目录

返回目录

1. 阅读说明

*绿色斜体*代表个人的思考理解，*黄色斜体*代表阅读理解过程中的疑问，红色正体代表关键重要信息，下划线代表次关键重要信息

第 2 章到第 6 章基本完成了比特币网络的所有要素，后面章节为相应的细节补充(*尽可能理解这些细节与前面 5 章构建比特币网络的关系*)

2. 比特币是什么

狭义：比特币是一种基于分布式网络的数字货币；广义：比特币系统（广义的比特币）则是用来构建这种数字货币的网络系统，是一个分布式的点对点网络系统

比特币系统与银行一样拥有账本，不同银行由单一的组织负责记录，比特币的记账由所有运行系统的人（即节点，可以简单理解为一台电脑）共同参与记录，每个节点都保存（同步）一份完整的账本。同时使用简单多数原则，来保证账本的一致性。

3. 区块链记账原理

1. 哈希函数：

Hash(原始信息) = 摘要信息，例：Hash(张三借给李四 100 万，利息 1%，1 年后还本息 …..) = AC4635D34DEF

哈希函数有几个特点：A. 原始信息任何微小的变化都会哈希出面目全非的摘要信息 B. 从摘要信息无法逆向推算出原始信息

2. 区块链记账方法

区块链在记账是会把账页信息（包含序号、记账时间、交易记录）作为原始信息进行 Hash，得到一个 Hash 值，如：787635ACD，用函数表示为：

```
1. Hash(序号 0、记账时间、交易记录) = 787635ACD
```

账页信息和 Hash 值组合在一起就构成了第一个区块，比特币系统里约 10 分钟记一次账，即每个区块生成时间大概间隔 10 分钟。在记第 2 个账页的时候，会把上一个块的 Hash 值和当前的账页信息一起作为原始信息进行 Hash，即：

```
1. Hash(上一个 Hash 值、序号 1、记账时间、交易记录) = 456635BCD
```

所有这些区块组合起来就形成了区块链，这样的区块链就构成了一个便于验证（只要验证最后一个区块的 Hash 值就相当于验证了整个账本），不可更改（任何一个交易信息的更改，会让所有之后的区块的 Hash 值发生变化*(对应哈希函数 A 特点)*，这样在验证时就无法通过）的总账本

4. 比特币所有权及隐私问题-非对称加密应用

比特币系统是如何确定某个账户的比特币是属于谁的？谁可以支付这个账户比特币？

1. 匿名账户

比特币的账户是用地址来表示，转账是把比特币从一个地址转移到另一个地址，交易信息如：

```
{
    "付款地址": "2A39CBa2390FDe"
    "收款地址": "AAC9CBa239aFcc"
    "金额": "0.2btc"
}
```

支付和所有权实际是同一个问题，如果此比特币只有我可以用来支付，那么说明我拥有所有权

2. 地址与私钥

比特币的解决方案是，谁拥有某个地址的私钥(可以简单的把私钥当作密码)，谁就能用这个地址进行支付。比特币地址和私钥是一个非对称的关系，私钥经过一系列运算（其中有两次 Hash）之后，可以得到地址，但是无法从地址反推得到私钥。银行系统银行账号和密码是完全独立的，无法互相推导，转出时需要同时验证账号和密码。

```
地址:  2A39CBa2390FDe
私钥:  sdgHsdniNIhdsgaKIhkgnakgaihNKHIskdgal

Hash(Hash(fun(sdgHsdniNIhdsgaKIhkgnakgaihNKHIskdgal)))  ->
2A39CBa2390FDe
```

3. 非对称加密技术

问题演变为：如何证明你拥有某个地址的私钥（在不泄漏私钥的情况下）

- 对交易信息进行签名

A. 交易信息进行 Hash 运算得到摘要信息(Hash 值)：

```
hash('
    {"付款地址": "2A39CBa2390FDe",
    "收款地址": "AAC9CBa239aFcc",
    "金额": "0.2btc"
    }') -> 8aDB23CDEA6
```

B. 用私钥对交易摘要进行签名（付款方在安全的环境下进行，以避免私钥泄密）

```
1. #参数 1 为交易摘要
2. #参数 2 为私钥
3. #返回签名信息
4. sign("8aDB23CDEA6", "J78sknJhidhLIqdngalket") -> "3cdferdadgadg"
```

签名后的信息类似于一张签名支票，支票上包含交易信息以及资产所有人的签名

- 广播

签名运算之后，付款节点就开始在全网进行广播：我支付了 0.2btc 到 AAC9CBa239aFcc,签名信息是 3cdferdadgadg；广播过程实际上是发信息到相连的其它节点，其它节点在验证通过后再转发到与之相连的节点；广播的信息包含了交易原始信息和签名信息。

- 验证

其它节点在收到广播信息之后，会验证签名信息是不是付款方用私钥对交易原始信息签名产生的，如果验证通过说明确实是付款方本人发出的交易，说明交易有效，才会记录到账本中去。验证过程实际是签名过程的逆运算：

```
1. #参数 1 为签名信息
2. #参数 2 为付款方地址
3. #返回交易摘要
4. verify("3cdferdadgadg", "2A39CBa2390FDe") -> "8aDB23CDEA6"
```

大家可以理解为付款地址为公钥，签名过程即为用私钥对交易摘要的加密过程，验证过程为用公钥解密的过程(为方便大家理解，严格来讲是不准确的)

- 补充

比特币系统使用了椭圆曲线签名算法，算法的私钥由 32 个字节随机数组成，通过私钥可以计算出公钥，公钥经过一序列哈希算法和编码算法得到比特币地址，地址也可以理解为公钥的摘要

不清楚上述过程是如何回答问题的：签名过程：交易信息(包含收付款地址)+私钥 -> 签名信息，验证过程：签名信息+付款方地址 -> 交易信息，验证过程怎么就证明了你拥有某个地址的私钥？（可能需要了解公私钥的原理）

5. 比特币如何挖矿（挖矿原理）-工作量证明

记账是把交易记录、交易时间、账本序号、上一个 Hash 值等信息计算 Hash 打包的过程。所有的计算和存贮是需要消耗计算机资源的，既然要付出成本，那节点为什么还要参与记账呢？在中本聪（比特币之父）的设计里，完成记账的节点可以获得系统给与的一定数量的比特币奖励，这个奖励的过程也就是比特币的发行过程，因此大家形象的把记账称为“挖矿”*(区块链与比特币不是同一事物，*

即便账本(区块链)用于记录其他信息而不是交易\资产信息，比如记录不同人的身高体重，依然要消耗计算机资源，系统依然要提供奖励)

1. 记账工作

记账可以凭空增加一定数量的比特币，因此就出现大家争相记账，一起记账就会出现记账不一致的问题，比特币系统引入工作量证明来解决这个问题，规则如下：

- 一段时间内（10 分钟左右，具体时间会与密码学难题难度相互影响）只有一人可以记账成功
- 通过解决密码学难题（即工作量证明）竞争获得唯一记账权(*假设该难题在规定时间内没解决怎么办？*)
- 其他节点复制记账结果

进行工作量证明之前，记账节点会做进行其他准备工作，可参考《比特币如何挖矿（挖矿原理）-工作量证明》

2. 工作量证明

每次记账会把上一个块的 Hash 值和当前的账页信息一起作为原始信息进行 Hash，为了保证 10 分钟左右只有一个人可以记账，就必须要提高记账的难度，使得 Hash 的结果必须以若干个 0 开头(*此即是需满足的记账要求，满足该要求则可获得记账权*)。提高记账难度的设计则是：在进行 Hash 时引入一个随机数变量(*此时找出随机数来满足前述记账要求的难度就增大了*)

```
1. Hash(上一个 Hash 值，交易记录集) = 456635BCD #这样记账很容易，大家都可以记
2. Hash(上一个 Hash 值，交易记录集，随机数) = 0000aFD635BCD
```

3. 验证

此验证与第 4 点比特币所有权验证不同，但究竟区别是什么？此处的验证是验证新区块里的交易信息是否是正确的，是验证已写入了新区块的信息，验证的目的是看其是否可以加入总账本，而前述的比特币所有权的验证是验证当前在进行的交易信息，验证的目的是看其是否可以加入新区块

节点成功找到满足的 Hash 值之后，会马上对全网进行广播打包区块，网络的节点收到广播打包区块，会立刻对其进行验证。如果验证通过，则表明已经有节点成功解迷，自己就不再竞争当前区块打包，而是选择接受这个区块，记录到自己的账本中，然后进行下一个区块的竞争猜谜。网络中只有最快解谜的区块，才会添加到账本中，其他的节点进行复制，这样就保证了整个账本的唯一性。

假如节点有任何的作弊行为，都会导致网络的节点验证不通过(*为什么？验证过程究竟是如何进行的，既然节点拥有了该区块的记账权，别的节点都以自己的区块数据为准，按道理自己篡改该数据不会有问题，反正是自己说了算*)，直接丢弃其打包的区块，这个区块就无法记录到总账本中，作弊的

节点耗费的成本就白费了，因此在巨大的挖矿成本下，也使得矿工自觉自愿的遵守比特币系统的共识协议，也就确保了整个系统的安全。

6. 比特币如何达成共识 - 最长链的选择

比特币几乎所有的完整节点都有一份公共总帐本，那么大家如何达成共识：确认哪一份才是公认权威的总账本呢?

1. 去中心化共识

比特币的共识由所有节点的 4 个独立过程相互作用而产生：

- 每个节点（挖矿节点）依据标准对每个交易进行独立验证(*即 4.3 中的验证*)
- 挖矿节点通过完成工作量证明(*节点抢到记账权*)，将交易记录独立打包进新区块
- 每个节点独立的对新区块进行校验(*即 5.3 中的验证*)并组装进区块链
- 每个节点对区块链进行独立选择，在工作量证明机制下选择累计工作量最大的区块链

2. 最长链的选择

现详细说明第 4 个过程。在一般情况下，也是包含最多区块的那个链称为主链，每一个（挖矿）节点总是选择并尝试延长主链

- 分叉

当有两名矿工在几乎在相同的时间内(*是否有可能存在 3 名矿工同时解谜成功*)，各自都算得了工作量证明解，便立即传播自己的“获胜”区块到网络中，先是传播给邻近的节点而后传播到整个网络。每个收到有效区块的节点都会将其并入并延长区块链。

当这个两个区块传播时，一些节点首先收到#3458A，一些节点首先收到#3458B，这两个候选区块（通常这两个候选区块会包含几乎相同的交易）都是主链的延伸，分叉就会产生，这时分叉出有竞争关系的两条链，如图：

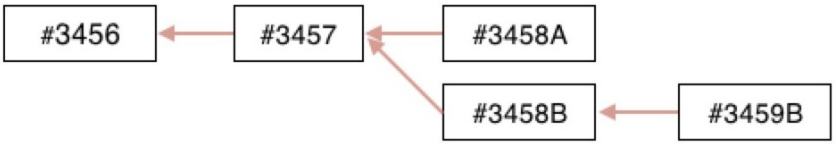
两个块都收到的节点，会把其中有更多工作量(*既然能选出工作量更多的，直接作为主链然后通知全网选这个节点就可以了，为何还要留个备用链*)的一条会继续作为主链，另一条作为备用链保存（保存是因为备用链将来可能会超过主链难度称为新主链）(*是否存在只收到一个块的节点？如果所有的节点最终都能收到两个块，只是先后顺序不同，既然已经识别出哪个块存在更多工作量，直接抛*

弃另一个即可，何来主链备用链的区别，保留备用链是否说明存在某些节点只收到一个块？根据下面可能出现孤块的情形，说明存在节点收到 0 个、1 个、2 个区块的可能)

- 分叉解决

收到#3458A 的（挖矿）节点，会立刻以这个区块为父区块来产生新的候选区块，并尝试寻找这个候选区块的工作量证明解。同样地，接受#3458B 区块的节点会以这个区块为链的顶点开始生成新块，延长这个链（下面称为 B 链）。

这时总会有一方抢先发现工作量证明解并将其传播出去(*这个抢先不知道究竟有什么意义，哪怕你抢先了，由于广播问题依然会导致某些节点在接受你这个区块的时候已先接受了别的区块*)，假设以#3458B 为父区块的工作量证明首先解出，如图:



当原本以#3458A 为父区块求解的节点在收到#3458B，#3459B 之后，会立刻将 B 链作为主链（因为#3458A 为顶点的链已经不是最长链了）继续挖矿。

节点也有可能先收到#3459B，再收到#3458B，收到#3459B 时，会被认为是“孤块“（因为还找不到#3459B 的父块#3458B）保存在孤块池中，一旦收到父块#3458B 时，节点就会将孤块从孤块池中取出，并且连接到它的父区块，让它作为区块链的一部分。

比特币将区块间隔设计为 10 分钟，是在更快速的交易确认和更低的分叉概率间作出的妥协。更短的区块产生间隔会让交易确认更快地完成，也会导致更加频繁地区块链分叉。与之相对地，长的间隔会减少分叉数量，却会导致更长的确认时间

最长链机制非常不清晰，主要在于不明白获胜区块在传播中出现的问题，比如说节点 A 收到 A1 区块，节点 B 收到 B1 区块，各自广播全网，分叉产生；双方均以各自区块作为顶点计算新区块，假设 A 先算出 A2，B 后算出 B2，各自广播全网；对于 B 节点，由于广播原因(前述孤块例子中连子区块比父区块先收到这种情形都能出现)，它可能一直没有接收到 A1，A2，此时 B 算出新区块 B3，A 尚未算出，B 广播全网；对于某一节点 C，，它可能在 A、B 分别都算出 A2、B2 的时候都还没接收到 A1 或 B1，以上问题需要阅读比特币白皮书才能解决，此即下一章拜占庭问题

7. 延伸阅读-拜占庭将军问题

也被称为“拜占庭容错”、“拜占庭将军问题”。拜占庭将军问题是 Leslie Lamport（2013 年的图灵奖得主）用来为描述**分布式系统一致性问题**（Distributed Consensus），例子大意如下:

返回目录

拜占庭帝国想要进攻一个强大的敌人，为此派出了 10 支军队去包围这个敌人。这个敌人虽不比拜占庭帝国，但也足以抵御 5 支常规拜占庭军队的同时袭击。这 10 支军队在分开的包围状态下同时攻击。他们任一支军队单独进攻都毫无胜算，除非有至少 6 支军队（一半以上）同时袭击才能攻下敌国。他们分散在敌国的四周，依靠通信兵骑马相互通信来协商进攻意向及进攻时间。困扰这些将军的问题是，他们不确定他们中是否有叛徒，叛徒可能擅自变更进攻意向或者进攻时间。在这种状态下，拜占庭将军们才能保证有多于 6 支军队在同一时间一起发起进攻，从而赢取战斗(拜占庭将军问题中并不去考虑通信兵是否会被截获或无法传达信息等问题，即消息传递的信道绝无问题。Lamport 已经证明了*(课后阅读)*在消息可能丢失的不可靠信道上试图通过消息传递的方式达到一致性是不可能的。所以，在研究拜占庭将军问题的时候，已经假定了信道是没有问题的)

1. 问题分析

- 先看在没有叛徒情况下，假如一个将军 A 提一个进攻提议（如：明日下午 1 点进攻，你愿意加入吗？）由通信兵通信分别告诉其他的将军，如果幸运中的幸运，他收到了其他 6 位将军以上的同意，发起进攻。如果不幸，其他的将军也在此时发出不同的进攻提议（如：明日下午 2 点、3 点进攻，你愿意加入吗？），由于时间上的差异，不同的将军收到（并认可）的进攻提议可能是不一样的，这是可能出现 A 提议有 3 个支持者，B 提议有 4 个支持者，C 提议有 2 个支持者等等
- 再加一点复杂性，在有叛徒情况下，一个叛徒会向不同的将军发出不同的进攻提议（通知 A 明日下午 1 点进攻， 通知 B 明日下午 2 点进攻等等），一个叛徒也会可能同意多个进攻提议（即同意下午 1 点进攻又同意下午 2 点进攻）

2. 比特币的解决方案

在出现比特币之前，解决分布式系统一致性问题主要是 Lamport 提出的 Paxos 算法或其衍生算法。Paxos 类算法仅适用于中心化的分布式系统，这样的系统的没有不诚实的节点（不会发送虚假错误消息，但允许出现网络不通或宕机出现的消息延迟）。

比特币通过工作量证明增加了发送信息的成本，降低节点发送消息速率，以保证在一个时间只有一个节点(或是很少)在进行广播，同时在广播时会附上自己的签名。以上就是比特币网络中是单个区块（账本）达成共识的方法（取得一致性）。理解了单个区块取得一致性的方法，那么整个区块链（总账本）如果达成一致也好理解。

3. 经济学分析

工作量证明其实相当于提高了做叛徒（发布虚假区块）的成本，在工作量证明下，只有第一个完成证明的节点才能广播区块，竞争难度非常大，需要很高的算力，如果不成功其算力就白白的耗费了

（算力是需要成本的），如果有这样的算力作为诚实的节点，同样也可以获得很大的收益（这就是矿工所作的工作），这也实际就不会有做叛徒的动机，整个系统也因此而更稳定。

很多人批评工作量证明造成巨大的电力浪费，促使人们去探索新的解决一致性（共识）问题的机制：权益证明机制（POS: Proof of Stake）是一个代表。在拜占庭将军问题的角度来看，它同样提高了做叛徒的成本，因为账户需要首先持有大量余额才能有更多的几率广播区块。共识算法的核心就是解决拜占庭将军问题（分布式网络一致性问题）

8. 延伸阅读-分析比特币网络：一种去中心化、点对点的网络架构

1. P2P 网络

P2P 网络是指位于同一网络中的每台计算机都彼此对等，各个节点共同提供网络服务，不存在任何“特殊”节点，每个网络节点以扁平（flat）的拓扑结构相互连通。

2. 如何发现节点

新节点启动后，它是如何跟其他的节点建立连接，从而加入到比特币网络。为了能够加入到比特币网络，比特币客户端会做以下几件事情(仅列出新节点如何加入比特币网络，其余参考《分析比特币网络：一种去中心化、点对点的网络架构》):

- 在节点启动时(*这是新节点加入吗？*)，可以给节点指定一个正活跃节点 IP，如果没有，客户端也维持一个列表，列出了那些长期稳定运行的节点(*这应该就是新节点加入网络的方式*)。这样的节点也被称为种子节点（其实和 BT 下载的种子文件道理是一样的），就可以通过种子节点来快速发现网络中的其他节点

节点间通信参考《分析比特币网络：一种去中心化、点对点的网络架构》

9. 延伸阅读-比特币区块结构 Merkle 树及简单支付验证分析

10.延伸阅读-比特币脚本及交易分析 - 智能合约雏形

1. 未花费的交易输出(UTXO)

未花费的交易输出——UTXO（Unspent Transaction Output），即交易的输入是之前交易未花费的输出，这笔交易的输出可以被当做下一笔新交易的输入。具备如下三个特征：

- 挖矿奖励属于一个特殊的交易（称为 coinbase 交易），可以没有输入。
- UTXO 是交易的基本单元，不能再分割。

- 在比特币没有余额概念，只有分散到区块链里的 UTXO

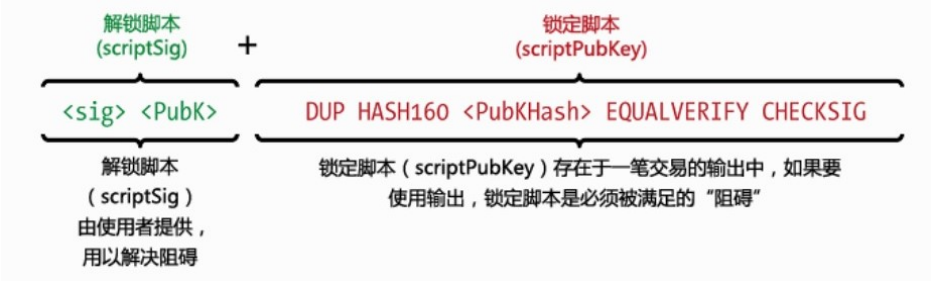
2. 比特币脚本

解锁 UTXO：用私钥去匹配锁定脚本

交易输入：一个用于解锁 UTXO 的脚本（常称为解锁脚本：Signature script)。

交易输出：交易的输出则是指向一个脚本（称为锁定脚本：PubKey script），这个脚本表达了：谁的签名（签名是常见形式，并不一定必须是签名）能匹配这个输出地址，钱就支付给谁。

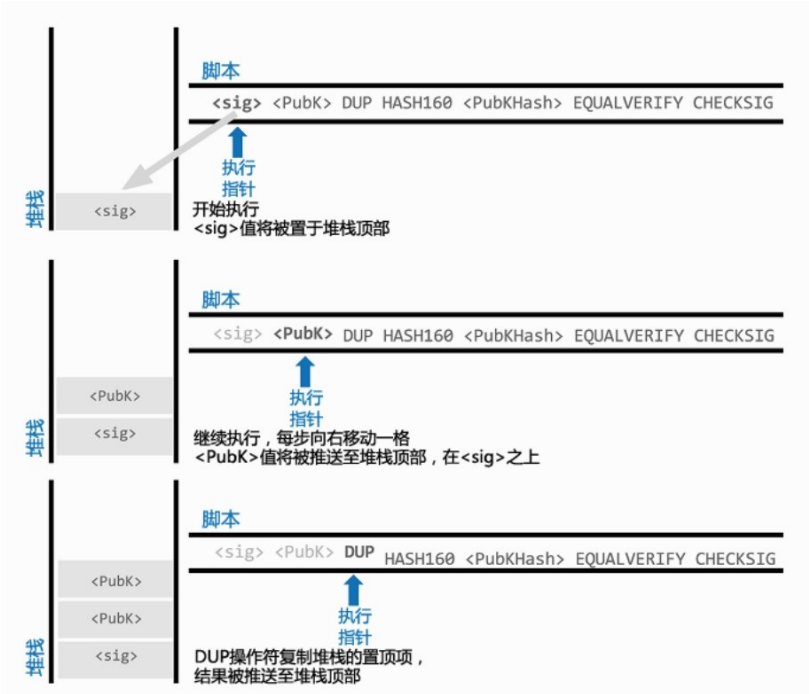
每一个比特币节点会通过同时执行这解锁和锁定脚本（不是当前的锁定脚本，是指上一个交易的锁定脚本）来验证一笔交易，脚本组合结果为真，则为有效交易。最为常见类型的比特币交易脚本（支付到公钥哈希：P2PKH（Pay-to-Public-Key-Hash））组合如下：



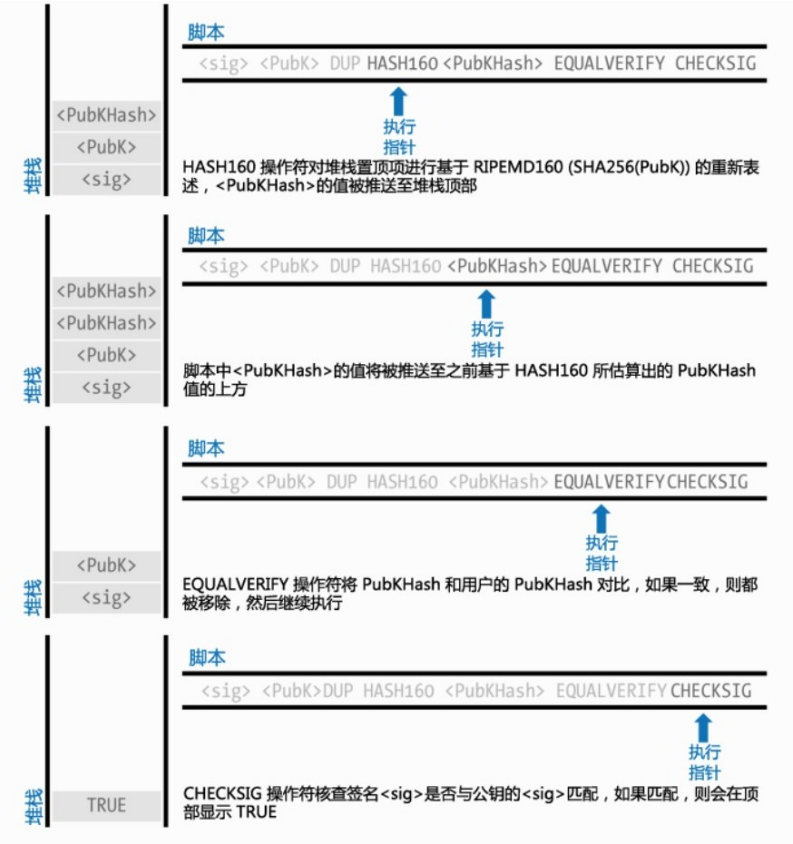
3. 常见交易脚本验证过程

比特币交易脚本语言是一种基于逆波兰表示法的基于栈的执行语言

解锁脚本运行过程（主要是入栈）如下：



下图为锁定脚本运行过程（主要是出栈），最后的结果为真，说明交易有效。



4. 交易分析

比特币的交易被设计为可以纳入多个输入和输出

- 完整交易结构

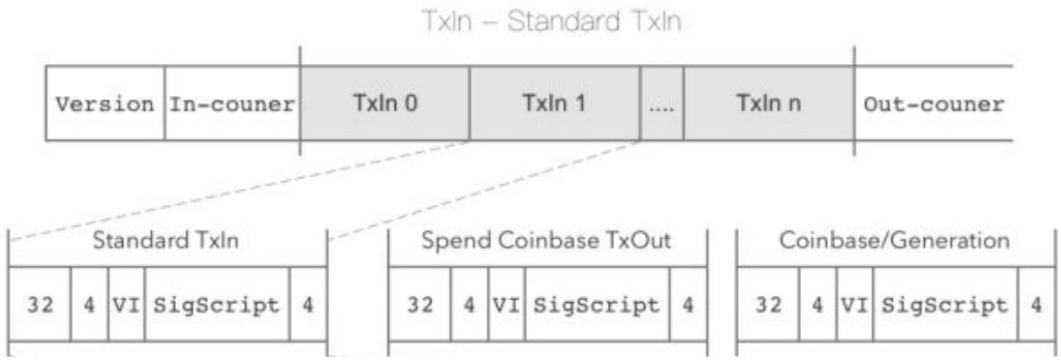
详细的各字段意义参考《比特币脚本及交易分析 - 智能合约雏形》

字段	描述	大小
版本	这笔交易参照的规则	4 字节
输入数量	交易输入列表的数量	1 - 9 字节
输入列表	一个或多个交易输入	不定
输出数量	交易输出列表的数量	1 - 9 字节
输出列表	一个或多个交易输出	不定
锁定时间	锁定时间	4 字节

- 交易输入结构

字节	字段	描述
32	交易哈希值	指向被花费的UTXO所在的交易的哈希指针
4	输出索引	被花费的UTXO的索引号，第一个是0
1-9	解锁脚本大小	用字节表示的后面的解锁脚本长度
不定	解锁脚本	满足UTXO解锁脚本条件的脚本
4	序列号	目前未被使用的交易替换功能，设为0xFFFFFFFF

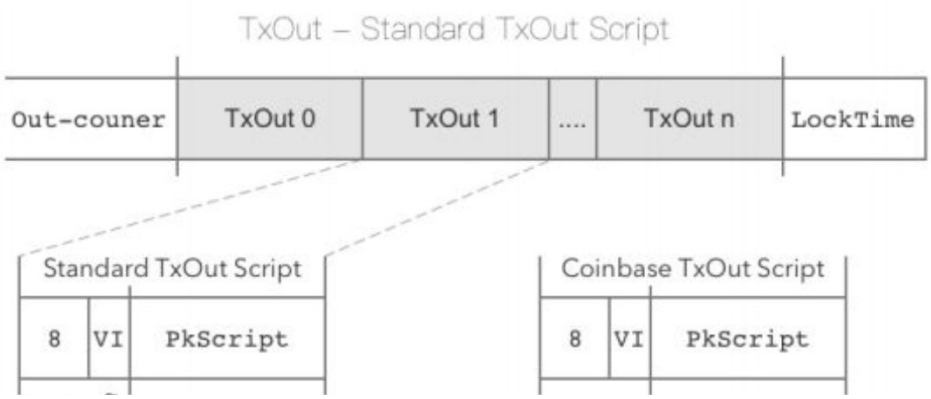
结合整个交易的结构里看输入结构:



- 交易输出结构

字节	字段	描述
8	总量	用聪表示的比特币值
1-9	锁定脚本大小	用字节表示的后面的锁定脚本长度
不定	锁定脚本	一个定义了支付输出所需条件的脚本

结合整个交易的结构里看输出结构



- 交易哈希计算

参考《比特币脚本及交易分析 - 智能合约雏形》

11.延伸阅读-以太坊是什么 - 以太坊开发入门指南

1. 以太坊是什么

以太坊（Ethereum）是一个建立在区块链技术之上，去中心化应用平台。它允许任何人在平台中建立和使用通过区块链技术运行的去中心化应用。(*通俗理解：以太坊是区块链里的 Android，它是一个开发平台*)

在没有以太坊之前，写区块链应用是这样的：拷贝一份比特币代码，然后去改底层代码如加密算法，共识机制，网络协议等等（山寨币就是这样，改改就出来一个新币）。以太坊平台对底层区块链技术进行了封装，让区块链应用开发者可以直接基于以太坊平台进行开发，开发者只要专注于应用本身的开发。

2. 智能合约

以太坊上的程序称之为智能合约，它是代码和数据(状态)的集合

3. 运行环境：EVM

EVM（Ethereum Virtual Machine）以太坊虚拟机是以太坊中智能合约的运行环境。Solidity(ETH 编程语言)之于 EVM，就跟 JAVA 之于 JVM 的关系(*我也不知道什么关系*)一样。以太坊虚拟机是一个隔离的环境，外部无法接触到在 EVM 内部运行的代码。

EVM 运行在以太坊节点上，当我们把合约部署到以太坊网络上之后，合约就可以在以太坊网络中运行了。

4. 合约的部署

在以太坊上开发应用时，常常要使用到以太坊客户端（钱包）。平时我们在开发中，一般不接触到客户端或钱包的概念，它是什么呢?

- **以太坊客户端（钱包）**

可以把它理解为一个开发者工具，它提供账户管理、挖矿、转账、智能合约的部署和执行等等功能。EVM 是由以太坊客户端提供的(*由客户端提供，但是运行在以太坊网络上*)

- **智能合约的部署**

智能合约的部署是指把合约字节码发布到区块链上，并使用一个特定的地址来标示这个合约，这个地址称为合约账户。以太坊中有两类账户：A. 外部账户：该类账户被私钥控制（由人控制），没

有关联任何代码。B. 合约账户，该类账户被它们的合约代码控制且有代码与之关联。和比特币使用 UTXO 的设计不一样，以太坊使用更为简单的账户概念。两类账户对于 EVM 来说是一样的。

合约部署就是将编译好的合约字节码通过外部账号发送交易的形式部署到以太坊区块链上（由实际矿工出块之后，才真正部署成功）

- 运行

合约部署之后，当需要调用这个智能合约的方法时只需要向这个合约账户发送消息（交易）即可，通过消息触发后智能合约的代码就会在 EVM 中执行了。

5. 总结

以太坊是平台，它让我们方便的使用区块链技术开发去中心化的应用，在这个应用中，使用 Solidity 来编写和区块链交互的智能合约，合约编写好后之后，我们需要用以太坊客户端用一个有余额的账户去部署及运行合约。

更多细节参考《以太坊是什么 - 以太坊开发入门指南》

12.附录

1. 表

表:

- 1
- 2

2. 图

- 1
- 2