffff0260e3160
000000001976a914977ae6e32349b99b72196cb62b5ef37329ed81b488ac063d1000000000001976a914f76bc4190f3d8e2315e5
c11c59cfc8be9df747e388ac00000000
```

这是一条还未解码成JSON对象的十六进制数据。虽然确实不是很容易看的懂，但其实组织的还是很有条理的。以上面这条信息为例，从起始位开始，一条交易一般包含如下内容：

1. 比特币的版本（Version）：`01000000`
2. 交易的输入数量（Input Count）：`01`
3. 交易的输入信息（Input Info）：`f3f6a909f8521adb57d898d2985834e632374e770fd9e2b98656f1bf1fdfd427010000006b48304502203a776322ebf8eb8b58cc6ced4f2574f4c73aa664edce0b0022690f2f6f47c521022100b82353305988cb0ebd443089a173ceec93fe4dbfe98d74419ecc84a6a698e31d012103c5c1bc61f60ce3d6223a63cedbece03b12ef9f0068f2f3c4a7e7f06c523c3664ffffffff`
4. 交易的输出数量（Output Count）：`02`
5. 交易的输出信息（Output Info）：`60e31600000000001976a914977ae6e32349b99b72196cb62b5ef37329ed81b488ac063d1000000000001976a914f76bc4190f3d8e2315e5c11c59cfc8be9df747e388ac`

6. 锁定时间（loctime）：`00000000`。它表示该条交易最早被确认后，写入的最早区块或最早被确认写入的时间：
    - 若该字段非零，且<5亿，则表示该条交易最早被写入的区块的区块号。
    - 若>5亿，则表示该条交易最早被写入区块的时间。
    - 若为零，则表示该条交易立即被写入区块。

其中在交易的输入信息和输出信息中，还分别包含了一小段用以验证该次交易是否有效地指令脚本：即输入信息中的解锁脚本（UnLocking script）和输出信息中的锁定脚本（Locking script）。

> 这里的脚本（script），指的是记录在每条交易中的一系列指令字符，执行用于验证交易是否有效及比特币能否发出。而名称与之类似的比特币脚本语句
> （Bitcoin Script）是一种基于栈的简单轻量级的语句，被设计用来能通用于一系列硬件平台上做相关运算的指令。我们可以在栈中存储数字或数据常量，并使用一系列前缀为`OP_`的指令（Opcode）对数据进行操作。例如通过`OP_ADD`将栈中的两个数据进行相加，通过`OP_EQUAL`来检查栈顶的两个元素是否相等，`OP_DUP`复制栈顶的数据等等，总共大概有80多个指令，详见[Opcodes]的维基百科。

接下来我们通过一条简单的算数运算指令来具体观察上面提到的三个概念：解锁脚本、锁定脚本和包含`Opcodes`指令的比特币脚本语句，算数指令如下：

```
2 6 OP_ADD 8 OP_EQUAL
```

比特币脚本语句的执行顺序是从左向右的，并且是基于栈结构的，那么这条语句的执行步骤就应当是：

1. 数字2入栈；
2. 数字6入栈；
3. 执行`OP_ADD`：数字6和2依次出栈后，相加所得的结果（8）再入栈；
4. 数字8入栈
5. 执行`OP_EQUAL`：数字8和8依次出栈后，进行相等比较，所得的结果（`True`）再入栈

其中我们可以将`6 OP_ADD 8 OP_EQUAL`这部分视为锁定脚本，它需要满足使其最终结果为`True`的解锁脚本（2），才能完成算数验证。也就是说如果用这条语句来验证交易的有效性，那么所有知道数字2能满足条件的解锁语句，都可使其生效。

> 对于比特币脚本语言有两个特性：
>
> - 无流程控制：语句简单，不存在循环和条件控制，好处是不用担心死循环之类的阻塞性错误；缺点是不够灵活。
> - 无状态：在执行过程前后，不保存任何关于状态的值，好处是安全，不论在哪个平台上执行相同的语句都会得到相同的答案；不足是比较简单。
>
> 任何实现方式的特点，都有其长短优劣，在做整体方案架构的考量时，应谨慎根据业务场景进行选取。

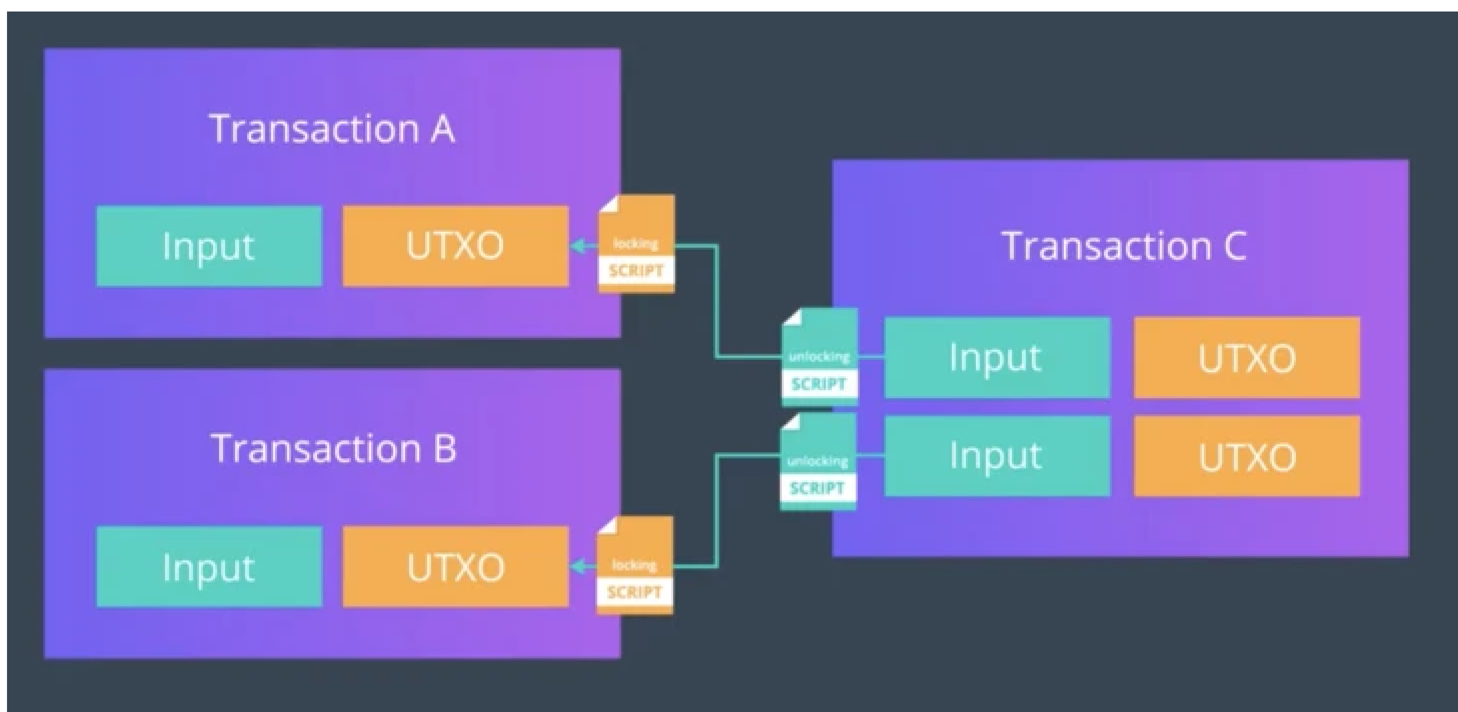
而在实际情况中，我们验证交易有效性所使用得解锁脚本（UnLocking script）和锁定脚本（Locking script）构成的比特币脚本语句是如下的结构：

```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

其中对应于解锁脚本（UnLocking script）和锁定脚本（Locking script）的部分分别是：

- UnLocking script：`<sig> <pubKey>`
- Locking script：`OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

忘了说清楚一点，验证交易发生的有效性，并不是用同一个交易的解锁脚本（UnLocking script）和锁定脚本（Locking script）进行验证。而是用当前进行交易的解锁脚本，与该输入回溯的`UTXO`中的锁定脚本进行验证，而当前交易的锁定脚本则是用来，和未来将要发生的交易中的解锁脚本进行验证。具体验证关系如下图：



交易的有效性验证的工作原理其实很简单，就是利用了非对称加密，在解锁脚本中，包含了钱包所有者用私钥生成的签名。因为只有钱包所有者才有交易权，才能生成判断交易有效地解锁脚本。

具体拆分上面的原始交易数据如下图：

Version		01000000
Input Count		01
Input Info	Previous output hash (reversed)	f3f6a909f8521adb57d898d2985834e632374e770fd9e2b98656f1bf1fdfd42701
	Previous output index	000000
	Script Size (bytes)	6b
	scriptSig	48304502203a776322ebf8eb8b58cc6ced4f2574f4c73aa664edce0b0022690f2f6f47c521022100b82353305988cb0ebd443089a173ceec93fe4dbfe98d74419ecc84a6a698e31d012103c5c1bc61f60ce3d6223a63cedbece03b12ef9f0068f2f3c4a7e7f06c523c3664
	sequence	ffffffff
Output Count		02
Output Info	Value	60e3160000000000
	Script Size (bytes)	19
	scriptPubKey	76a914977ae6e32349b99b72196cb62b5ef37329ed81b488ac063d1000000000001976a914f76bc4190f3d8e2315e5c11c59cfc8be9df747e388ac
Lock time		00000000

其中我们将输入信息细化为如下部分：

- `Previous output hash`：所有的输入都可以回溯到一个输出，即上一笔交易所产生的UTXO。
- `Previous output index`：可能一笔交易会包含多项UTXO，这项便是指定多个UTXO的索引，其中第一个UTXO从0开始算。
- `Script Size(bytes)`：表示解锁脚本的字节数大小。
- `scriptSig`：上文谈到的解锁脚本
- `Sequence`：这目前是比特币废弃的一个属性位，默认设为`ffffffff`。

而输出信息也可细化出如下部分：

- `Amount`：比特币输出的数量，按比特币最小单位（Satoshis）计算，10^8 Satoshis = 1 Bitcoin.
- `Script Size(bytes)`： 表示锁定脚本的字节数大小。
- `scriptPubKey`：上文谈到的解锁脚本。

2 创建交易

通过比特币钱包的GUI工具，虽然能够完成比特币区块链生命周期中的基本操作，但存在一些局限性，所以接下来为了更深入的了解比特币区块链交易的细节，我们将使用调试控制台来创建一个交易，具体步骤如下：

1. 在比特币钱包中查看所有的UTXO
2. 查看一个特定UTXO的细节
3. 创建一个原始交易
4. 解码该原始交易

5. 对该原始交易进行签名
6. 将这个交易提交到网络
7. 通过TxID查询所创建的交易

2.1 查看UTXO

我们可以通过在上一节介绍的比特币钱包的调试窗口（Help-Debug Window）中，查看本钱包所有的UTXO，查询命令为：`listunspent`。发现查询结果是由一个个UTXO对象构成的数组组成的，截取其中一个UTXO如下所示：

```
[
  ...
  {
    "txid": "811ffa0a5c8020a21f115df020b35a00503e4a87523b025390577ee727fbb73f", // 交易Id
    "vout": 1, // 输出序号
    "address": "2N1KFMyBJZksopo7gpr7L5QwbtuphLREkGN", // 地址
    "redeemScript": "001462fab42642cbfe84c69a9e17fcb6c1ae27f63748", // 赎回脚本
    "scriptPubKey": "a9145883d125a1bb6db07e886bb167d966013f407c4487", // 公钥脚本
    "amount": 0.01898328, // 可用金额
    "confirmations": 26738, // 确认次数
    "spendable": true, // 当前钱包是否拥有私钥，以便能够消费该UTXO
    "solvable": true, // 是否可用，缺少秘钥时忽略
    "safe": true // 未经确认的交易将被认为是不安全的
  },
  ...
]
```

2.2 查看一个UTXO详情

这步我们使用命令：`gettxout`来查询一个未花费交易的详情，该命令接收三个参数：交易ID、未花费输出的序号（从0开始）、一个可选的布尔值用来控制是否显示内存池中还未验证的输出。
复制上一步中的交易ID的查询命令如下：

```
gettxout 811ffa0a5c8020a21f115df020b35a00503e4a87523b025390577ee727fbb73f 0
```

运行后得到的结果如下：

```
{
  "bestblock": "00000000000000a88e2e39c56235eb61eaf40fca8273e31d5ce49a4d8577d51f",
  "confirmations": 26842, // 验证次数
  "value": 0.00100000, // 交易金额（单位是BTC）
  "scriptPubKey": { // 解锁脚本
    "asm": "OP_HASH160 c6176d6f78b0205a83bf4bbc516a23dc00a4ca64 OP_EQUAL", // 汇编格式（assembly）
    "hex": "a914c6176d6f78b0205a83bf4bbc516a23dc00a4ca6487", // 十六进制格式
    "reqSigs": 1, // 所需的签名数
    "type": "scripthash", // 加密类型
    "addresses": [ // 收款地址列表
      "2NBJdr34cWkdr31rQRRMvcFYAg7kM8wTiNB"
    ]
  },
  "coinbase": false
}
```

2.3 创建一个原始交易

使用命令：createrawtransaction，创建一个未签名的序列化交易，该交易并不会存储在钱包或传输到网络。需要两个传参：第一个是前一个输出的引用，第二个是P2PKH或P2SH标准的收款地址及收款数量。创建命令示意如下：

```
createrawtransaction '[{"txid":"811ffa0a5c8020a21f115df020b35a00503e4a87523b025390577ee727fbb73f","vout":1}]' '{"2NBn87R8AAwtXUNmmFULvDhmPyeka1X7rRD":0.001, "2NBJdr34cWkdr31rQRRMvcFYAg7kM8wTiNB": 0.001}'
```

我执行后得到的输出：

```
020000000013fb7fb27e77e579053023b52874a3e50005ab320f05d111fa220805c0afa1f810100000000ffffffff02a08601000000000017a914cb4a40c6ccaf652cc9a6459047494359c3ff25d787a08601000000000017a914c6176d6f78b0205a83bf4bbc516a23dc00a4ca648700000000
```

2.4 解码

上一步所创建原始交易的输出，是一串十六进制字符串，显然没有什么可读性。为了确认我们所创建的正确性，我们需要将其解码为可读的JSON格式，使用到的命令是decoderawtransaction，执行如下：

```
decoderawtransaction
020000000013fb7fb27e77e579053023b52874a3e50005ab320f05d111fa220805c0afa1f810100000000ffffffff02a08601000000000017a914cb4a40c6ccaf652cc9a6459047494359c3ff25d787a08601000000000017a914c6176d6f78b0205a83bf4bbc516a23dc00a4ca648700000000
```

输出结果如下：

```
{
  "txid": "8af75c03ca2e7e84135b2809f73e75d758cfc5b72c1e51ae18b770baef844b54",
  "hash": "8af75c03ca2e7e84135b2809f73e75d758cfc5b72c1e51ae18b770baef844b54",
  "version": 2,
  "size": 115,
  "vsize": 115,
  "weight": 460,
  "locktime": 0,
  "vin": [
    {
      "txid": "811ffa0a5c8020a21f115df020b35a00503e4a87523b025390577ee727fbb73f",
      "vout": 1,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.00100000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_HASH160 cb4a40c6ccaf652cc9a6459047494359c3ff25d7 OP_EQUAL",
        "hex": "a914cb4a40c6ccaf652cc9a6459047494359c3ff25d787",
```

2.5 签名

从上面可读性更好的原始交易信息中，看到交易输入的scriptSig字段为空，这是因为我们还没有为这个签名，证明我们拥有对UTXO的使用权。接下来使用命令signrawtransactionwithwallet进行签名：

```
signrawtransactionwithwallet
020000000013fb7fb27e77e579053023b52874a3e50005ab320f05d111fa220805c0afa1f810100000000ffffffff02a08601000000000017a914cb4a40c6ccaf652cc9a6459047494359c3ff25d787a08601000000000017a914c6176d6f78b0205a83bf4bbc516a23dc00a4ca648700000000
```

签名成功的输出结果如下：

```
{
  "hex":
"020000000001013fb7fb27e77e579053023b52874a3e50005ab320f05d111fa220805c0afa1f81010000001716001462fab42642cbf

  "complete": true
}
```

然后对签名后的输出进行JSON解码，会发现输入部分多了些内容：

```
{
...
"vin": [{
  "txid": "811ffa0a5c8020a21f115df020b35a00503e4a87523b025390577ee727fbb73f",
  "vout": 1,
  "scriptSig": {
    "asm": "001462fab42642cbfe84c69a9e17fcb6c1ae27f63748",
    "hex": "16001462fab42642cbfe84c69a9e17fcb6c1ae27f63748"
  },
  "txinwitness":
["304402207fbd59f6e806dc1aab5f602b796dc2ecfa96f0e018c7fe4ecc7dcf190e0619f10220168dffa1d5bd5876518530c72fd3bf
 "03959e3af1e6ddb01d6ac54966cda59464ab27fcaf34b0dca6df02f75d3df76688"],
  "sequence": 4294967295
}],
...
}
```

2.6 将签名后的交易推送至网络

使用命令sendrawtransaction将已签名的交易推送至网络。

```
sendrawtransaction
020000000001013fb7fb27e77e579053023b52874a3e50005ab320f05d111fa220805c0afa1f81010000001716001462fab42642c
bfe84c69a9e17fcb6c1ae27f63748ffffffff02a086010000000000017a914cb4a40c6ccaf652cc9a64590474494359c3ff25d787a0
8601000000000017a914c6176d6f78b0205a83bf4bbc516a23dc00a4ca64870247304402207fbd59f6e806dc1aab5f602b796dc2e
cfa96f0e018c7fe4ecc7dcf190e0619f10220168dffa1d5bd5876518530c72fd3bff59337050949c9732dabcfaeef7533de440121
03959e3af1e6ddb01d6ac54966cda59464ab27fcaf34b0dca6df02f75d3df7668800000000
```

执行后返回的结果是交易ID的十六进制值：

```
24cd5619a366ad6a3a34a29766fd5f82c39657bc15dcfdcd4d7363a65f401c8b
```

2.7 查看交易详情

至此整个交易的声明周期就完成了，我们可以通过gettransaction来查看，上一步完成交易的详情：

```
gettransaction 24cd5619a366ad6a3a34a29766fd5f82c39657bc15dcfdcd4d7363a65f401c8b
```

得到详情结果如下：

```
{
  "amount": -0.00200000,
  "fee": -0.01698328,
  "confirmations": 1,
  "blockhash": "000000000000006715d295c34b2896d0c28f67a092869610200684e45fdd3ad9",
  "blockindex": 1,
  "blocktime": 1558762656,
  "txid": "24cd5619a366ad6a3a34a29766fd5f82c39657bc15dcfdcd4d7363a65f401c8b",
  "walletconflicts": [
  ],
  "time": 1558762586,
  "timereceived": 1558762586,
  "bip125-replaceable": "no",
  "details": [
    {
      "address": "2NBn87R8AAwtXUNmmFULvDhmPyeka1X7rRD",
      "category": "send",
      "amount": -0.00100000,
      "vout": 0,
      "fee": -0.01698328,
      "abandoned": false
    },
    {
      "address": "2NBJdr34cWkdr31rQRRMvcFYAg7kM8wTiNB",
      "category": "send",
      "amount": -0.00100000,
```


区块链　比特币

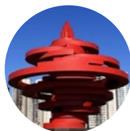
阅读 2.4k • 发布于 2019-05-25

赞 5　收藏 1　分享

本作品系原创，采用《署名-非商业性使用-禁止演绎 4.0 国际》许可协议

有赞美业前端团队

关注专栏



sept08

230 声望　7 粉丝

关注作者

0 条评论

得票数　最新

撰写评论 ...

提交评论

你知道吗?

在存储日期中的年份的时候，请使用四位数字。

注册登录

继续阅读

一个比特币交易的完整流程。

作为加密货币用户，你需要熟悉交易雏形——为了你对这种不断发展的创新有信心，以及作为理解新兴多签名交易和合约的基础，...

malakashi • 阅读 12.5k • 2 赞

JavaScript编写自己的比特币交易代码

今天我们将编写第一个比特币交易代码。为了实现这一目标，我们将使用名为bitcore的JavaScript库。JavaScript是最流行的现代...

malakashi • 阅读 1.8k

币圈小蝶：币圈炒币，一文看懂比特币交易的全过程

学习区块链，肯定离不开比特币，我们的系列课程第一部分聊聊比特币的那些事。真正能让我们长期持有一项资产的前提是我们需...

已注销 • 阅读 14

比特币脚本及交易分析 - 智能合约雏形

大家都有转过账，每笔交易是这样的：张三账上减￥200，李四账上加￥200。在比特币区块链中，交易不是这么简单，交易实际...

已注销 • 阅读 465

比特币:交易的数据结构

比特币协议中最重要的部分就是交易，比特币协议其他的部分也都是为了确保交易的生成、广播、验证和打包而实现的。本文内容...

姜家志 • 阅读 3.6k

币安交易所网址是多少？比特币如何交易的？比特币到底是什么？什么是比特币的底层逻辑？

很多小伙伴都听说了比特币，但是不知道怎么购买？交易比特币当然选择，全球最大的交易所！币安交易所。币安拥有全球领先的...

币安文博 • 阅读 49

比特币脚本及交易分析 - 智能合约雏形

大家都有转过账，每笔交易是这样的：张三账上减￥200，李四账上加￥200。在比特币区块链中，交易不是这么简单，交易实际...

Tiny熊 • 阅读 1.5k

一文看懂怎样用 Python 创建比特币交易

比特币价格的上上下下，始终撩动着每一个人无比关切的小心脏。从去年初的 800 美元左右，飞涨到去年底到 19783.21 美元最高...

链客 • 阅读 292

Copyright © 2011-2021 SegmentFault. 当前呈现版本 21.06.03
浙ICP备15005796号-2 浙公网安备33010602002000号 ICP 经营许可 浙B2-20201554
杭州堆栈科技有限公司版权所有