10.1 简介

“挖矿”这个词有点误导。 一般意义的挖矿类似贵金属的提取，更多将人们的注意力集中到创造每个区块中获得的奖励。 虽然挖矿能够获得这种奖励作为激励，但挖矿的主要目的不是这个奖励或者产生新币。 如果您只是把挖矿视为创建新币的过程，则会将比特币系统中的这个手段（激励）作为挖矿过程的目标。 挖矿最重要的作用是巩固了去中心化的清算交易机制，通过这种机制，交易得到验证和清算//清除。 挖矿是使得比特币与众不同的发明，它实现去中心化的安全机制，是P2P数字货币的基础。

挖矿确保了比特币系统安全，并且在没有中央权力机构的情况下实现了全网络范围的共识。 新币发行和交易费的奖励是将矿工的行动与网络安全保持一致的激励计划，同时实现了货币发行。

提示：挖矿的目的不是创造新的比特币。 这是激励机制。 挖矿是一种机制，这种机制实现了去中心化的安全。

矿工们验证每笔新的交易并把它们记录在总帐簿上。每10分钟就会有一个新的区块被“挖掘”出来，每个区块里包含着从 上一个区块产生到目前这段时间内发生的所有交易，这些交易被依次添加到区块链中。我们把包含在区块内且被添加到 区块链上的交易称为“确认”（confirmed）交易，交易经过“确认”之后，新的拥有者才能够花费他在交易中得到的比特币。

矿工们在挖矿过程中会得到两种类型的奖励：创建新区块的新币奖励，以及区块中所含交易的交易费。为了得到这些奖 励，矿工们争相完成一种基于加密哈希算法的数学难题，这些难题的答案包括在新区块中，作为矿工的计算工作量的证 明，被称为““工作量证明”。该算法的竞争机制以及获胜者有权在区块链上进行交易记录的机制，这二者是比特币安全的基石。

新比特币的生成过程被称为挖矿，是因为它的奖励机制被设计为速度递减模式，类似于贵重金属的挖矿过程。比特币的货币是通过挖矿发行的，类似于中央银行通过印刷银行纸币来发行货币。矿工通过创造一个新区块得到的比特币数量大约 每四年（或准确说是每210,000个块）减少一半。开始时为2009年1月每个区块奖励50个比特币，然后到2012年11月减 半为每个区块奖励25个比特币。之后在2016年7月 再次减半为每个新区块奖励12.5个比特币。基于这个公 式，比特币挖矿奖励以指数方式递减，直到2140年。届时所有的比特币（20,999,999,980）全部发行完毕。换句话说 在2140年之后，不会再有新的比特币产生。

矿工们同时也会获取交易费。每笔交易都可能包含一笔交易费，交易费是每笔交易记录的输入和输出的差额。在挖矿过 程中成功“挖出”新区块的矿工可以得到该区块中包含的所有交易“小费”。目前，这笔费用占矿工收入的0.5%或更少，大 部分收益仍来自挖矿所得的比特币奖励。然而随着挖矿奖励的递减，以及每个区块中包含的交易数量增加，交易费在矿 工收益中所占的比重将会逐渐增加。在2140年之后，所有的矿工收益都将由交易费构成。

在本章中，我们先来审视比特币的货币发行机制，然后再来了解挖矿的最重要的功能：支撑比特币安全的去中心化的共识机制。

为了了解挖矿和共识，我们将跟随Alice的交易，以及上海的矿工Jing如何收到并利用挖矿设备将交易加入区块。 然后，我们将继续跟踪区块被挖矿，加入区块链，并通过共识被比特币网络接受。

10.1.1 比特币经济学和货币创造

通过创造出新区块，比特币以一个确定的但不断减慢的速率被铸造出来。大约每十分钟产生一个新区块，每一个新区块 都伴随着一定数量从无到有的全新比特币。每开采210,000个块，大约耗时4年，货币发行速率降低50%。在比特币运 行的第一个四年中，每个区块创造出50个新比特币。

2012年11月，比特币的新发行速度降低到每区块25个比特币。2016年7月，降低到12.5比特币/区块。2020年的某个时候，也就是在区块630,000，它将再次下降至6.25比特币。新币的发行速度会以指数级进行32次“等分”，直到第6,720,000块（大约在2137年开采），达到比特币的最小货币单位1 satoshi。最终，在经过693万个区块之后，所有的共 2,099,999,997,690,000聪比特币将全部发行完毕。也就是说，到2140年左右，会存在接近 2,100万比特

币。在那之后，新的区块不再包含比特币奖励，矿工的收益全部来自交易费。图10-1展示了在发行速度不断降低的情况下，比特币总流通量与时间的关系。

图10-1 比特币发行与时间的关系

注意：比特币挖矿发行的最大数量也就成为挖矿奖励的上限。 在实践中，矿工可能故意挖掘哪些不足全额奖励的区块。 这些块已经开采了，未来可能会有更多被开采，这样导致货币发行总量的下降。

在例10-1的代码展示中，我们计算了比特币的总发行量

例10-1 比特币发行总量的计算脚本

```
link:code/max_money.py[]
```

Running the max_money.py script说明了运行脚本的输出结果

例10-2 显示上述脚本的输出

```
$ python max_money.pyTotal BTC to ever be created: 2099999997690000 Satoshis
```

总量有限并且发行速度递减创造了一种抗通胀的货币供应模式。法币可被中央银行无限制地印刷出来，而比特币永远不会因超额印发而出现通胀。

通货紧缩

最重要并且最有争议的一个结论是一种事先确定的发行速率递减的货币发行模式会导致货币通货紧缩（简称通缩）。通缩是一种由于货币的供应和需求不匹配导致的货币增值的现象。它与通胀相反，价格通缩意味着货币随着时间有越来越强的购买力。

许多经济学家提出通缩经济是一种无论如何都要避免的灾难型经济。因为在快速通缩时期，人们预期着商品价格会下跌，人们将会储存货币，避免花掉它。这种现象充斥了日本经济“失去的十年”，就是因为在需求坍塌之后导致了滞涨状态。

比特币专家们认为通缩本身并不坏。更确切地说，我们将通缩与需求坍塌联系在一起是因为过去出现的一个特例。在法币届，货币是有可能被无限制印刷出来的，除非遇到需求完全崩塌并且毫无发行货币意愿的情形，因此经济很难进入滞涨期。而比特币的通缩并不是需求坍塌引起的，它遵循一种预定且有节制的货币供应模型。

通货紧缩的积极因素当然是与通货膨胀相反。 通货膨胀导致货币缓慢但不可避免的贬值，这是一种隐性税收的形式，惩罚在银行存钱的人从而实现解救债务人（包括政府这个最大的债务人）。 政府控制下的货币容易遭受债务发行的道德风险，之后可能会以牺牲储蓄者为代价，通过贬值来抹去债务。

比特币这种不是因经济快速衰退而引起的通缩，是否会引发其他问题，仍有待观察。

10.2 去中心化共识

在上一章中我们了解了区块链。可以将区块链看作一本记录所有交易的公开总帐簿（列表），比特币网络中的每个参与 者都把能接受区块链，把它看作一本证明所有权的权威记录。

但在不考虑相信任何人的情况下，比特币网络中的所有参与者如何达成对任意一个所有权的共识呢？所有的传统支付系 统都依赖于一个中心认证机构，依靠中心机构提供的结算服务来验证并处理所有的交易。比特币没有中心机构，几乎所 有的完整节点都有一份公共总帐的备份，这份总帐可以被视为权威记录。区块链并不是由一个中心机构创造的，它是由比特币网络中的所有节点各自独立竞争完成的。换句话说比特币网络中的所有节点，依靠着节点间

的不稳定的网络连接所传输的信息，最终得出同样的结果并维护了同一个公共总帐。这一章将介绍比特币网络不依靠中心机构而达成共识的机制。

中本聪的主要发明就是这种去中心化的自发共识（emergent consensus）机制。这种自发，是指共识没有明确的完成点，因为共识达成时，没有明确的选举和固定时刻。换句话说，共识是数以千计的独立节点遵守了简单的规则通过异步交互自发形成的产物。所有的比特币属性，包括货币、交易、支付以及不依靠中心机构和信任的安全模型等都依赖于这个发明。

比特币的去中心化共识由所有网络节点的4种独立过程相互作用而产生：

▷ 每个全节点依据综合标准对每个交易进行独立验证

▷ 通过完成工作量证明算法的验算，挖矿节点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。

在接下来的几节中，我们将审视这些过程，了解它们之间如何相互作用并达成全网的自发共识，从而使任意节点组合出 它自己的权威、可信、公开的总帐副本。

10.3 交易的独立校验

在第6章交易中，我们知道了钱包软件通过收集UTXO、提供正确的解锁脚本、构造一个新的支出（支付）给接收者这一系列的方式来创建交易。产生的交易随后将被发送到比特币网络临近的节点，从而使得该交易能够在整个比特币网络中传播。

然而，在交易传递到临近的节点前，每一个收到交易的比特币节点将会首先验证该交易，这将确保只有有效的交易才会 在网络中传播，而无效的交易将会在第一个节点处被废弃。

每一个节点在校验每一笔交易时，都需要对照一个长长的标准列表：

▷交易的语法和数据结构必须正确。

▷输入与输出列表都不能为空。

▷交易的字节大小是小于 MAX_BLOCK_SIZE 的。

▷每一个输出值，以及总量，必须在规定值的范围内 （小于2,100万个币，大于0）。

▷没有哈希等于0，N等于-1的输入（coinbase交易不应当被传递）。

▷nLockTime是小于或等于 INT_MAX 的。或者nLocktime and nSequence的值满足MedianTimePast（译者注：MedianTime是这个块的前面11个块按照block time排序后的中间时间）

▷交易的字节大小是大于或等于100的。

▷交易中的签名数量(SIGOPS)应小于签名操作数量上限。

▷解锁脚本（ scriptSig ）只能够将数字压入栈中，并且锁定脚本（ scriptPubkey ）必须要符合isStandard的格式（该格式将会拒绝非标准交易）。

▷池中或位于主分支区块中的一个匹配交易必须是存在的。

▷对于每一个输入，引用的输出是必须存在的，并且没有被花费。

▷对于每一个输入，如果引用的输出存在于池中任何别的交易中（译者注：这笔输入引用的输出有人家自己的输入，不是你），该交易将被拒绝。

▷对于每一个输入，在主分支和交易池中寻找引用的输出交易。如果输出交易缺少任何一个输入，该交易将成为一个孤 立的交易。如果与其匹配的交易还没有出现在池中，那么将被加入到孤立交易池中。

▷对于每一个输入，如果引用的输出交易是一个coinbase输出，该输入必须至少获得 COINBASE_MATURITY(100)个确认。

▷使用引用的输出交易获得输入值，并检查每一个输入值和总值是否在规定值的范围内 （小于2100万个币，大于0）。

▷如果输入值的总和小于输出值的总和，交易将被中止。

▷如果交易费用太低以至于无法进入一个空的区块，交易将被拒绝。

▷每一个输入的解锁脚本必须依据相应输出的锁定脚本来验证。

这些条件能够在比特币标准客户端下的 AcceptToMemoryPool 、 CheckTransaction 和 CheckInputs 函数中获得更详细的阐述。 请注意，这些条件会随着时间发生变化，为了处理新型拒绝服务攻击，有时候也为交易类型多样化而放宽规则。

在收到交易后，每一个节点都会在全网广播前对这些交易进行独立校验，并以接收时的相应顺序，为有效的新交易建立一个验证池（还未确认），这个池可以叫做交易池，或者memory pool或者mempool。

10.4 挖矿节点

在比特币网络中，一些节点被称为专业节点“矿工”。第1章中，我们介绍了Jing，在中国上海的计算机工程专业学生，他 就是一位矿工。Jing通过矿机挖矿获得比特币，矿机是专门设计用于挖比特币的计算机硬件系统。Jing的这台专业挖矿设备连接着一个运行完整比特币节点的服务器。与Jing不同，一些矿工是在没有完整节点的条件下进行挖矿，正如我们 在“矿池”一节中所述的。与其他任一完整节点相同，Jing的节点在比特币网络中进行接收和传播未确认交易记录。然而，Jing的节点也能够把这些交易记录打包进入一个新区块。

同其他节点一样，Jing的节点时刻监听着传播到比特币网络的新区块。而这些新加入的区块对挖矿节点有着特殊的意 义。矿工间的竞争以新区块的传播而结束，如同宣布谁是最后的赢家。对于矿工们来说，收到一个新区块进行验证意味着别人已经赢了，而自己则输了这场竞争。然而，一轮竞争的结束也代表着下一轮竞争的开始。新区块并不仅仅是象征着竞赛结束的方格旗；它也是下一个区块竞赛的发令枪。

10.5 打包交易至区块

验证交易后，比特币节点会将这些交易添加到自己的内存池中。内存池也称作交易池，用来暂存尚未被加入到区块的交 易记录。与其他节点一样，Jing的节点会收集、验证并传递新的交易。而与其他节点不同的是，Jing的节点会把这些交 易整合到一个候选区块中。

让我们继续跟进，看下Alice从Bob咖啡店购买咖啡时产生的那个区块。Alice的交易在区块 277,316。为了演示本章中提到的概念，我们假设这个区块是由Jing的挖矿系统挖出的，并且继续跟进Alice的交易，因 为这个交易已经成为了新区块的一部分。

Jing的挖矿节点维护了一个区块链的本地副本。当Alice买咖啡 的时候，Jing节点的区块链已经收集到了区块277,314，并继续监听着网络上的交易，在尝试挖掘新区块的同时，也监 听着由其他节点发现的区块。当Jing的节点在挖矿时，它从比特币网络收到了区块277,315。这个区块的到来标志着终 结了产出区块277,315竞赛，与此同时也是产出区块277,316竞赛的开始。

在上一个10分钟内，当Jing的节点正在寻找区块277,315的解的同时，它也在收集交易记录为下一个区块做准备。目前 它已经收到了几百笔交易记录，并将它们放进了内存池。直到接收并验证区块277,315后，Jing的节点会检查内存池中 的全部交易，并移除已经在区块277,315中出现过的交易记录，确保任何留在内存池中的交易都是未确认的，等待被记 录到新区块中。

Jing的节点立刻构建一个新的空区块，做为区块277,316的候选区块。称作候选区块是因为它还没有包含有效的工作量 证明，不是一个有效的区块，而只有在矿工成功找到一个工作量证明解之后，这个区块才生效。

现在，Jing的节点从内存池中整合到了全部的交易，新的候选区块包含有418笔交易，总的矿工费为0.09094925个比特 币。你可以通过比特币核心客户端命令行来查看这个区块，如例10-3所示：

例10-3 使用命令行检索区块277,316

```
$ bitcoin-cli get blockhash 277316
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

$ bitcoin-cli getblock 0000000000000001b6b9a13b095e96db41c4a928b97ef2d9\44a9b31b2cc7bdc4
```

```
{
"hash" : "0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
"confirmations" : 35561,
"size" : 218629,
"height" : 277316,
"version" : 2,
"merkleroot" : "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
"tx" : [
  "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
  "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",
  ... 417 more transactions ...//这一行显示交易数量
 ],
"time" : 1388185914,
"nonce" : 924591752,
"bits" : "1903a30c",
"difficulty" : 1180923195.25802612,
"chainwork" : "0000000000000000000000000000000000000000000000934695e92aaf53afa1a",
"previousblockhash" : "0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569"
}
```

10.5.1 创币交易

区块中的第一笔交易是笔特殊交易，称为创币交易或者coinbase交易。这个交易是由Jing的节点构造并用来奖励矿工们所做的贡献的。

注意:当块277,316开采时，每个块的奖励是25比特币。 此后，已经过了一个“减半”时期。 2016年七月份的奖励为12.5比特币，2020年达到210000区块时，将再次减半。

Jing的节点会创建“向Jing的地址支付25.09094928个比特币”这样一个交易，把生成交易的奖励发送到自己的钱包。Jing挖出区块获得的奖励金额是coinbase奖励（25个全新的比特币）和区块中全部交易矿工费的总和。

如 例10-4所示：coinbase交易

```
$ bitcoin-cli getrawtransaction
d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1
```

```
{
"hex" :
"010000000100000000000000000000000000000000000000000000000000000000ffffffff0f034
43b0403858402062f503253482fffffffff0110c08d9500000000232102aa970c592640d19de03ff6f329d6f
d2eecb023263b9ba5d1b81c29b523da8b21ac00000000",
"txid" : "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
"version" : 1,
"locktime" : 0,
"vin" : [
  {
  "coinbase" : "03443b0403858402062f503253482f",
  "sequence" : 4294967295
  }
],
"vout" : [
  {
  "value" : 25.09094928,
  "n" : 0,
  "scriptPubKey" : {
  "asm" :
"02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21OP_CHECKSIG",
  "hex" : "2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac",
  "reqSigs" : 1,
  "type" : "pubkey",
  "addresses" : ["1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
    ]
  }
  }
]
}
```

与常规交易不同，创币交易没有输入，不消耗UTXO。它只包含一个被称作coinbase的输入，仅仅用来创建新的比特 币。创币交易有一个输出，支付到这个矿工的比特币地址。创币交易的输出将这25.09094928个比特币发送到矿工的比 特币地址，如本例所示的1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N。

10.5.2 Coinbase奖励与矿工费

为了构造创币交易，Jing的节点需要计算矿工费的总额，将这418个已添加到区块交易的输入和输出分别进行求和，然 后用输入总额减去输出总额得到矿工费总额，公式如下：

```
Total Fees = Sum(Inputs) - Sum(Outputs)
```

在区块277,316中，矿工费的总额是0.09094925个比特币。

紧接着，Jing的节点计算出这个新区块正确的奖励额。奖励额的计算是基于区块高度的，以每个区块50个比特币为开 始，每产生210,000个区块减半一次。这个区块高度是277,316，所以正确的奖励额是25个比特币。

详细的计算过程可以参看比特币核心客户端中的GetBlockValue函数，如例10-5所示：

例10-5 计算区块奖励—函数GetBlockValue, Bitcoin Core Client, main.cpp

```
CAmount GetBlockSubsidy(int nHeight, const Consensus::Params& consensusParams){
int halvings = nHeight / consensusParams.nSubsidyHalvingInterval;
// Force block reward to zero when right shift is undefined.
if (halvings >= 64)
    return 0;

CAmount nSubsidy = 50 * COIN;
// Subsidy is cut in half every 210,000 blocks which will occur approximately every 4
years.
nSubsidy >> = halvings;
return nSubsidy;
}
```

变量Subsidy表示初始奖励额，值为 COIN 常量（100,000,000聪)与50的乘积，也就是说初始奖励额为50亿聪。

紧接着，这个函数用当前区块高度除以减半间隔(SubsidyHalvingInterval 函数)得到减半次数（变量 halvings ）。每 210,000个区块为一个减半间隔，对应本例中的区块277316，所以减半次数为1。

变量 halvings 最大值64，如果超出这个值，代码算得的奖励额为0。

然后，这个函数会使用二进制右移操作将奖励额(变量 nSubsidy)右移一位（等同与除以2），每一轮减半右移一次。在这个例子中，对于区块277,316只需要将值为50亿聪的奖励额右移一次，得到25亿聪，也就是25个比特币的奖励额。之所以采用二进制右移操作，是因为相比于整数或浮点数除法，右移操作的效率更高。

最后，将coinbase奖励额（变量 nSubsidy ）与矿工费(nFee)总额求和，并返回这个值。

注意: 如果Jing的挖矿节点把coinbase交易写入区块，那么如何防止Jing奖励自己100甚至1000比特币？ 答案是，不正确的奖励将被其他人视为无效，浪费了Jing用于工作证明的投入。 只有这个区块被大家认可，Jing才能得到报酬。

10.5.3创币交易的结构

经过计算，Jing的节点构造了一个创币交易，支付给自己25.09094928枚比特币。

例10-4所示，创币交易的结构比较特殊，与一般交易输入需要指定一个先前的UTXO不同，它包含一个“coinbase“输 入。在之前的章节中，我们已经给出了交易输入的结构。现在让我们来比较一下常规交易输入与创币交易输入。表10-1给出了常规交易输入的结构，表10-2给出的是创币交易输入的结构。

表10-1 常规交易输入结构

Size	Field	Description
32 bytes	Transaction Hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent, first one is 0
1–9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking script
4 bytes	Sequence Number	Currently disabled Tx-replacement feature, set to 0xFFFFFFFF

表10-2，coinbase交易输入结构

Size	Field	Description
32 bytes	Transaction Hash	All bits are zero: Not a transaction hash reference
4 bytes	Output Index	All bits are ones: 0xFFFFFFFF
1–9 bytes (VarInt)	Coinbase Data Size	Length of the coinbase data, from 2 to 100 bytes
Variable	Coinbase Data	Arbitrary data used for extra nonce and mining tags. In v2 blocks; must begin with block height
4 bytes	Sequence Number	Set to 0xFFFFFFFF

在Coinbase交易中，“交易哈希”字段32个字节全部填充0，“交易输出索引”字段全部填充0xFF(十进制的255)，这两个字段的值表示不引用UTXO。“解锁脚本”由coinbase数据代替，数据可以由矿工自定义。

10.5.4 Coinbase数据

创币交易不包含“解锁脚本“(又称作 scriptSig)字段，这个字段被coinbase数据替代，长度最小2字节，最大100字节。除 了开始的几个字节外，矿工可以任意使用coinbase的其他部分，随意填充任何数据。

以创世块为例，中本聪在coinbase中填入了这样的数据“The Times 03/Jan/ 2009 Chancellor on brink of second bailout for banks“(泰晤士报 2009年1月3日 财政大臣将再次对银行施以援手)，表示对日期的证明，同时也表达了对银行系统的不信任。现在，矿工使用coinbase数据实现extra nonce功能，并嵌入字符串来标识挖出它的矿池，这部分内容会在后面的小节描述。

coinbase前几个字节也曾是可以任意填写的，不过在后来的第34号比特币改进提议(BIP34)中 规定了版本2的区块（版本字段为2的区块），这个区块的高度必须跟在脚本操作“push“之后，填充在coinbase字段的起始处。

我们以例10-4中的区块277,316为例，coinbase就是交易输入的“解锁脚本“（或scriptSig）字段，这个字段的十六进制值 为03443b0403858402062f503253482f。下面让我们来解码这段数据。

第一个字节是03，脚本执行引擎执行这个指令将后面3个字节压入脚本栈(见表4-1);紧接着的3个字节——0x443b04， 是以小端格式(最低有效字节在先)编码的区块高度。翻转字节序得到0x043b44，表示为十进制是277,316。

紧接着的几个十六进制数（03858402062）用于编码extra nonce(参见"10.11.1 随机值升位方案")，或者一个随机值，从而求解一个适当的工作量证明。

coinbase数据结尾部分(2f503253482f)是ASCII编码字符 /P2SH/，表示挖出这个区块的挖矿节点支持BIP0016所定义的 pay-to-script-hash(P2SH)改进方案。在P2SH功能引入到比特币的时候，曾经有过一场对P2SH不同实现方式的投票， 候选者是BIP0016和BIP0017。支持BIP0016的矿工将/P2SH/放入coinbase数据中，支持BIP0017的矿工将 p2sh/CHV 放入他们的coinbase数据中。最后，BIP0016在选举中胜出，直到现在依然有很多矿工在他们的coinbase中填 入/P2SH/以表示支持这个功能。

10-6使用了libbitcoin库（在之前"其他替代客户端、资料库、工具包"中提到）从创世块中提取coinbase数据，并显示 出中本聪留下的信息。libbitcoin库中自带了一份创世块的静态拷贝，所以这段示例代码可以直接取自库中的创世块数 据。

例10-6 从创世区块中提取coinbase数据

```
link:code/satoshi-words.cpp\[\]
```

例8-7中，我们使用GNU C++编译器编译源代码并运行得到的可执行文件，

例8-7 编译并运行satoshi-words示例代码

```
$ # Compile the code
$  g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)
$ # Run the executable
$ ./satoshi-words
^D��<GS>^A^DEThe Times 03/Jan/2009 Chancellor on brink of second bailout for banks
```

10.6 构造区块头

为了构造区块头，挖矿节点需要填充六个字段，如表10-3中所示。

表10-3 区块头结构

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	Target	The Proof-of-Work algorithm target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

在区块277,316被挖出的时候，区块结构中用来表示版本号的字段值为2，长度为4字节，以小端格式编码值为0x20000000。

接着，挖矿节点需要填充“前区块哈希”，在本例中，这个值为Jing的节点从网络上接收到的区块277,315 的区块头哈希值，它是区块277316候选区块的父区块。区块277,315的区块头哈希值为：

```
000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

提示：通过选择候选块头中的先前块哈希字段指定的特定父区块，Jing正在通过确认挖矿能力来扩展区块链。 从本质上讲，这是用他的采矿权为最长难度的有效链进行的“投票”。

为了向区块头填充merkle根字段，要将全部的交易组成一个merkle树。创币交易作为区块中的首个交易，后将余下的 418笔交易添至其后，这样区块中的交易一共有419笔。在之前，我们已经见到过“Merkle树”，树中必须有偶数个叶子 节点，所以需要复制最后一个交易作为第420个叶子节点，每个叶子节点是对应交易的哈希值。这些交易的哈希值逐层地、成对地组合，直到最终组合并成一个根节点。merkle数的根节点将全部交易数据摘要为一个32字节长度的值，例10-3中 merkel根的值如下：

```
c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e
```

挖矿节点会继续添加一个4字节的时间戳，以Unix纪元时间编码，即自1970年1月1日0点到当下总共流逝的秒数。本例 中的1388185914对应的时间是2013年12月27日，星期五，UTC/GMT。

接下来，Jing的节点需要填充Target字段（难度目标值），为了使得该区块有效，这个字段定义了所需满足的工作量证明的难度。难度在区块中以“尾数-指数”的格式，编码并存储，这种格式称作target bits（难度位）。这种编码的首字节表示指数，后面的3字节表示尾数(系 数)。以区块277316为例，难度位的值为0x1903a30c，0x19是指数的十六进制格式，后半部0x03a30c是系数。这部分的概念在后面的“难度目标与难度调整”和“难度表示”有详细的解释。

最后一个字段是nonce，初始值为0。

区块头完成全部的字段填充后，挖矿就可以开始进行了。挖矿的目标是找到一个使区块头哈希值小于难度目标的nonce。挖矿节点通常需要尝试数十亿甚至数万亿个不同的nonce取值，直到找到一个满足条件的nonce值。

10.7 构建区块

既然Jing的节点已经构建了一个候选区块，那么就轮到Jing的矿机对这个新区块进行“挖掘”，求解工作量证明算法以使这个区块有效。从本书中我们已经学习了比特币系统中不同地方用到的哈希加密函数。比特币挖矿过程使用的是SHA256哈希函数。

用最简单的术语来说，挖矿就是重复计算区块头的哈希值，不断修改该参数，直到与哈希值匹配的一个过程。哈希函数 的结果无法提前得知，也没有能得到一个特定哈希值的模式。哈希函数的这个特性意味着：得到哈希值的唯一方法是不断的尝试，每次随机修改输入，直到出现适当的哈希值。

10.7.1 工作量证明算法

哈希函数输入一个任意长度的数据，输出一个长度固定且绝不雷同的值，可将其视为输入的数字指纹。对于特定输入，哈希的结果每次都一样，任何人都可以用相同的哈希函数，计算和验证哈希结果。一个加密哈希函数的主要特征就是不同的 输入几乎不可能出现相同的数字指纹。因此，有意的选择一个输入去生成一个想要的哈希值值是几乎不可能的，更别提用随机的方式生成想要的哈希值了。

无论输入的大小是多少，SHA256函数的输出的长度总是256bit。在例10-8中，我们将使用Python解释器来计算语句 "I am Satoshi Nakamoto" 的SHA256的哈希值。

例10-8 SHA256示例

```
$ python
Python 2.7.1
>>> import hashlib
>>> print hashlib.sha256("I am Satoshi Nakamoto").hexdigest()
5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e
```

在例10-8中， 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e 是"I am Satoshi Nakamoto"的哈希值。改变原句中的任何一个字母、标点、或增加字母都会产生不同的哈希值。

如果我们改变原句，得到的应该是完全不同的哈希值。例如，我们在句子末尾加上一个数字，运行例10-9中的Python脚本。例10-9 通过反复修改 nonce 来生成不同哈希值的脚本（SHA256）

```
link:code/hash\_example.py\[\]
```

执行这个脚本就能生成这些只是末尾数字不同的语句的哈希值。例10-10 中显示了我们只是增加了这个数字，却得到了非 常不同的哈希值。

例10-10 通过反复修改nonce 来生成不同哈希值的脚本的输出

```
$ python hash_example.py
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

每个语句都生成了一个完全不同的哈希值。它们看起来是完全随机的，但你在任何计算机上用Python执行上面的脚本都 能重现这些完全相同的哈希值。

类似这样在语句末尾的变化的数字叫做nonce（随机数）。Nonce是用来改变加密函数输出的，在这个示例中改变了这个语句的 SHA256指纹。

为了使这个哈希算法变得富有挑战，我们来设定一个具有任意性的目标：找到一个语句，使之哈希值的十六进制表示以 0开头。幸运的是，这很容易！在例10-10中语句 "I am Satoshi Nakamoto13" 的哈希值是 0ebc56d59a34f5082aaef3d66b37a661696c2b618e62432727216ba9531041a5 ，刚好满足条件。我们得到它用了13次。用概率的角度 来看，如果哈希函数的输出是平均分布的，我们可以期望每16次得到一个以0开头的哈希值（十六进制个位数字为0到 F）。从数字的角度来看，我们要找的是小于 0x1000000000000000000000000000000000000000000000000000000000000000 的哈希值。

我们称这个为Target目标阈值，我们的目的是找到一个小于这个目标的哈希值。如果我们减小这个目标值，那找到一个小于它的哈希值会越来越难。

简单打个比方，想象人们不断扔一对骰子以得到小于一个特定点数的游戏。第一局，目标是12。只要你不扔出两个6， 你就会赢。然后下一局目标为11。玩家只能扔10或更小的点数才能赢，不过也很简单。假如几局之后目标降低为了5。

现在有一半机率以上扔出来的骰子加起来点数会超过5，因此无效。随着目标越来越小，要想赢的话，扔骰子的次数会 指数级的上升。最终当目标为2时（最小可能点数），只有一个人平均扔36次或2%扔的次数中，他才能赢。从一个知道骰子游戏目标为2的观察者的角度来看，如果有人要成功中奖，假设他平均尝试了36次。

换句话说，可以估计从实现目标难度取得成功所需的工作量。 当算法是基于诸如SHA256的确定性函数时，输入本身就成为证据，必须要一定的工作量才能产生低于目标的结果。 因此，称之为工作量证明。

提示：尽管每次尝试产生一个随机的结果，但是任何可能的结果的概率可以预先计算。 因此，指定特定难度的结果构成了具体的工作量证明。

在例10-10中，成功的nonce为13，且这个结果能被所有人独立确认。任何人将13加到语句 "I am Satoshi Nakamoto" 后面再计算哈希值都能确认它比目标值要小。这个正确的结果同时也是工作量证明（Proof of Work），因为它证明我们的确花时间找到了这个nonce。验证这个哈希值只需要一次计算，而我们找到它却花了13次。如果目标值更小（难度更大），那我们需要多得多的哈希计算才能找到合适的nonce，但其他人验证它时只需要一次哈希计算。此外，知道目标值后，任何人都可以用统计学来估算其难度，因此就能知道找到这个nonce需要多少工作。

提示：工作证明必须产生小于目标的哈希值。 更高的目标意味着找到低于目标的哈希是不太困难的。 较低的目标意味着在目标下方找到哈希更难。 目标和难度是成反比。

比特币的工作量证明和例10-10中的挑战非常类似。矿工用一些交易构建一个候选区块。接下来，这个矿工计算这个区块头信息的哈希值，看其是否小于当前目标值。如果这个哈希值不小于目标值，矿工就会修改这个nonce（通常将之加 1）然后再试一次。按当前比特币系统的难度，矿工得试10^15次（10的15次方）才能找到一个合适的nonce使区块头信息哈希值足够小。

例10-11是一个简化很多的工作量证明算法的实现。

例10-11 简化的工作量证明算法

```
link:code/proof-of-work-example.py\[]
```

你可以任意调整难度值（按二进制bit数来设定，即哈希值开头多少个bit必须是0）。然后执行代码，看看在你的计算机 上求解需要多久。在例10-12中，你可以看到该程序在一个普通笔记本电脑上的执行情况。

例10-12 多种难度值的工作量证明算法的运行输出

```
$ python proof-of-work-example.py*
```

```
Difficulty: 1 (0 bits)

[...]

Difficulty: 8 (3 bits)
Starting search...
Success with nonce 9
Hash is 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1
Elapsed Time: 0.0004 seconds
Hashing Power: 25065 hashes per second
Difficulty: 16 (4 bits)
Starting search...
Success with nonce 25
Hash is 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Elapsed Time: 0.0005 seconds
Hashing Power: 52507 hashes per second
Difficulty: 32 (5 bits)
Starting search...
Success with nonce 36
Hash is 029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903
Elapsed Time: 0.0006 seconds
Hashing Power: 58164 hashes per second

[...]

Difficulty: 4194304 (22 bits)
Starting search...
Success with nonce 1759164
Hash is 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3
Elapsed Time: 13.3201 seconds
Hashing Power: 132068 hashes per second
Difficulty: 8388608 (23 bits)
Starting search...
Success with nonce 14214729
Hash is 000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3
Elapsed Time: 110.1507 seconds
Hashing Power: 129048 hashes per second
Difficulty: 16777216 (24 bits)
Starting search...
Success with nonce 24586379
Hash is 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Elapsed Time: 195.2991 seconds
Hashing Power: 125890 hashes per second

[...]

Difficulty: 67108864 (26 bits)
Starting search...
Success with nonce 84561291
Hash is 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Elapsed Time: 665.0949 seconds
Hashing Power: 127141 hashes per second
```

你可以看出，随着难度位一位一位地增加，查找正确结果的时间会呈指数级增长。如果你考虑整个256bit数字空间，每 次要求多一个0，你就把哈希查找空间缩减了一半。在例10-12中，为寻找一个nonce使得哈希值开头的26位值为0，一共尝试了8千多万次。即使家用笔记本每秒可以达270,000多次哈希计算，这个查找依然需要10分钟。

在写这本书的时候，比特币网络要寻找区块头信息哈希值小于

```
0000000000000000029AB9000000000000000000000000000000000000000000
```

可以看出，这个目标哈希值开头的0多了很多。这意味 着可接受的哈希值范围大幅缩减，因而找到正确的哈希值更加困难。生成下一个区块需要网络每秒计算1.8 septa-hashes（(thousand billion billion次哈希）。这看起来像是不可能的任务，但幸运的是比特币网络已经拥有3EH/sec的处理能力，平均每10分钟就可以找到一个新区块。

10.7.2 难度表示

在例10-3中，我们在区块中看到难度目标，其被标为"难度位"或简称"bits"。在区块277,316中，它的值为0x1903a30c。 这个标记的值被存为系数/指数格式，前两位十六进制数字为幂（exponent），接下来得六位为系数（coefficient）。在这个区块里，0x19为幂，而 0x03a30c为系数。

计算难度目标的公式为：

```
target = coefficient \* 2^\(8 \* \(exponent – 3\)\)
```

由此公式及难度位的值 0x1903a30c，可得：

```
    target = 0x03a30c * 2^(0x08 * (0x19 - 0x03))^

     => target = 0x03a30c * 2^(0x08 * 0x16)^

    => target = 0x03a30c * 2^0xB0^
```

按十进制计算为：

```
=> target = 238,348 * 2^176^

=> target =
22,829,202,948,393,929,850,749,706,076,701,368,331,072,452,018,388,575,715,328
```

转化为十六进制后为：

```
=> target = 0x000000000000003A30C0000000000000000000000000000000000000000000
```

也就是说高度为277,316的有效区块的头信息哈希值是小于这个目标值的。这个数字的二进制表示中前60位都是0。在 这个难度上，一个每秒可以处理1万亿个哈希计算的矿工（1 tera-hash per second 或 1 TH/sec）平均每8,496个区块才能找到一个正确结果，换句话说，平均每59天，才能为某一个区块找到正确的哈希值。

10.7.3 难度目标与难度调整

如前所述，目标决定了难度，进而影响求解工作量证明算法所需要的时间。那么问题来了：为什么这个难度值是可调整 的？由谁来调整？如何调整？

比特币的区块平均每10分钟生成一个。这就是比特币的心跳，是货币发行速率和交易达成速度的基础。不仅是在短期 内，而是在几十年内它都必须要保持恒定。在此期间，计算机性能将飞速提升。此外，参与挖矿的人和计算机也会不断 变化。为了能让新区块的保持10分钟一个的产生速率，挖矿的难度必须根据这些变化进行调整。事实上，难度是一个动 态的参数，会定期调整以达到每10分钟一个新区块的目标。简单地说，难度被设定在，无论挖矿能力如何，新区块产生 速率都保持在10分钟一个。

那么，在一个完全去中心化的网络中，这样的调整是如何做到的呢？难度的调整是在每个完整节点中独立自动发生的。 每2,016个区块中的所有节点都会调整难度。难度的调整公式是由最新2,016个区块的花费时长与20,160分钟（即 这些区块以10分钟一个速率所期望花费的时长）比较得出的。难度是根据实际时长与期望时长的比值进行相应调整的 （或变难或变易）。简单来说，如果网络发现区块产生速率比10分钟要快时会增加难度。如果发现比10分钟慢时则降低 难度。

这个公式可以总结为如下形式：

```
New Difficulty = Old Difficulty \* \(Actual Time of Last 2016 Blocks / 20160 minutes\)
```

例10-13展示了比特币核心客户端中的难度调整代码。

例10-13 工作量证明的难度调整 源文件(pow.cpp文件钟的CalculateNextWorkRequired() 函数)

第43行函数 GetNextWorkRequired()// Limit adjustment step

```
// Limit adjustment step
    int64_t nActualTimespan = pindexLast->GetBlockTime() - nFirstBlockTime;
    LogPrintf("  nActualTimespan = %d  before bounds\n", nActualTimespan);
    if (nActualTimespan < params.nPowTargetTimespan/4)
        nActualTimespan = params.nPowTargetTimespan/4;
    if (nActualTimespan > params.nPowTargetTimespan*4)
        nActualTimespan = params.nPowTargetTimespan*4;

    // Retarget
    const arith_uint256 bnPowLimit = UintToArith256(params.powLimit);
    arith_uint256 bnNew;
    arith_uint256 bnOld;
    bnNew.SetCompact(pindexLast->nBits);
    bnOld = bnNew;
    bnNew *= nActualTimespan;
    bnNew /= params.nPowTargetTimespan;

    if (bnNew > bnPowLimit)
        bnNew = bnPowLimit;
```

注意：虽然目标校准每2,016个块发生，但是由于Bitcoin Core客户端的一个错误，它是基于之前的2,015个块的总时间（不应该是2,016个），导致重定向偏差向较高难度提高0.05%。

参数Interval(2,016区块)和TargetTimespan(1,209,600秒即两周) 的定义在文件chainparams.cpp中。

为了防止难度的变化过快，每个周期的调整幅度必须小于一个因子（值为4）。如果要调整的幅度大于4倍，则按4倍调 整。由于在下一个2,016区块的周期不平衡的情况会继续存在，所以进一步的难度调整会在下一周期进行。因此平衡哈 希计算能力和难度的巨大差异有可能需要花费几个2,016区块周期才会完成。

提示：寻找一个比特币区块需要整个网络花费10分钟来处理，每发现2,016个区块时会根据前2,016个区块完成的时间对难度进行调整。

值得注意的是目标难度与交易的数量和金额无关。这意味着哈希算力的强弱，即让比特币更安全的电力投入量，与交易 的数量完全无关。换句话说，当比特币的规模变得更大，使用它的人数更多时，即使哈希算力保持当前的水平，比特币 的安全性也不会受到影响。哈希算力的增加表明更多的人为得到比特币回报而加入了挖矿队伍。只要为了回报，公平正 当地从事挖矿的矿工群体保持足够的哈希算力，"接管"攻击就不会得逞，让比特币的安全无虞。

目标难度和挖矿电力消耗与将比特币兑换成现金以支付这些电力之间的关系密切相关。高性能挖矿系统就是要用当前硅 芯片以最高效的方式将电力转化为哈希算力。挖矿市场的关键因素就是每度电转换为比特币后的价格。因为这决定着挖 矿活动的营利性，也因此刺激着人们选择进入或退出挖矿市场。

10.8 成功构建区块

前面已经看到，Jing的节点创建了一个候选区块，准备拿它来挖矿。Jing有几个安装了ASIC（专用集成电路）的矿机， 上面有成千上万个集成电路可以超高速地并行运行SHA256算法。这些定制的硬件通过USB连接到他的挖矿节点上。接 下来，运行在Jing的桌面电脑上的挖矿节点将区块头信息传送给这些硬件，让它们以每秒亿万次的速度进行nonce测 试。

在对区块277,316的挖矿工作开始大概11分钟后，这些硬件里的其中一个求得了解并发回挖矿节点。当把这个结果放进 区块头时， nonce 4,215,469,401 就会产生一个区块哈希值：

```
0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569
```

而这个值小于难度目标值：

```
00000000000000003A30C00000000000000000000000000000000000000000000
```

Jing的挖矿节点立刻将这个区块发给它的所有相邻节点。这些节点在接收并验证这个新区块后，也会继续传播此区块。当这个新区块在网络中扩散时，每个节点都会将它作为区块277,316加到自身节点的区块链副本中。当挖矿节点收到并验证了这个新区块后，它们会放弃之前对构建这个相 同高度区块的计算，并立即开始计算区块链中下一个区块的工作。

下节将介绍节点进行区块验证、最长链选择、达成共识，并以此形成一个去中心化区块链的过程。

10.9 校验新区块

比特币共识机制的第三步是通过网络中的每个节点独立校验每个新区块。当新区块在网络中传播时，每一个节点在将它 转发到其节点之前，会进行一系列的测试去验证它。这确保了只有有效的区块会在网络中传播。独立校验还确保了诚实 的矿工生成的区块可以被纳入到区块链中，从而获得奖励。行为不诚实的矿工所产生的区块将被拒绝，这不但使他们失 去了奖励，而且也浪费了本来可以去寻找工作量证明解的机会，因而导致其电费亏损。

当一个节点接收到一个新的区块，它将对照一个长长的标准清单对该区块进行验证，若没有通过验证，这个区块将被拒 绝。这些标准可以在比特币核心客户端的CheckBlock函数和CheckBlockHead函数中获得，

它包括：

▷ 区块的数据结构语法上有效

▷ 区块头的哈希值小于目标难度（确认包含足够的工作量证明）

▷ 区块时间戳早于验证时刻未来两个小时（允许时间错误）

▷ 区块大小在长度限制之内

▷ 第一个交易（且只有第一个）是coinbase交易

▷ 使用检查清单验证区块内的交易并确保它们的有效性

参见之前章节 “交易的独立校验”一节已经讨论过这个清单。每一个节点对每一个新区块的独立校验，确保了矿工无法欺诈。在前面的章节中，我们看到了矿工们如何去记录一笔交 易，以获得在此区块中创造的新比特币和交易费。为什么矿工不为他们自己记录一笔交易去获得数以千计的比特币？这 是因为每一个节点根据相同的规则对区块进行校验。一个无效的coinbase交易将使整个区块无效，这将导致该区块被拒 绝，因此，该交易就不会成为总账的一部分。矿工们必须构建一个完美的区块，基于所有节点共享的规则，并且根据正 确工作量证明的解决方案进行挖矿，他们要花费大量的电力挖矿才能做到这一点。如果他们作弊，所有的电力和努力都 会浪费。这就是为什么独立校验是去中心化共识的重要组成部分。

10.10 区块链的组装与选择

比特币去中心化的共识机制的最后一步是将区块集合至有最大工作量证明的链中。一旦一个节点验证了一个新的区块， 它将尝试将新的区块连接到到现存的区块链，将它们组装起来。

节点维护三种区块：第一种是连接到主链上的，第二种是从主链上产生分支的（备用链），最后一种是在已知链中没有 找到已知父区块的。在验证过程中，一旦发现有不符合标准的地方，验证就会失败，这样区块会被节点拒绝，所以也不 会加入到任何一条链中。

任何时候，主链都是累计了最多难度的区块链。在一般情况下，主链也是包含最多区块的那个链，除非有两个等长的链 并且其中一个有更多的工作量证明。主链也会有一些分支，这些分支中的区块与主链上的区块互为“兄弟”区块。这些区 块是有效的，但不是主链的一部分。 保留这些分支的目的是如果在未来的某个时刻它们中的一个延长了并在难度值上超 过了主链，那么后续的区块就会引用它们。在“10.10.1 区块链分叉”，我们将会看到在同样的区块高度，几乎同时挖出区 块时，候选链是如何产生的。

当节点接收到新区块，它会尝试将这个区块插入到现有区块链中。节点会看一下这个区块的“previous block hash”字 段，这个字段是该区块对其父区块的引用。同时，新的节点将尝试在已存在的区块链中找出这个父区块。大多数情况 下，父区块是主块链的“顶点”，这就意味着这个新的区块延长了主链。举个例子，一个新的区块——区块277,316引用 了它的父区块——区块277,315。收到277316区块的大部分节点都已经将277315最为主链的顶端，因此，连接这个新区块并延长区块链。

有时候，新区块所延长的区块链并不是主链，这一点我们将在“10.10.1 区块链分叉”中看到。在这种情况下，节点将新的区块添加到备用链，同时比较备用链与主链的难度。如果备用链比主链积累了更多的难度，节点将收敛于备用链，意味 着节点将选择备用链作为其新的主链，而之前那个老的主链则成为了备用链。如果节点是一个矿工，它将开始构造新的 区块，来延长这个更新更长的区块链。

如果节点收到了一个有效的区块，而在现有的区块链中却未找到它的父区块，那么这个区块被认为是“孤块”。孤块会被 保存在孤块池中，直到它们的父区块被节点收到。一旦收到了父区块并且将其连接到现有区块链上，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。当两个区块在很短的时间间隔内被挖出来，节点有 可能会以相反的顺序接收到它们，这个时候孤块现象就会出现。

选择了最大难度的区块链后，所有的节点最终在全网范围内达成共识。随着更多的工作量证明被添加到链中，链的暂时 性差异最终会得到解决。挖矿节点通过“投票”来选择它们想要延长的区块链，当它们挖出一个新块并且延长了一个链， 新块本身就代表它们的投票。

相互竞争的链之间是存在差异的，下节我们将看到节点是怎样通过独立选择最长难度链来解决这种差异的。

10.10.1 区块链分叉

因为区块链是去中心化的数据结构，所以不同副本之间不能总是保持一致。区块有可能在不同时间到达不同节点，导致节点有不同的区块链全貌。解决的办法是，每一个节点总是选择并尝试延长代表累计了最大工作量证明的区块链，也就 是最长的或最大累计工作的链（greatest cumulative work chain）。节点通过累加链上的每个区块的工作量，得到建立这个链所要付出的工作量证明的总量。只要所有的节点选择最长累计工作的区块链，整个比特币网络最终会收敛到一致的状态。分叉即在不同区块链间发生的临时差异，当更多的区块添加到了某个分叉中，这个问题便会迎刃而解。

提示由于全球网络中的传输延迟，本节中描述的区块链分叉自动会发生。 我们也将在本章稍后再看看故意引起的分叉。

在接下来的几个图表中，我们将通过网络跟踪“fork”事件的进展。 该图是比特币网络的简化表示。 为了便于描述，不同的区块被显示为不同的形状（星形，三角形，倒置三角形，菱形），遍布网络。 网络中的每个圆表示一个节点。

每个节点都有自己的全局区块链视图。 当每个节点从其邻居接收区块时，它会更新其自己的区块链副本，选择最大累积工作链。 为便于描述，每个节点包含一个图形形状，表示它相信的区块处于主链的顶端。 因此，如果在节点里面看到星形，那就意味着该节点认为星形区块处于主链的顶端。

在第一张图（图10-2）中，网络有一个统一的区块链视角，以星形区块为主链的“顶点”。

图10-2

当有两个候选区块同时想要延长最长区块链时，分叉事件就会发生。正常情况下，分叉发生在两名矿工在较短的时间 内，各自都算得了工作量证明解的时候。两个矿工在各自的候选区块一发现解，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。如果该节点在 随后又收到了另一个候选区块，而这个区块又拥有同样父区块，那么节点会将这个区块连接到候选链上。其结果是，一 些节点收到了一个候选区块，而另一些节点收到了另一个候选区块，这时两个不同版本的区块链就出现了。在图10-3中，我们看到两个矿工（NodeX和NodeY）几乎同时挖到了两个不同的区块。这两个区块是顶点区块——星形区块的子区块，可以 延长这个区块链。为了方便查看，我们把节点X产生的区块标记为三角形，把节点Y生产的区块标记为倒三角形。

图10-3

例如，我们假设矿工节点X找到扩展区块链工作量证明的解，即三角形区块，构建在星形父区块的顶端。与此同时，同样进行星形区块扩展的节点Y也找到了扩展区块链工作量证明的解，即倒三角形区块作为候选区块。现在有两个可能的块，节点X的三角形区块和节点Y的倒三角形区块，这两个区块都是有效的，均包含有效的工作量证明解并延长同一 个父区块。这个两个区块可能包含了几乎相同的交易，只是在交易的排序上有些许不同。

当两个区块开始在网络传播时，一些节点首先接收到三角形区块，另外一些节点首先接收倒三角形区块。如下图10-4所示，比特币网络上的节点对于区块链的顶点产生了分歧，一派以三角形区块为顶点，而另一派以倒三角形区块为顶点。

图10-4

在图中，假设节点X首先接收到三角形块，并用它扩展星形链。节点X选择三角形区块为主链。之后，节点X也收到倒三角区块。由于是第二次收到，因此它判定这个倒三角形区块是竞争失败的产物，认为是无效区块。然而，倒三角形的区块不会被丢弃。它被链接到星形链的父区块，并形成备用链。虽然节点X认为自己已经正确选择了获胜链，但是它还会保存“丢失”链，使得“丢失”链如果可能最终“获胜”，它还具有重新打包的所需的信息。

在网络的另一端，节点Y根据自己的视角构建一个区块链。首先获得倒三角形区块，并选择这条链作为“赢家”。当它稍后收到三角形区块时，它也将三角形区块连接到星形链的父区块作为备用链。

双方都是“正确的”或“不正确的”。两者都是自己关于区块链的有效立场。只有事后，才能理解这两个竞争链如何通过额外的工作得到延伸。

节点X阵营的其他节点将立即开始挖掘候选区块，以“三角形”作为扩展区块链的顶端。通过将三角形作为候选区块的父区块，它们用自己的哈希算力进行投票。它们的投票标明支持自己选择的链为主链。

同样，节点Y阵营的其他节点，将开始构建一个以倒三角形作为其父节点的候选节点，扩展它们认为是主链的链。比赛再次开始。分叉问题几乎总是在一个区块内就被解决了。网络中的一部分算力专注于“三角形”区块为父区块，在其之上建立新的区 块；另一部分算力则专注在“倒三角形”区块上。即便算力在这两个阵营中平均分配，也总有一个阵营抢在另一个阵营前发现工作量证明解并将其传播出去。在这个例子中我们可以打个比方，假如工作在“三角形”区块上的矿工找到了一个“菱形”区块 延长了区块链(星形-三角形-菱形)，他们会立刻传播这个新区块，整个网络会都会认为这个区块是有效的，如下图10-5所示。

图10-5

选择“三角形”作为上一轮中胜出者的所有节点将简单地将区块链扩展一个块。然而，选择“倒三角”的节点现在将看到两个链：星形-三角形-菱形和星型-到三角形。星形-三角形-菱形这条链现在比其他链条更长（更多累积的工作）。因此，这些节点将星形-三角形-菱形设置为主链，并将星型-倒三角形链变为备用链，如图10-6所示。这是一个链的重新共识，因为这些节点被迫修改他们对块链的立场，把自己纳入更长的链。任何从事延伸星形-倒三角形的矿工现在都将停止这项工作，因为他们的候选人是“孤儿”，因为他们的父母“倒三角形”不再是最长的连锁。 “倒三角形”内的交易重新插入到内存池中用来包含在下一个块中，因为它们所在的块不再位于主链中。整个网络重新回到单一链状态，星形-三角形-菱形，“菱形”成为链中的最后一个块。所有矿工立即开始研究以“菱形”为父区块的候选块，以扩展这条星形-三角形-菱形链。

图10-6

从理论上来说，两个区块的分叉是有可能的，这种情况发生在因先前分叉而相互对立起来的矿工，又几乎同时发现了两 个不同区块的解。然而，这种情况发生的几率是很低的。单区块分叉每周都会发生，而双块分叉则非常罕见。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 易清算更快地完成，也会导致更加频繁地区块链分叉。与之相对地，更长的间隔会减少分叉数量，却会导致更长的清算时间。

10.11 挖矿和算力竞赛

比特币挖矿是一个极富竞争性的行业。自从比特币存在开始，每年比特币算力都成指数增长。一些年份的增长还体现出技术的变革，比如在2010年和2011年，很多矿工开始从使用CPU升级到使用GPU，进而使用FGPA（现场可编程门阵 列）挖矿。在2013年，ASIC挖矿的引入，把SHA256算法直接固化在挖矿专用的硅芯片上，引起了算力的另一次巨大飞 跃。一台采用这种芯片的矿机可以提供的算力，比2010年比特币网络的整体算力还要大。

下表表示了比特币网络开始运行后最初五年的总算力：

2009 0.5 MH/sec–8 MH/sec (16× growth)

2010 8 MH/sec–116 GH/sec (14,500× growth)

2011 16 GH/sec–9 TH/sec (562× growth)

2012 9 TH/sec–23 TH/sec (2.5× growth)

2013 23 TH/sec–10 PH/sec (450× growth)

2014 10 PH/sec–300 PH/sec (3000× growth)

2015 300 PH/sec-800 PH/sec (266× growth)

2016 800 PH/sec-2.5 EH/sec (312× growth))

在图10-7的图表中，我们可以看到近两年里，矿业和比特币的成长引起了比特币网络算力的指数增长（每秒网络总算 力）。

随着比特币挖矿算力的爆炸性增长，与之匹配的难度也相应增长。图10-8中的相对难度值显示了当前难度与最小难度 （第一个块的难度）的比例。

近两年，ASIC芯片变得更加密集，特征尺寸接近芯片制造业前沿的16纳米。挖矿的利润率驱动这个行业以比通用计算 更快的速度发展。

目前，ASIC制造商的目标是超越通用CPU芯片制造商，设计特征尺寸为14纳米的芯片。对比特币挖矿而言，已经没有更多飞跃的空间，因为这个行业已经触及了摩尔定律的最前沿。摩尔定律指出计算能力每18个月增加一倍。

尽管如此，随着更高密度的芯片和数据中心的部署竞赛，网络算力继续保持同步的指数增长。现在的竞争已经不再是比较单一芯片的能力，而是一个矿场能塞进多少芯片，并处理好散热和供电问题。

10.11.1 随机值升位方案 the extra nonce solution

2012年以来，比特币挖矿发展出一个解决区块头基本结构限制的方案。在比特币的早期，矿工可以通过遍历随机数 (Nonce)获得符合要求的hash来挖出一个块。

难度增长后，矿工经常在尝试了40亿个值后仍然没有出块。然而，这很容 易通过读取块的时间戳并计算经过的时间来解决。因为时间戳是区块头的一部分，它的变化可以让矿工用不同的随机值 再次遍历。当挖矿硬件的速度达到了4GH/秒，这种方法变得越来越困难，因为随机数的取值在一秒内就被用尽了。

当出现ASIC矿机并很快达到了TH/秒的hash速率后，挖矿软件为了找到有效的块，需要更多的空间来储存nonce值。可以把 时间戳延后一点，但将来如果把它移动得太远，会导致区块变为无效。

区块头需要信息来源的一个新的“变革”。解决方案是使用coinbase交易作为额外的随机值来源，因为coinbase脚本可以储存2-100字节的数据，矿工们开始使用这个空间作为额外随机值的来源，允许他们去探索一个大得多的区块头值范围来找到有效的块。这个coinbase交易包含在merkle树中，这意味着任何coinbase脚本的变化将导致Merkle根的变化。

8个字节的额外随机数，加上4个字节的“标准”随机数，允许矿工每秒尝试2^96（8后面跟28个零）种可能性而无需修改时间戳。如果未来矿工穿过了以上所有的可能性，他们还可以通过修改时间戳来解决。同样，coinbase脚本中也有更多额外的空间可以为将来随机数的扩展做准备。

10.11.2 矿池

在这个激烈竞争的环境中，个体矿工独立工作（也就是solo挖矿）没有一点机会。他们找到一个区块以抵消电力和硬件 成本的可能性非常小，以至于可以称得上是赌博，就像是买彩票。就算是最快的消费型ASIC也不能和那些在巨大机房里拥有数万芯片并靠近水电站的商业矿场竞争。

现在矿工们合作组成矿池，汇集数以千计参与者们的算力并分享奖励。通 过参加矿池，矿工们得到整体回报的一小部分，但通常每天都能得到，因而减少了不确定性。

让我们来看一个具体的例子。假设一名矿工已经购买了算力共计14,000GH/S，或14TH/S的设备，在2017年，它的价 值大约是2500美元。

该设备运行功率为1.3千瓦（KW），每日耗电32度，每日平均成本1或2美元。以目前的比特币难度， 该矿工solo方式挖出一个块平均需要4年。如果这个矿工确实在这个时限内挖出一个区块，奖励12.5个比特币，如果 每个比特币价格约为1000美元（译者：这是作者出版此书的价格，2017年10月22日译者看到的价格是5880美元），可以得到12,500美元的收入。这甚至不能覆盖整个硬件和整个时间段的电力消耗，净亏损约为1,000美元。而且，在4年的时间周期内能否挖出一个块还主要靠矿工的运气。他有可能在4年中得到两个块从而赚到非常大的利润。或者，他可能5年都找不到一个块，从而遭受经济损失。

更糟的是，比特币的工作证明 （POW）算法的难度可能在这段时间内显著上升，按照目前算力增长的速度，这意味着矿工在设备被下一代更有效率的矿机取代之前，最多有1年的时间取得成果。如果这个矿工加入矿池，而不是等待4年内可能出现一次的暴利，他每周能赚取大约50-60美元。矿池的常规收入能帮他随时间摊销硬件和电力的成本，并且不用承担巨大的风险。在1-2年后，硬件仍然会过时，风险仍然很高，但在此期间的收入至少是定期的和可靠的。

从财务数据分析，这只有非常低的电力成本（每千瓦不到1美分），非常大的规模时才有意义。矿池通过专用挖矿协议协调成百上千的矿工。

个人矿工在建立矿池账号后，设置他们的矿机连接到矿池服务器。他们的 挖矿设备在挖矿时保持和矿池服务器的连接，和其他矿工同步各自的工作。这样，矿池中的矿工分享挖矿任务，之后分 享奖励。成功出块的奖励支付到矿池的比特币地址，而不是单个矿工的。一旦奖励达到一个特定的阈值，矿池服务器便会定期支 付奖励到矿工的比特币地址。

通常情况下，矿池服务器会为提供矿池服务收取一个百分比的费用。参加矿池的矿工把搜寻候选区块的工作量分割，并根据他们挖矿的贡献赚取“份额”。矿池为赚取“份额”设置了一个低难度 的目标，通常比比特币网络难度低1000倍以上。

当矿池中有人成功挖出一块，矿池获得奖励，并和所有矿工按照他们做 出贡献的“份额”数的比例分配。矿池对任何矿工开放，无论大小、专业或业余。一个矿池的参与者中，有人只有一台小矿机，而有些人有一车库高端挖 矿硬件。有人只用几十度电挖矿，也有人会用一个数据中心消耗兆瓦级的电量。矿池如何衡量每个人的贡献，既能公平分配奖励，又避免作弊的可能？答案是在设置一个较低难度的前提下，使用比特币的工作量证明算法来衡量每个矿工的 贡献。

因此，即使是池中最小的矿工也经常能分得奖励，这足以激励他们为矿池做出贡献。通过设置一个较低的取得份 额的难度，矿池可以计量出每个矿工完成的工作量。每当矿工发现一个小于矿池难度的区块头hash，就证明了它已经完 成了寻找结果所需的hash计算。更重要的是，这些为取得份额贡献而做的工作，能以一个统计学上可衡量的方法，整体 寻找一个比特币网络的目标散列值。成千上万的矿工尝试较小区间的hash值，最终可以找到符合比特币网络要求的结 果。

让我们回到骰子游戏的比喻。如果骰子玩家的目标是扔骰子结果都小于4（整体网络难度），一个矿池可以设置一个更 容易的目标，统计有多少次池中的玩家扔出的结果小于8。当池中的玩家扔出的结果小于8（矿池份额目标），他们得到份额，但他们没有赢得游戏，因为没有完成游戏目标（小于4）。但池中的玩家会更经常的达到较容易的矿池份额目 标，规律地赚取他们的份额，尽管他们没有完成更难的赢得比赛的目标。时不时地，池中的一个成员有可能会扔出一个小于4的结果，矿池获胜。然后，收益可以在池中玩家获得的份额基础上分配。

尽管目标设置为8或更少并没有赢得游戏，但是这是一个衡量玩家们扔出的点数的公平方法，同时它偶尔会产生一个小于4的结果。同样的，一个矿池会将矿池难度设置在保证一个单独的矿工能够频繁地找到一个符合矿池难度的区块头hash来赢取份 额。时不时的，某次尝试会产生一个符合比特币网络目标的区块头hash，产生一个有效块，然后整个矿池获胜。

10.11.2.1 托管矿池

大部分矿池是“托管的”，意思是有一个公司或者个人经营一个矿池服务器。矿池服务器的所有者叫矿池管理员，同时他 从矿工的收入中收取一个百分比的费用。矿池服务器运行专业软件以及协调池中矿工们活动的矿池采矿协议。矿池服务器同时也连接到一个或更多比特币完全节 点并直接访问一个块链数据库的完整副本。这使得矿池服务器可以代替矿池中的矿工验证区块和交易，缓解他们运行一 个完整节点的负担。

对于池中的矿工，这是一个重要的考量，因为一个完整节点要求一个拥有最少100-150GB的永久储存空间（磁盘）和最少2GB到4GB内存（RAM）的专用计算机。

此外，运行一个完整节点的比特币软件需要监控、维护和频繁升级。由于缺乏维护或资源导致的任何宕机都会伤害到矿工的利润。对于很多矿工来说，不需要跑一个完整节点就能采矿，也是加入托管矿池的一大好处。

矿工连接到矿池服务器使用一个采矿协议比如Stratum (STM)或者 GetBlockTemplate (GBT)。一个旧标准GetWork (GWK) 自从2012年底已经基本上过时了，因为它不支持在hash速度超过4GH/S时采矿。STM和GBT协议都创建包含候选区块头模板的区块模板。矿池服务器通过打包交易，添加coinbase交易（和额外的随机值空间），计算MERKLE根， 并连接到上一个块hash来建立一个候选区块。这个候选区块的头部作为模板分发给每个矿工。矿工用这个区块模板在低于比特币网络的难度下采矿，并发送成功的结果返回矿池服务器赚取份额。

10.11.2.2 P2P矿池

托管矿池存在管理人作弊的可能，管理人可以利用矿池进行双重支付或使区块无效。（参见“10.12 共识攻击”） 此外，中 心化的矿池服务器代表着单点故障。如果因为拒绝服务攻击服务器挂了或者被减慢，池中矿工就不能采矿。

在2011年， 为了解决由中心化造成的这些问题，提出和实施了一个新的矿池挖矿方法。P2Pool是一个点对点的矿池，没有中心管理 人。P2Pool通过将矿池服务器的功能去中心化，实现一个并行的类似区块链的系统，名叫份额链（share chain）。

一个份额链是一个难度低 于比特币区块链的区块链系统。份额链允许池中矿工在一个去中心化的池中合作，以每30秒一个份额区块的速度在份额 链上采矿，并获得份额。份额链上的区块记录了贡献工作的矿工的份额，并且继承了之前份额区块上的份额记录。当一 个份额区块上还实现了比特币网络的难度目标时，它将被广播并包含到比特币的区块链上，并奖励所有已经在份额链区 块中取得份额的池中矿工。

本质上说，比起用一个矿池服务器记录矿工的份额和奖励，份额链允许所有矿工通过类似比特币区块链系统的去中心化的共识机制跟踪所有份额。P2Pool采矿方式比在矿池中采矿要复杂的多，因为它要求矿工运行空间、内存、带宽充足的专用计算机来支持一个比特 币的完整节点和P2Pool节点软件。P2Pool矿工连接他们的采矿硬件到本地P2Pool节点，它通过发送区块模板到矿机来 模拟一个矿池服务器的功能。在P2Pool中，单独的矿工创建自己的候选区块，打包交易，非常类似于solo矿工，但是他 们在份额链上合作采矿。

P2Pool是一种比单独挖矿有更细粒度收入优势的混合方法。但是不需要像托管矿池那样给管理 人太多权力。即使P2Pool减少了采矿池运营商的中心化程度，但也可能容易受到份额链本身的51％的攻击。 P2Pool的广泛采用并不能解决比特币本身的51％攻击问题。 相反，作为多样化采矿生态系统的一部分，P2Pool整体使得比特币更加强大。

10.12 共识攻击

比特币的共识机制指的是，被矿工（或矿池）试图使用自己的算力实行欺骗或破坏的难度很大，至少理论上是这样。就像我们前面讲的，比特币的共识机制依赖于这样一个前提，那就是绝大多数的矿工，出于自己利益最大化的考虑，都会 通过诚实地挖矿来维持整个比特币系统。然而，当一个或者一群拥有了整个系统中大量算力的矿工出现之后，他们就可 以通过攻击比特币的共识机制来达到破坏比特币网络的安全性和可靠性的目的。

值得注意的是，共识攻击只能影响整个区块链未来的共识，或者说，最多能影响不久的过去几个区块的共识（最多影响 过去10个块）。而且随着时间的推移，整个比特币块链被篡改的可能性越来越低。

理论上，一个区块链分叉可以变得很 长，但实际上，要想实现一个非常长的区块链分叉需要的算力非常非常大，随着整个比特币区块链逐渐增长，过去的区 块基本可以认为是无法被分叉篡改的。同时，共识攻击也不会影响用户的私钥以及加密算法（ECDSA）。

共识攻击也 不能从其他的钱包那里偷到比特币、不签名地支付比特币、重新分配比特币、改变过去的交易或者改变比特币持有纪 录。共识攻击能够造成的唯一影响是影响最近的区块（最多10个）并且通过拒绝服务来影响未来区块的生成。共识攻击的一个典型场景就是“51%攻击”。想象这么一个场景，一群矿工控制了整个比特币网络51%的算力，他们联合 起来打算攻击整个比特币系统。由于这群矿工可以生成绝大多数的块，他们就可以通过故意制造块链分叉来实现“双重支 付”或者通过拒绝服务的方式来阻止特定的交易或者攻击特定的钱包地址。

区块链分叉/双重支付攻击指的是攻击者通过 不承认最近的某个交易，并在这个交易之前重构新的块，从而生成新的分叉，继而实现双重支付。有了充足算力的保 证，一个攻击者可以一次性篡改最近的6个或者更多的区块，从而使得这些区块包含的本应无法篡改的交易消失。值得注意的是，双重支付只能在攻击者拥有的钱包所发生的交易上进行，因为只有钱包的拥有者才能生成一个合法的签名用 于双重支付交易。攻击者在自己的交易上进行双重支付攻击，如果可以通过使交易无效而实现对于不可逆转的购买行为不予付款， 这种攻击就是有利可图的。

让我们看一个“51%攻击”的实际案例吧。在第1章我们讲到，Alice 和 Bob 之间使用比特币完成了一杯咖啡的交易。咖啡 店老板 Bob 愿意在 Alice 给自己的转账交易确认数为零的时候就向其提供咖啡，这是因为这种小额交易遭遇“51%攻 击”的风险和顾客购物的即时性（Alice 能立即拿到咖啡）比起来，显得微不足道。这就和大部分的咖啡店对低于25美元 的信用卡消费不会费时费力地向顾客索要签名是一样的，因为和顾客有可能撤销这笔信用卡支付的风险比起来，向用户 索要信用卡签名的成本更高。

相应的，使用比特币支付的大额交易被双重支付的风险就高得多了，因为买家（攻击者） 可以通过在全网广播一个和真实交易的UTXO一样的伪造交易，以达到取消真实交易的目的。双重支付可以有两种方 式：要么是在交易被确认之前，要么攻击者通过块链分叉来完成。进行51%攻击的人，可以取消在旧分叉上的交易记 录，然后在新分叉上重新生成一个同样金额的交易，从而实现双重支付。

再举个例子：攻击者Mallory在Carol的画廊买了描绘伟大的中本聪的三联组画（The Great Fire），Mallory通过转账价值25万美金的比特币 与Carol进行交易。在等到一个而不是六个交易确认之后，Carol放心地将这幅组画包好，交给了Mallory。这时，Mallory 的一个同伙，一个拥有大量算力的矿池的人Paul，在这笔交易写进区块链的时候，开始了51%攻击。

首先，Paul利用自己矿池的算力重新计算包含这笔交易的块，并且在新块里将原来的交易替换成了另外一笔交易（比如直接转给了Mallory 的另一个钱包而不是Carol的），从而实现了“双重支付”。这笔“双重支付”交易使用了跟原有交易一致的UTXO，但收款人被替换成了Mallory的钱包地址。

然后，Paul利用矿池在伪造的块的基础上，又计算出一个更新的块，这样，包含这 笔“双重支付”交易的块链比原有的块链高出了一个块。到此，高度更高的分叉区块链取代了原有的区块链，“双重支付”交 易取代了原来给Carol的交易，Carol既没有收到价值25万美金的比特币，原本拥有的三幅价值连城的画也被Mallory白白 拿走了。

在整个过程中，Paul矿池里的其他矿工可能自始至终都没有觉察到这笔“双重支付”交易有什么异样，因为挖矿程序都是自动在运行，并且不会时时监控每一个区块中的每一笔交易。

为了避免这类攻击，售卖大宗商品的商家应该在交易得到全网的6个确认之后再交付商品。或者，商家应该使用第三方 的多方签名的账户进行交易，并且也要等到交易账户获得全网多个确认之后再交付商品。一条交易的确认数越多，越难 被攻击者通过51%攻击篡改。

对于大宗商品的交易，即使在付款24小时之后再发货，对买卖双方来说使用比特币支付也 是方便并且有效率的。而24小时之后，这笔交易的全网确认数将达到至少144个（能有效降低被51%攻击的可能性）。共识攻击中除了"双重支付"攻击，还有一种攻击场景就是拒绝对某个特定的比特币地址提供服务。一个拥有了系统中绝 大多数算力的攻击者，可以轻易地忽略某一笔特定的交易。如果这笔交易存在于另一个矿工所产生的区块中，该攻击者 可以故意分叉，然后重新产生这个区块，并且把想忽略的交易从这个区块中移除。这种攻击造成的结果就是，只要这名 攻击者拥有系统中的绝大多数算力，那么他就可以持续地干预某一个或某一批特定钱包地址产生的所有交易，从而达到拒绝为这些地址服务的目的。

需要注意的是，51%攻击并不是像它的命名里说的那样，攻击者需要至少51%的算力才能发起，实际上，即使其拥有不 到51%的系统算力，依然可以尝试发起这种攻击。之所以命名为51%攻击，只是因为在攻击者的算力达到51%这个阈值 的时候，其发起的攻击尝试几乎肯定会成功。

本质上来看，共识攻击，就像是系统中所有矿工的算力被分成了两组，一 组为诚实算力，一组为攻击者算力，两组人都在争先恐后地计算块链上的新块，只是攻击者算力算出来的是精心构造 的、包含或者剔除了某些交易的块。因此，攻击者拥有的算力越少，在这场决逐中获胜的可能性就越小。

从另一个角度 讲，一个攻击者拥有的算力越多，其故意创造的分叉块链就可能越长，可能被篡改的最近的块或者或者受其控制的未来 的块就会越多。一些安全研究组织利用统计模型得出的结论是，算力达到全网的30%就足以发动51%攻击了。全网算力的急剧增长已经使得比特币系统不再可能被某一个矿工攻击，因为一个矿工已经不可能占据全网哪怕的1%算 力。

但是中心化控制的矿池则引入了矿池操作者出于利益而施行攻击的风险。矿池操作者控制了候选块的生成，同时也控制哪些交易会被放到新生成的块中。这样一来，矿池操作者就拥有了剔除特定交易或者双重支付的权力。如果这种权 利被矿池操作者以微妙而有节制的方式滥用的话，那么矿池操作者就可以在不为人知的情况下发动共识攻击并获益。但是，并不是所有的攻击者都是为了利益。

一个可能的场景就是，攻击者仅仅是为了破坏整个比特币系统而发动攻击， 而不是为了利益。这种意在破坏比特币系统的攻击者需要巨大的投入和精心的计划，因此可以想象，这种攻击很有可能 来自政府资助的组织。同样的，这类攻击者或许也会购买矿机，运营矿池，通过滥用矿池操作者的上述权力来施行拒绝 服务等共识攻击。但是，随着比特币网络的算力呈几何级数快速增长，上述这些理论上可行的攻击场景，实际操作起来 已经越来越困难。

近期比特币系统的一些升级，比如旨在进一步将挖矿控制去中心化的P2Pool挖矿协议，也都正在让这 些理论上可行的攻击变得越来越困难。毫无疑问，一次严重的共识攻击事件势必会降低人们对比特币系统的信心，进而可能导致比特币价格的跳水。然而，比 特币系统和相关软件也一直在持续改进，所以比特币社区也势必会对任何一次共识攻击快速做出响应，以使整个比特币 系统比以往更加稳健和可靠。

10.13改变共识规则

共识规则确定交易和块的有效性。 这些规则是所有比特币节点之间协作的基础，并且负责将所有不同角色的本地视角融合到整个网络中的单一一致的区块链中。虽然共识规则在短期内是不变的，并且在所有节点之间必须一致，但长期来看它们并不总是不变的。 为了演进和开发比特币系统，规则必须随时改变以适应新功能，改进或修复错误。然而，与传统软件开发不同，升级到共识系统要困难得多，需要所有参与者之间的协调。

10.13.1硬分叉

在前面章节比特币分叉中，我们研究了比特币网络如何短暂地分叉，网络中的两个部分在短时间内处于区块链的两个不同分支。我们看到这个过程是如何自然发生的，作为网络的正常运行的一部分，以及如何在一个或多个块被挖掘之后，网络在一个统一的区块链上重新收敛。

另一种情况是，网络也可能会分叉到两条链条，这是由于共识规则的变化。这种分叉称为硬分叉，因为这种分叉后，网络不会重新收敛到单个链路上。相反，这两条链子独立发展。当比特币网络的一部分节点按照与网络的其余部分节点不同的一致性规则运行时，硬分叉就会发生。

这可能是由于错误或者由于故意改变了协商一致规则的实施而发生的。硬分叉可用于改变共识的规则，但需要在系统中所有参与者之间进行协调。没有升级到新的共识规则的任何节点都不能参与共识机制，并且在硬分叉时刻被强制到单独的链上。

因此，硬分叉引入的变化可以被认为**不是**"向前兼容"，因为未升级的系统不能再处理新的共识规则。让我们来看一下硬分叉的技巧和具体的例子。下图10-9显示区块链出现两个分叉。在块高度4处，发生单一个区块分叉。这是我们在前面章节比特币分叉中看到的自发分叉的类型。经过块5的挖掘，网络在一条链上重新收敛，分叉被解决。

然而，后来在块高度6处发生了硬分叉。我们假设原因是由于新共识规则的变化出现新的客户端版本。从块高度7开始，矿工运行新的版本，需要接受新类型的数字签名，我们称之为"Smores"签名，它不是基于ECDSA的签名。紧接着，运行新版本的节点创建了一笔包含Smores签名的交易，一个更新了软件的矿工挖矿出了包含此交易的区块7b。

任何尚未升级软件以验证Smores签名的节点或矿工现在都无法处区块7b。从他们的角度来看，包含Smores签名的交易和包含该交易的块7b的交易都是无效的，因为它们是根据旧的共识规则进行评估的。

这些节点将拒绝该笔交易和区块，并且不会传播它们。正在使用旧规则的任何矿工都不接受块7b，并且将继续挖掘其父级为块6的候选块。实际上，使用如果它们连接的所有节点都是也遵守旧的规则，遵守旧规则的矿工甚至可能接收不到块7b，因此不会传播这个区块。最终，他们将能够开采区块7a，这个旧的规则是有效的，其中不包含与Smores签名的任何交易。

这两个链条从这一点继续分歧。 "b"链的矿工将继续接受并开采含有Smores签名的交易，而"a"链上的矿工将继续忽视这些交易。即使块8b不包含任何Smores签名的交易，"a"链上的矿工也无法处理。对他们来说，它似乎是一个孤立的块，因为它的父"7b"不被识别为一个有效的块。

10.13.2硬分叉：软件，网络，采矿和链

对于软件开发人员来说，术语"分叉"具有另一个含义，对"硬分叉"一词增加了混淆。在开源软件中，当一组开发人员选择遵循不同的软件路线图并启动开源项目的竞争实施时，会发生叉。我们已经讨论了两种情况，这将导致硬分叉：共识规则中的错误，以及对共识规则的故意修改。在故意改变共识规则的情况下，软件叉在硬分叉之前。但是，对于这种类型的硬分叉，新的共识规则必须通过开发，采用和启动新的软件实现。

试图改变共识规则的软分叉的例子包括Bitcoin XT，Bitcoin Classic和最近的Bitcoin Unlimited。但是，这些软分叉都没有产生硬分叉。虽然软件叉是一个必要的前提条件，但它本身不足以发生硬分叉。对于硬分叉发生，必须是由于采取相互竞争的实施方案，并且规则需要由矿工，钱包和中间节点激活。相反，有许多比特币核心的替代实现方案，甚至还有软分叉，这些没有改变共识规则，阻止发生错误，可以在网络上共存并互操作，最终并未导致硬分叉。

不同的共识规则在交易或块的验证中会以明确的和清晰的方式有所不同。这种差别可能以微妙的方式表现，比如共识规则适用于比特币脚本或加密原语（如数字签名）时。最后，共识规则的差别还可能会由于意料之外的方式，比如由于系统限制或实现细节所产生的隐含共识约束。在将Bitcoin Core 0.7升级到0.8之前的意料之中的硬分叉中看到了后者的一个例子，这是由于用于存储块的Berkley DB实现的限制引起的。

从概念上讲，我们可以将硬分叉子看成四个阶段：软分叉，网络分叉，挖矿分叉和区块链分叉。该过程开始于开发人员创建的客户端，这个客户端对共识规则进行了修改。当这种新版本的客户端部署在网络中时，一定百分比的矿工，钱包用户和中间节点可以采用并运行该版本客户端。得到的分叉将取决于新的共识规则是否适用于区块，交易或系统其他方面。如果新的共识规则与交易有关，那么当交易被挖掘成一个块时，根据新规则创建交易的钱包可能

会产生出一个网络分叉，这就是一个硬分叉。如果新规则与区块有关，那么当一个块根据新规则被挖掘时，硬分叉进程将开始。

首先，是网络分叉。基于旧的共识规则的节点将拒绝根据新规则创建的任何交易和块。此外，遵循旧的共识规则的节点将暂时禁止和断开发送这些无效交易和块的任何节点。因此，网络将分为两部分：旧节点将只保留连接到旧节点，新节点只能连接到新节点。基于新规则的单个交易或块将通过网络波动，实现分区，导致出现两个网络。

一旦使用新规则的矿工开采了一个块，挖矿和区块链也将分叉。新的矿工将在新区块之上挖掘，而老矿工将根据旧的规则挖掘一个单独的链条。处于分区的网络将使得按照各自共识规则运行的矿工将不会接收彼此的块，因为它们连接到两个单独的网络。

10.13.3分离矿工和难度

随着矿工们被分裂开始开采两条不同的链条，链上的哈希算力也被分裂。两个链之间的挖矿能力可以分成任意比例。新的规则可能只有少数人跟随，或者也可能是绝大多数矿工。

我们假设，例如，80%-20%的分割，大多数挖矿能力使用新的共识规则。我们还假设分叉在改变目标（retarget）后立即出现。这两条链将从改变目标之后各自接受自己的难度。新的共识规则拥有80%的以前可用的挖矿权的承诺。从这个链的角度来看，与上一周期相比，挖矿能力突然下降了20%。区块将会平均每12分钟发现一次，这意味着可以扩大这条链的挖矿能力下降了20%。这个块发行速度将持续下去（除非有任何改变哈希功率的因素出现），直到2016块开采，这将需要大约24,192分钟（每个块需要12分钟）或16.8天。 16.8天后，基于此链中哈希算力的减少，改变目标将再次发生，并将难度调整（减少20%）每10分钟产生一个区块。

少数人认可的那条链，根据旧规则继续挖矿，现在只有20%的哈希算力，将面临更加艰巨的任务。在这条链上，平均每隔50分钟开采一次。这个难度将不会在2016个块之前进行调整，这将需要100,800分钟，或大约10周的时间。假设每个块具有固定容量，这也将导致交易容量减少5倍，因为每小时可用于记录交易的块大幅减少了。

10.13.4有争议的硬叉

这是共识软件开发的黎明。正如开源开发改变了软件的方法和产品，创造了新的方法论，新工具和新社区，共识软件开发也代表了计算机科学的新前沿。在比特币发展路线图的辩论，实验和纠结之中，我们将看到新的开发工具，实践，方法和社区的出现。硬分叉被视为有风险，因为它们迫使少数人被迫选择升级或是必须保留在少数派链条上。将整个系统分为两个竞争系统的风险被许多人认为是不可接受的风险。结果，许多开发人员不愿使用硬分叉机制来实现对共识规则的升级，除非整个网络都能达成一致。

任何没有被所有人支持的硬分叉建议也被认为是“有争议”的，不愿冒险分裂系统。硬分叉的问题在比特币开发社区也是非常有争议的，尤其是与控制区块大小限制的共识规则的任何提议相关。一些开发商反对任何形式的硬叉，认为它太冒险了。另一些人认为硬分叉机制是提升共识规则的重要工具，避免了“技术债务”，并与过去提供了一个干净的了断。

最后，有些开发商看到硬分叉作为一种应该很少使用的机制，应该经过认真的计划，并且只有在近乎一致的共识下才建议使用。我们已经看到出现了新的方法来解决硬分叉的危险。在下一节中，我们将看一下软分叉，以及BIP-34和BIP-9信号和激活共识修改的方法。

10.13.5软分叉

并非所有共识规则的变化都会导致硬分叉。只有前向不兼容的共识规则的变化才会导致分叉。如果共识规则的改变也能够让未修改的客户端仍然按照先前的规则对待交易或者区块，那么就可以在不进行分叉的情况下实现共识修改。这就是软分叉，来区分之前的硬分叉。实际上软分叉不是分叉。软分叉是与共识规则的前向兼容并作些变化，允许未升级的客户端程序继续与新规则同时工作。

软分叉的一个不是很明显的方面就是，软分叉级只能用于增加共识规则约束，而不是扩展它们。为了向前兼容，根据新规则创建的交易和块也必须在旧规则下有效，但是反之亦然。新规则只能增加限制，否则，根据旧规则创建的交易和区块被拒绝时，还是会将触发硬分叉。

软叉可以通过多种方式实现，该术语不定义单一方法，而是一组方法，它们都有一个共同点：它们不要求所有节点升级或强制非升级节点必须脱离共识。

10.13.5.1软分叉重新定义NOP操作码

基于NOP操作码的重新解释，在比特币中实现了一些软分叉。 Bitcoin脚本有10个操作码保留供将来使用，NOP1到NOP10。 根据共识规则，这些操作码在脚本中的存在被解释为无效的运算符，这意味着它们没有任何效果。 在NOP操作码之后继续执行，就好像它不存在一样。因此，软叉可以修改NOP代码的语义给它新的含义。

例如，BIP-65（CHECKLOCKTIMEVERIFY）重新解释了NOP2操作码。 实施BIP-65的客户将NOP2解释为OP_CHECKLOCKTIMEVERIFY，并在其锁定脚本中包含该操作码的UTXO上，强制了绝对锁定实践的共识规则。 这种变化是一个软分叉，因为在BIP-65下有效的交易在任何没有实现（不了解）BIP-65的客户端上也是有效的。 对于旧的客户端，该脚本包含一个NOP代码，这被忽略。

10.13.5.2其他方式软分叉升级

NOP操作码的重新解释既是计划的，也是共识升级的明显机制。然而，最近，引入了另一种软分叉机制，其不依赖于NOP操作码，用于非常特定类型的共识改变。这在隔离见证掌机[segwit]中有更详细的检查。 Segwit是一个交易结构的体系结构变化，它将解锁脚本（见证）从交易内部移动到外部数据结构（将其隔离）。

Segwit最初被设想为硬分叉升级，因为它修改了一个基本的结构（交易）。在2015年11月，一位从事比特币核心工作的开发人员提出了一种机制，通过这种机制，可以将软件包引入segwit。用于此的机制是在segwit规则下创建的UTXO的锁定脚本的修改，使得未修改的客户端将任何解锁脚本视为可锁定脚本。

因此，可以引入segwit就是软分叉，而不需要每个节点必须从链上升级或拆分网络。有可能还有其他尚未被发现的机制，通过这种机制可以以向前兼容的方式进行升级，都作为软分叉。

10.13.6对软分叉的批评

基于NOP操作码的软分叉是相对无争议的。 NOP操作码被放置在比特币脚本中，明确的目标是允许无中断升级。然而，许多开发人员担心软分叉升级的其他方法会产生不可接受的折衷。对软分叉更改的常见批评包括：技术性债务由于软叉在技术上比硬叉升级更复杂，所以引入了技术性债务，这是指由于过去的设计权衡而增加代码维护的未来成本。代码复杂性又增加了错误和安全漏洞的可能性。验证放松未经修改的客户端将交易视为有效，而不评估修改的共识规则。

实际上，未经修改的客户端不会使用全面的协商一致的规则来验证，因为它们对新规则无视。这适用于基于NOP的升级，以及其他软分叉升级。

不可逆转升级因为软分叉产生额外的共识约束的交易，所以它们在实践中成为不可逆转的升级。如果软分叉升级在被激活后被回退，根据新规则创建的任何交易都可能导致旧规则下的资金损失。例如，如果根据旧规则对CLTV交易进行评估，则不存在任何时间锁定约束，并且可以花费在任何时间。因此，评论家认为，由于错误而不得不被回退的失败的软分叉几乎肯定会导致资金的流失。

10.14使用区块版本发出软分叉信号

由于软分叉允许未经修改的客户在协商一致的情况下继续运作，“激活”软分叉的机制是通过向矿工发出信号准备：大多数矿工必须同意他们准备并愿意执行新的共识规则。 为了协调他们的行动，有一个信号机制，使他们能够表达他们对共识规则改变的支持。 该机制是在2013年3月激活BIP-34并在2016年7月被BIP-9激活所取代。

10.14.1 BIP-34信号和激活

在BIP-34中的第一个实施使用块版本字段来允许矿工表示准备达成特定的共识规则更改。在BIP-34之前，按照惯例将块版本设定为“1”，而不是以共识方式执行。BIP-34定义了一个共识规则变更，要求将Coinbase交易的coinbade字段（输入）包含块高度。在BIP-34之前，Coinbase可以让矿工选择包含的任意数据。在BIP-34激活之后，有效块必须在Coinbase的开始处包含特定的块高度，并且使用大于或等于“2”的版本号进行标识。

为了标记BIP-34的更改和激活，矿工们将块版本设置为“2”而不是“1”。这没有立即使版本“1”块无效。一旦激活，版本“1”块将变得无效，并且将需要所有版本“2”块以包含Coinbase库中的块高度才能有效。

BIP-34基于1000个块的滚动窗口定义了两步启动机制。矿工将以“2”作为版本号来构建块，从而向BIP-34发出信号。严格来说，由于共识规则尚未被激活，这些区块还没有遵守新的共识规则，即将包含块高度在内的基础交易中纳入统一规则。

共识规则分为两个步骤激活：如果75%（最近1000个块中的750个）标有版本“2”，则版本“2”块必须包含coinbase交易中的块高度，否则被拒绝为无效。 版本“1”块仍然被网络接受，不需要包含块高度。 这个新时期的新旧共识规则共存。当95%（最近1000块中的950）是版本“2”时，版本“1”块不再被视为有效。 版本“2”块只有当它们包含coinbase中的块高度（根据先前阈值）时才有效。 此后，所有块必须符合新的一致性规则，所有有效块必须包含coinbase交易中的块高度。根据BIP-34规则成功发信号和激活后，该机制再次使用两次以激活软分叉：BIP-66标签的严格DER编码通过BIP-34信号通过块版本“3”激活，无效版本“2”块。BIP-65 CHECKLOCKTIMEVERIFY被块版本“4”的BIP-34信号激活，无效版本“3”块。BIP-65激活后，BIP-34的信号和激活机制退出，并用下面描述的BIP-9信号传导机制代替。这个标准在BIP-34 (Block v2, Height in Coinbase)中定义。

10.14.2 BIP-9信号和激活

BIP-34，BIP-66和BIP-65使用的机制成功地激活了三个软分叉。 然而，它又被替换了，因为它有几个限制：通过使用块版本的整数值，一次只能激活一个软分叉，因此需要软分叉提议之间的协调以及对其优先级和排序的协议。此外，由于块版本增加，所以机制并没有提供一种直接的方式来拒绝变更，而是提出一个不同的方法。 如果老客户仍然在运行，他们可能会错误地将信号转换为新的更改，作为以前拒绝的更改的信号。每个新的更改不可逆转地减少可用的供将来更改的块版本。提出BIP-9来克服这些挑战，提高实施未来变化的速度和便利性。BIP-9将块版本解释为bit字段而不是1个整数。因为块版本最初用作整数，因此版本1到4，只有29位可用作位字段。这留下29位，可以独立使用，同时在29个不同的提案上表示准备就绪。BIP-9还设置了信令和激活的最大时间。矿工们不需要永远发出信号。如果提案在TIMEOUT期间（在提案中定义）未激活，则该提案被视为被拒绝。该提议可以使用不同位的信令重新提交，更新激活周期。

此外，在TIMEOUT已经过去并且特征被激活或被拒绝之后，信令位可以被再次用于另一个特征而不会混淆。因此，多达29次更改可以并行发出信号，TIMEOUT后可将这些位“再循环”以提出新的更改。

注意虽然信令位可以重复使用或回收利用，但只要投票期间不重叠，BIP-9的作者就建议仅在必要时重复使用位; 主要是由于旧软件中的bug，可能会发生意外的行为。 总之，我们不应该期望在所有29位都被使用一次之前看到重用。

建议的更改由包含以下字段的数据结构标识：名称用于区分提案的简短描述。 通常，BIP将该提案描述为“bipN”，其中N是BIP编号。位0到28，矿工使用的块版本中的位用于表示此提案的批准。开始时间信号开始的时间（基于中值时间过去或MTP），之后该位的值被解释为提示的信令准备。时间结束该时间（基于MTP），如果尚未达到激活阈值，则认为该更改被拒绝。

与BIP-34不同，BIP-9根据2016块的难度改变目标（retarget）周期，在整个间隔中计数激活信号。 对于每个改变目标期间，如果提案的信号块的总和超过95%（2016中的1916），则该提案将在稍后的改变目标期间激活。BIP-9提供了一个提案状态图，以说明一个提案的各个阶段和转换，如图10-10所示。

一旦它们的参数在比特币软件中被知道（定义），提案将在DEFINED状态下开始。 对于具有MTP的块在开始时间之后，提议状态转换到STARTED。 如果在改变目标期间超过了投票阈值，并且未超过超时，则提案状态转换为LOCKED_IN。 一个改变目标期后，该提案变为活动。

一旦达到这个状态，提案仍然处于活动状态。 如果在达到投票阈值之前超时时间，提案状态将更改为“已故”，表示已拒绝的提案。 REJECTED的建议永远在这个状态。BIP-9首次实施用于激活CHECKSEQUENCEVERIFY和相关BIP（68,112,113）。 名为“csv”的建议在2016年7月成功启动。标准定义在BIP-9 (Version bits with timeout and delay).

10.15共识软件开发

共识软件开发不断发展，对于改变共识规则的各种机制进行了很多讨论。由于其本质，比特币在协调和变化共识方面树立了非常高的标杆。作为一个去中心化的制度，不存在将权力强加于网络参与者的“权威”。权力分散在多个支持者，如矿工，核心开发商，钱包开发商，交易所，商家和最终用户之间。这些支持者不能单方面做出决定。例如，矿工在理论上可以通过简单多数（51％）来改变规则，但受到其他支持者的同意的限制。

如果他们单方面行事，其他参与者可能会拒绝遵守，将经济活动保持在少数链条上。没有经济活动（交易，商人，钱包，交易所），矿工们将用空的块开采一个无价值的货币。这种权力的扩散意味着所有参与者必须协调，或者不能做出任何改变。现状是这个制度的稳定状态，只有在很大程度上达成一致的情况下，才能有几个变化。软分叉的95％门槛反映了这一现实。

重要的是要认识到，对于共识发展没有完美的解决办法。硬分叉和软分叉都涉及权衡。对于某些类型的更改，软分叉可能是一个更好的选择;对于别人来说，硬分叉可能是一个更好的选择。没有完美的选择，都带有风险。共识软件开发的一个不变特征是变革是困难的，是共识力量的妥协。有些人认为这是共识制度的弱点。在时间上，你可以像我一样看到它，把它当作这个系统最大的优势。

保全比特币是很有挑战性的事，因为比特币不像银行账户余额那样体现抽象价值。比特币其实更像数字现金或黄金。 你可能听过这样的说法，“谁持有几乎等同法律拥有。”好吧，在比特币的世界里，谁持有谁拥有。持有拥有解锁比特币的密钥就相当于持有现金或一块贵重金属。你可能会将密钥丢失，会放错地方，会被盗或者不小心错支了数额。无论是哪种场景，用户都没有办法撤回，因为这就像是将现金丢在了车水马龙的大街上。

不过，与现金、黄金或者银行账户相比，比特币有着一个独一无二的优势。你不能“备份”你的现金、黄金或者银行账 户，但你可以像备份其他文件一样，备份含有密钥的比特币钱包。它可以被复制成很多份，放到不同的地方保存起来， 甚至能打印到纸上进行实体备份。比特币与至今为止的其他货币是如此不同，以致于我们需要以一种全新的思维方式来 衡量比特币的安全性。V