Table of Contents

Introduction	1.1
介绍	1.2
区块链	1.3
交易	1.4
合约	1.5
钱包	1.6
支付处理	1.7
运行模式	1.8
P2P网络	1.9
挖矿	1.10

比特币开发者指南(中文版)

本指南是基于比特币官方网站的Bitcoin Developer Guide（比特币开发者指南）翻译而来，已同步开源到github【查看github版本】和gitbook【查看gitbook版本】。

目前本指南正在翻译当中，尚未完成，有兴趣一起翻译或对区块链技术有兴趣的同学可以通过搜索加qq群：574770647，或点击加群链接参与讨论。

本文档翻译尚处在初期阶段，欢迎大家积极参与翻译或校对，任何问题都可以在github issues留言或加群讨论，也可联系我的github上展示的邮箱。

后续计划与官网版本同步更新维护。

文档翻译完成后续的目标暂定分模块协作阅读比特币源码，希望有志同道合的朋友可以一起参与讨论、分享。

vacing, 2017年8月

当前翻译进度

（**[*]**表示进行中，**[√]**表示已完成，**[×]**表示未开始）：

- [√]介绍
- [√]区块链
- [√]交易
- [√]合约
- [*]钱包
- [×]支付处理
- [×]运行模式
- [√]P2P网络
- [×]挖矿

重要节点记录

- 2017年8月30日，P2P网络（P2P Network）部分翻译完成
- 2017年8月24日，创建github和gitbook项目，并将之前翻译的部分提交。

License

对应英文文档

本开发者只能旨在提供理解比特币和构建比特币应用的基础知识，但是并不是一个硬性标准。为了更好地使用它，你需要安装最新的bitcoin内核的客户端，可以下载github里面的源码或者官网的预编译版本 。

关于比特币开发的问题最好发送到比特币论坛和IRC 频道。 对于Bitcoin.org的文档的错误和建议，请提交Github提案或者发送到 bitcoin-documentation 邮件列表。

在以下的内容里, 某些字符串被缩写成: “[…]” ，表示额外的数据被删除了, 以 “\” 结束的行表明续行。如果你把你的鼠标移动到一个段落，有链接的文字会变成蓝色。如果你移动到有链接的文字上，在工具提示上会出现一个简介。

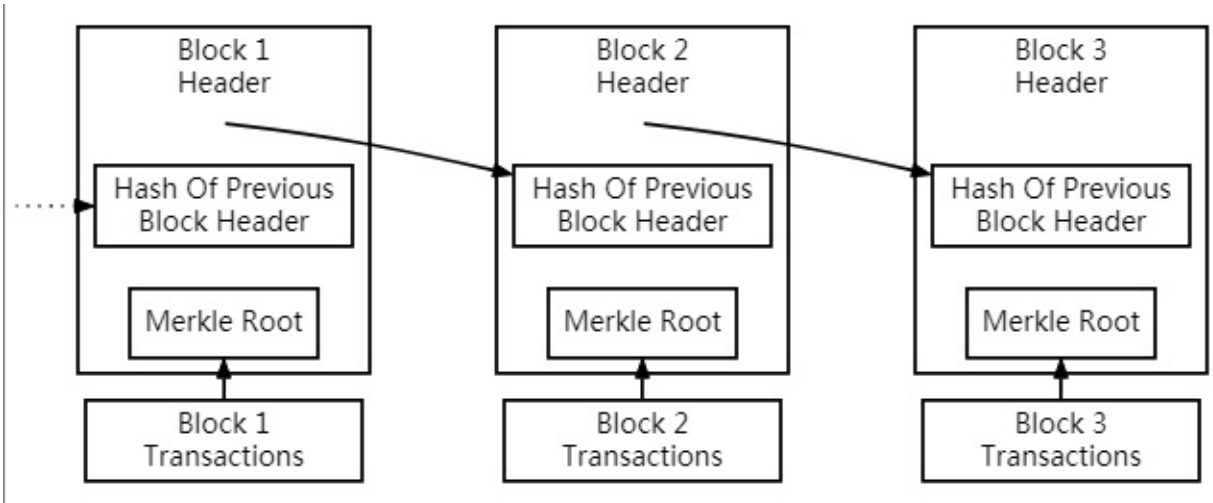
对应英文文档

区块链

区块链为比特比提供了一个公共账本，按照顺序和时间戳记录了交易。该系统可以防止双花交易和更改历史交易。

比特币网络中的每个全节点独立存储的区块链中，只包含该节点已经验证过的区块。当几个节点在它们的区块链中都包含了同一个区块，此时认为它们达成一致。节点之间为了保证一致性所遵循的验证规则别称为一致性规则。本节将讨论比特币核心使用多种一致性规则。

区块链概览

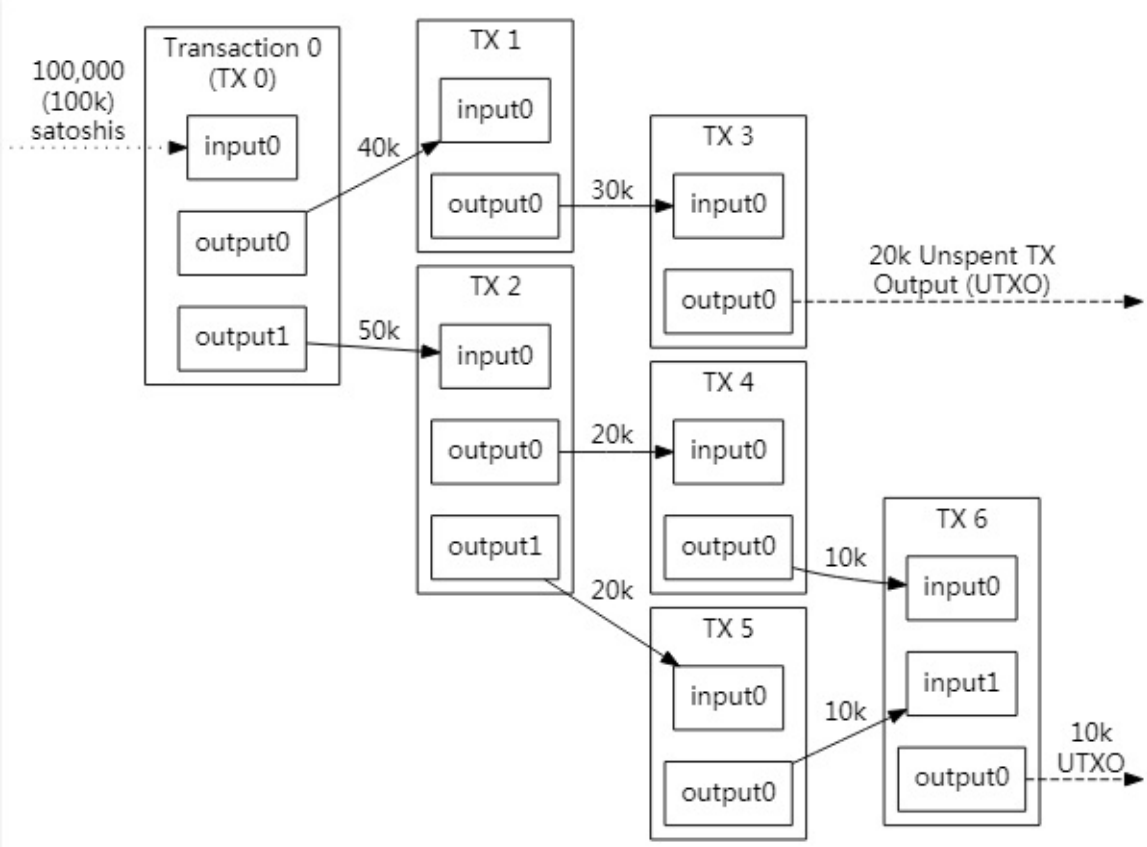


Simplified Bitcoin Block Chain

上图示意了一个简单版本的区块链。区块中包含的多个交易放置在区块的数据区，交易经过hash后组成pair，再进行hash，再组成pair，再进行hash，知道只剩下最后一个hash值，该值称为Merkle树的merkle根。

Merkle根保存在区块的头部中，每个区块头部还包含前一个区块的hash，这样就将区块链接起来（前向链表）。这样可以保证在不修改后续节点的情况下无法修改当前节点的内容。

交易也是被链接在一起。比特币钱包给人的印象是比特币从一个钱包到另一个钱包，但是实际上比特币是从一个交易到另一个交易。每个交易花费之前接收到的一个或多个交易的币，因此一个交易的输入是之前另一个交易的输出。



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

在将交易结果分送多个地址的情形下，一个交易可以产生多个输出，但是一个指定的交易输出只能在整条链中作为输入一次。后续任何的引用都会由于禁止双花导致失败。

输出和交易标志符(Transaction Identifiers，TXIDs)绑定，它是交易的签名。

由于每个输出只能被花费一次，因此区块链中所有的输出可以分为两类：未花费的输出(Unspent Transaction Outputs, UTXOS) 和已花费的输出。一个有效的交易，必须使用UTXOs作为输入。

除了基础币交易外，如果交易的输出值大于交易的输入值，这个交易会被拒绝；但是，如果交易的输入值大于交易的输出值，二者的差值则作为交易费用被挖出包含该交易区块的矿工占有。距离来说，上图显示的每个交易的输出都比输入少10000聪，缺少的部分就是作为交易费用的部分。

工作量证明POW（Proof Of Work）

区块链灾在网络上被匿名节点协作保存，因此比特币要求每个区块都投入一定工作量才能生成，以此来保证想要更新历史区块的不可靠节点需要比诚实节点付出更多的努力才能更新历史区块（不可靠节点更新历史区块的同时，诚实节点也在新增区块，二者是追赶的关系）。

链在一起的区块可以保证，如果不更新指定区块的所有后续区块，则无法更新该区块中的交易。因此，更新一个特定区块的代价随着新区块的增加而增加，同时放大了工作量证明的作用。

工作量证明机制利用了密码学hash结果明显的随机特性。一个好的hash算法将任意的数据转换成看起来随机的数字。如果输入数据的发生任何的变动，hash过程重新执行，一个新的随机数就会产生，因此无法通过更新输入数据有目的的预测hash结果。

为了证明节点为了生成区块做了一定的工作，节点必须产生出一个hash结果不超出指定值的区块头部。比如，如果最大的可能的hash值的个数为2^256-1，可以证明节点平均尝试2次就可以产生一个hash值小于2^255的头部。

在上面的例子中，节点几乎每个一次就可以成功一次。同时可以根据指定的目标门限，估计一次hash尝试成功的概率。比特币假定门限和成功概率是线性关系，门限越低，平均需要的尝试次数越多。

只有当区块的hash值满足一致性协议指定的难度值时，该区块才会被加入到区块链中。每隔2016个区块，网络利用存储在每个区块头中的时间戳计算产生这2016各区块的时间间隔。该间隔的理想值为120960S(两周)。

- 如果实际的间隔小于2周，则期望的难度值会按比例上升（最高300%），由此下一批2016个区块按相同速率产生，则刚好花费2周的时间。
- 如果实际的间隔大于2周，期望的难度值则会按比例下降（最低到75%）。

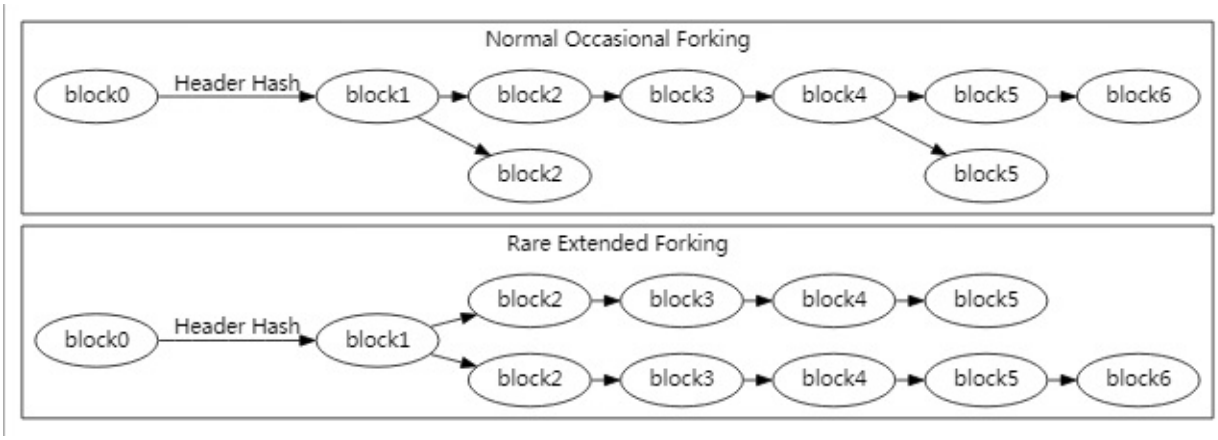
（注意：比特比核心实现时的一个1偏移错误导致需要使用2015个区块的时间戳代表2016个区块，引入一定偏差）

由于每个hash头都要满足指定的难度值，而且每个区块都会链接它前面的区块，因此更新一个区块平均来讲需要付出从该区块创造到当前时刻区块链整体算力的总和。因此只有你获得了了网络的大部分算力，才能够可靠的进行51%攻击修改交易历史（但是，需要直出的是，即使少于50%d 算力，仍然有很大可能性进行这种攻击）。

区块头部中提供了几个容易更新的字段，比如专门的nonce字段，因此获取新的hash值并不一定要等待新的交易。同时，只需要对80字节的区块头进行hash，因此在区块中包含大量的交易不会降低hash的效率，增加新的交易只需要重算Mercle树。

块高度和分叉

所有成功挖到新块的矿工都可以把他们的新块添加到区块链中（假定这些区块都是有效的）。这些区块通过它们的区块高度进行定位（区块高度是指当前区块到创世区块的之间的区块个数）。例如，2016是第一个进行难度调整的区块。



由于多个矿工可能几乎同时挖到新区块，因此可能存在多个区块拥有相同区块高度。这种情况下就在区块链中产生了明显的分叉，如上图所示。

当几个矿工同时生产出区块，每个节点独立的判断选择接受哪个，在没有其他考虑的情况下，节点通常选择接受他们看到的第一个区块。最终，一个矿工生产出来了一个区块，它附在了几条并行区块分叉中的一条，这时这条区块就比其他区块更有优势。假设一个分叉只包含有效的区块，正常的节点通常会跟随难度最大的区块继续工作，抛弃其他分叉上的孤儿区块。

如果不同的矿工为了不同的目的工作，长期的分叉也是可能。比如一些矿工在勤奋地延长区块，另一些却在进行51%攻击来修改交易历史。

由于可能存在多个分叉，因此区块高度不能作为区块的唯一标识，通常使用头部的hash值（通常进行字节顺序反转，并用16进制表示）。

交易数据

每个区块中必须包含一笔到多笔交易。这些交易中的第一笔都是币基础交易，或被称为生成交易，负责搜集和支付区块奖励（包括块补贴和包含在该块中的交易的手续费）。

生成交易的比特比作为未花费输出需要有特殊的条件：在100各区块以后才可以。这在一定程度上可以阻止矿工花费了该输出但是后续由于该区块所在分叉失效，导致该生成交易无效的情况。

区块当中并不强制要求一定有非生成交易，但是矿工为了获取交易手续费通常会包含额外的交易。

包含生成交易在内的所有交易，都被编码为二进制原始交易格式包含在区块中。

二进制原始交易格式通过hash产生一个交易标志TXIS，Merkle树算法把这些交易组成一对对，然后把他们hash在一起，如果这里有奇数各txid，没有txid的交易将会和自己的复制镜像配对；hash的结果之间继续进行配对hash，单独的结果还是和自己配对，这样依次递归，知道剩下唯一的hash结果，Merkle根。

一个包含5个交易的Merkle树生成过程如下：

```
        ABCDEEEE .......Merkle root
       /        \
    ABCD         EEEE
   /    \       /
  AB    CD    EE .......E is paired with itself
 /  \  /  \  /
A  B  C  D  E .........Transactions
```

正如在简化支付验证一节讲到的，Merkle树允许客户端通过向相邻完全节点请求区块头部的merkle根和一系列的中间hash结果来自己验证交易是否包含在指定区块中。相邻完全节点不一定必须是可信任节点，因为伪造节点头部和中间的哈希值代价巨大。

举例来说，如上图所示，为了验证交易D在区块中，SPV端只需要交易C的内容，以及hash值AB、EEEE和Merkle根，此外再不需要知道其他交易的任何内容。如果区块中的这5个交易都达到限定的最大值，那么下载该区块需要超过5000,000字节，而下载3各hash值和一个头部仅需要140字节。

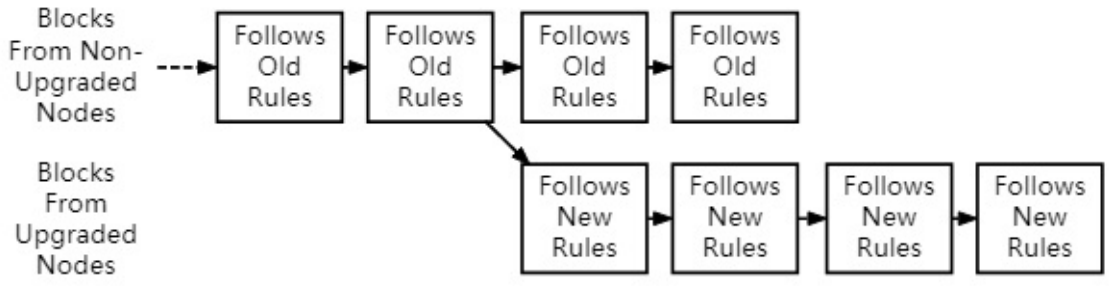
> 注意：如果在区块中发现两个相同的交易ID，存在一种情况是Merkle树可能可能会和一个去除了了全部重复的区块发生碰撞，这是由于merkle树对不平衡（奇数）叶节点的复制处理。因此，把交易ID不同的交易放在同个区块中是实际可行的方案，这并不会增加诚实节点的负担，但是需要在节点缓存时进行检查。否则一个去除重复的有效节点可能和另一个节点有相同的merkle树和块hash值，但是因缓存的无效交易被拒绝，导致安全问题。

一致性规则的改变

为了保持一致性，所有的劝解点使用相同的一致性规则确认区块的有效性。但是，有时在引入新特性或防止网络滥用时会导致一致性规则的变化。当新的规则实施时，存在一个遵守新旧规则的节点同事存在的时期，此时有两种打破一致性的可能：

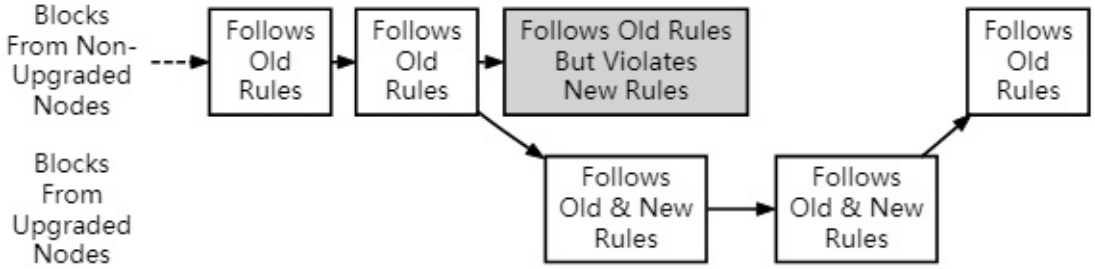
1. 一个符合新规则的区块被新的节点接受，但是不能被老的节点接受。比如，区块中使用了新的交易特性，升级的节点理解该特性，并接受它，但是老的节点按照旧规则判断一致性失败拒绝该区块。
2. 一个违反新规则的区块被新节点拒绝，但是会被老的节点接受。比如，一个滥用交易的特性在就区块中，该区块被新节点拒绝，但是被老节点接受。

在第一种情况下，被未升级节点拒绝，从旧节点接受数据的挖矿软件拒绝和从新节点接收数据的挖矿软件工作在相同的区块链。这会产生永久的分歧链，一个针对未升级的节点，一个针对已升级的节点，这被称为硬分叉



A Hard Fork: Non-Upgraded Nodes Reject The New Rules, Diverging The Chain

在第二种情况下，新节点拒绝旧区块的情况，如果新节点可以公职更多的算力是可以防止区块链发生硬分叉的。在这种情况下，未升级节点将会和新节点一样认为新节点的区块有效，因此新节点将会产生更强算力的链，此时旧节点也会接受最有效的长链。这被称为软分叉。



A Soft Fork: Blocks Violating New Rules Are Made Stale By The Upgraded Mining Majority

- 增强限制：旧客户端可以为新规则服务，新客户端只为新规则服务，因此新规则的服务能力大于旧规则服务能力，最终切- 换到新规则，导致暂时的分叉，称为软分叉。
- 减弱限制：旧客户端只为旧规则服务，新客户端可以为新规则和旧规则服务，因此旧规则的能力大于新规则的能力，形成永久的分叉，称为硬分叉。

尽管分叉表示对区块链实际的分裂，但是对一致性规则的改变通常还是会用潜在产生硬分叉或软分叉进行描述。比如“增加区块大小超过1MB需要一次硬分叉”，该例中，并不是真正需要一次区块链的分叉，只是可能而已。

一致性规则的改变可能通过多种方式引起。在比特比出现的前两年，中本聪通过释放后向兼容的版本改变强制立即使用新规则，进行过多次的软分叉。许多像BIP30这样的软分叉是通过在代码中预先编码指定的一个固定时间或区块高度实现的。这种通过指定固定日期进行分叉的方式称为用户发起软分叉（User Activated Soft Forks， UASF），它依靠大量的用户节点强制在指定日期后使新规则生效。

后续的软分叉会等待网络大部分算力（75%或95%）认可使用新的一致性规则。一旦超过认可门限，所有的节点都会使用新的规则。这种依赖矿工进行分叉的方式称为矿工发起软分叉（Miner Activated Soft Forks, MASF）。

检查分叉

未升级节点可能在两种分叉期间使用和发布错误的信息，存在多种情况可能导致经济损失。

- 未升级节点可能依赖和接受已经被新节点认为无效的区块，这些区块永远也不可能未来被接受区块链的一部分。
- 未升级节点也可能拒绝一些会被新节点添加的区块和交易，导致产生不完整的信息。

比特比核心包含检查区块分支的代码，通过查看区块链的工作量证明尽心判断。如果一个未升级节点收到的区块链头部表明该头部已经领先其认可有效的区块链头部6个块，节点就会通过 `getnetworkinfo` RPC函数结果发出一个警告，如果设定 `-alternotify` 命令则还会调用该命令。这将警告操作者未升级节点不能升级到可能的最有效区块链。

全节点还能检查区块和交易的版本数字。如果最近收到的区块或交易的版本高于该节点使用的版本，它就会假定自己没有使用最新的一致性规则。比特比核心就会把这个情况通过 `getnetworkinfo` RPC函数结果发出一个警告，如果设定 `-alternotify` 命令则还会调用该命令。

在另外一种情况下，如果区块和交易来自一个明显不使用最新协议的节点，那么其中的数据是不可依赖的。

SPV客户端通过连接到多个全节点可以探测可能的硬分叉，并且在排除掉传输延迟和无效区块后，可以判断他们是否在同一条链的相同高度。如果发生不一致，客户端将会从区块链较弱的节点断开。

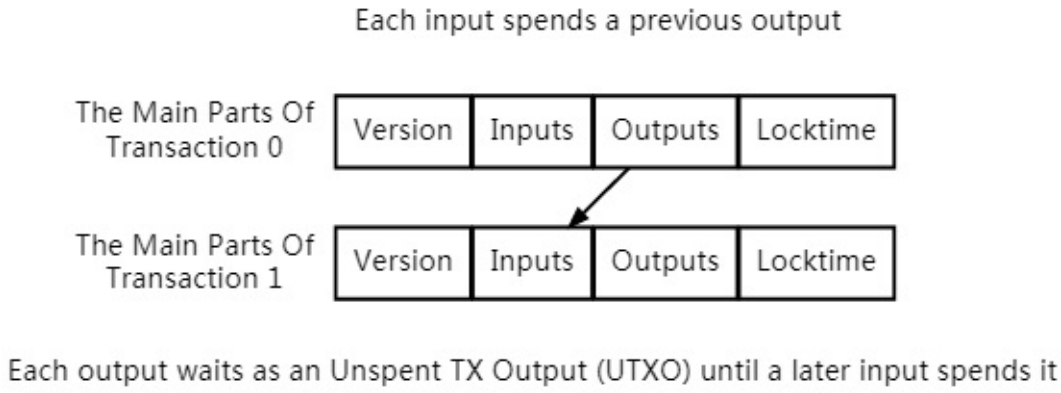
SPV客户端还需要监控区块和交易的版本号变化，保证他们接受交易和创建新交易都是使用最新的一致性规则。

对应英文文档

交易

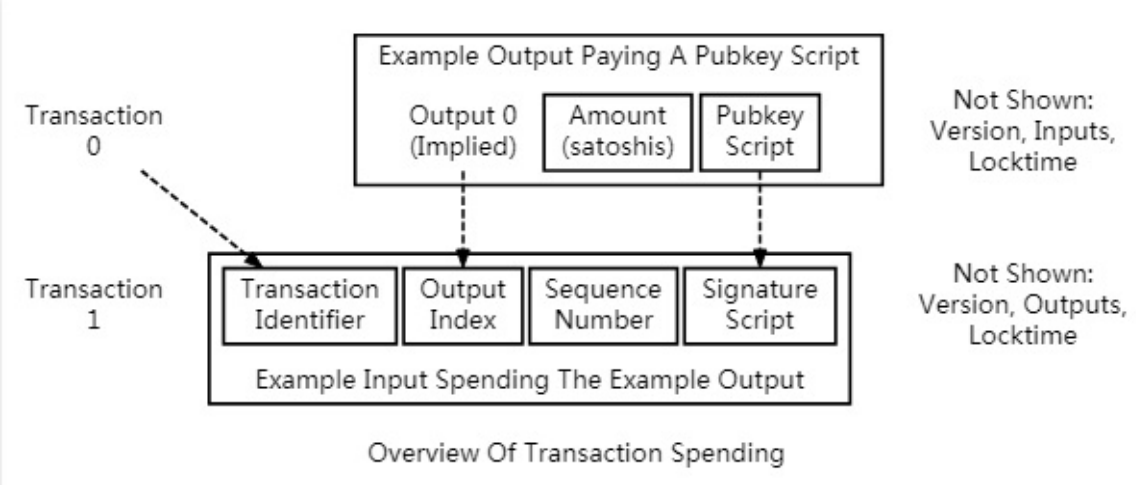
交易使用户可以消费他们的比特币。每个交易有多个部分组成，既包括简单直接的付款，也包括复杂交易。这一部分将会讨论交易的每个部分，并演示如果将它们组合成一个完整的交易。

简单起见，这部分将忽略生成交易。生成交易只能被比特币矿工创建，并且它们对下面的规则存在许多例外情况。这里建议读者在区块链一章了解生成交易的一些细节，本章将不在交易规则后面单独注明对生成交易的例外情况。



上图展示了比特币交易的主要部分。每笔交易至少有一个输入和一个输出，每个输入会花费上个输出产生的比特币，每个输出都作为UTXO直到被作为输入花费掉。当你的比特币钱包告诉你你还有10000聪的比特币，它实际上说的是你拥有的一个或多个UTXO输出中包含10000聪。

每个交易的前缀为4字节的交易版本号，告诉比特币终端和矿工，采取什么样的策略验证它。这种设计可以让开发者在不影响旧区块的前提下为未来交易创建新的一致性规则。

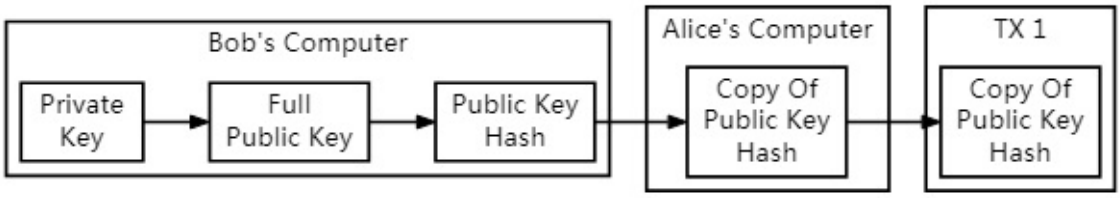


Overview Of Transaction Spending

一个输出有一个基于其在交易中位置的隐含索引号码，第一个输出的索引是0。同时，输出还有一定数量的比特币，这些比特币会支付给可以满足公钥脚本指定条件的人。任何可以满足该公钥脚本条件的人可以消费这些比特币。

一个输出使用交易标识和输出索引号（常被称为vout，代指输出向量output vector）来区分一个将要使用的输出。同时，它还有一个签名脚本，可以用来提供满足输出公钥脚本的条件的参数。（序列号和锁定时间是相关的，将在后续章节介绍）

下面的图片通过展示Alice给Bob一些币然后Bob把币花掉两个交易过程，反映出这些特性具体是如何被使用的。Alice和Bob都将使用最常见的P2PKH(Pay-To-Public-Key-Hash)的交易方式。P2PKH使Alice将比特币支付给一个地址，然后Bob再使用简单的密钥对继续将这些比特币花掉。



Creating A P2PKH Public Key Hash To Receive Payment

Bob必须首先创建一个公钥/私钥对，然后Alice才能创建第一各交易。比特币使用ECDSA签名算法结合secp256k1，secp256k1私钥是一个256bit的随机数。这个随机数被确定性地转换为一个secp256k1的公钥。由于这个转换过程是确定可重复的，因此不需要在本地保存公钥。

对公钥(pbkey)使用加密hash得到hash值，公钥hash过程也是可重复的，因此也没有必要保存该hash结果。Hash可以是公钥变短并模糊，方便手动誊写，并且提供一定的安全性保证防止未来不可预知的通过公钥重建私钥的问题。

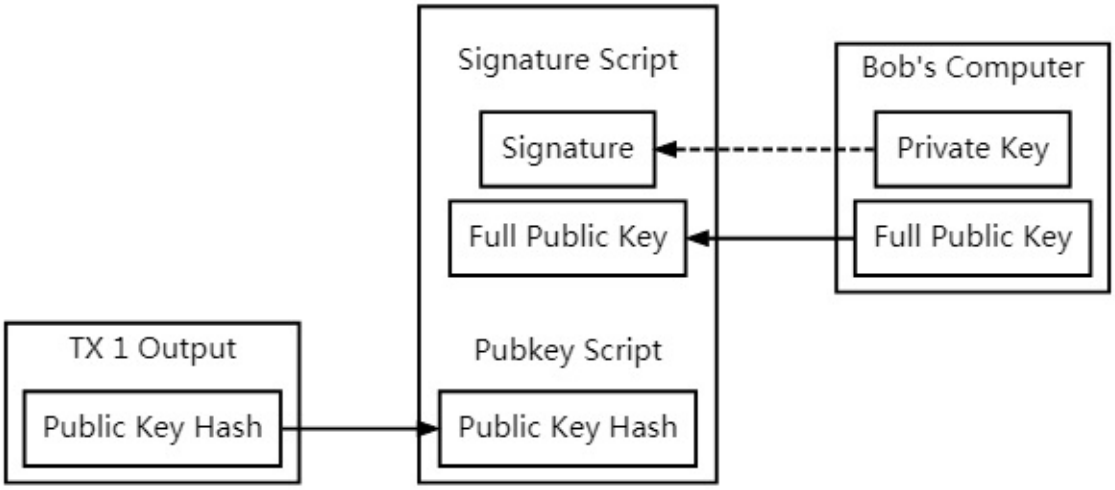
Bob把公钥hash提供给Alice。公钥hash通常像比特币地址一样进行编码后再进行传输，编码方式为使用base68对地址版本号、hash值、错误校验和编码为一个字符串。地址可以通过任何的媒介进行阐述，包括防止花费者联系接受者的单项没接，同时还可以进一步编码为另外的格式，比如包含 `bitcoin:` URI的二维码。

Alice收到该地址后会把它转换为标准hash后，就可以进行第一次交易了。她将创建一个包含指令的标准的P2PKH交易输出，允许任何可以证明自己拥有Bob提供的公钥对应私钥的人花费该笔输出。输出中的指令被称为公钥脚本。

Alice将交易广播出去后该交易会被添加到区块链。网络将他分类为UTXO，此时Bob的钱包软件将它展示为可花费的现金。

当后面Bob决定花费这份UTXO时，他必须创建一个指向Alice通过他提供的hash创建的交易的输入，称为交易标志（Transcation Identifier, txid），然后通过输出所以指定她使用的输出。Bob要创建一份签名，改签名包含满足Alice在签一份输出中保存的公钥脚本的条件。签名脚本又被称为ScriptSigs。

公钥脚本、签名脚本、secp256k1公钥、签名加上条件逻辑，可以创建一个可编程授权机制。

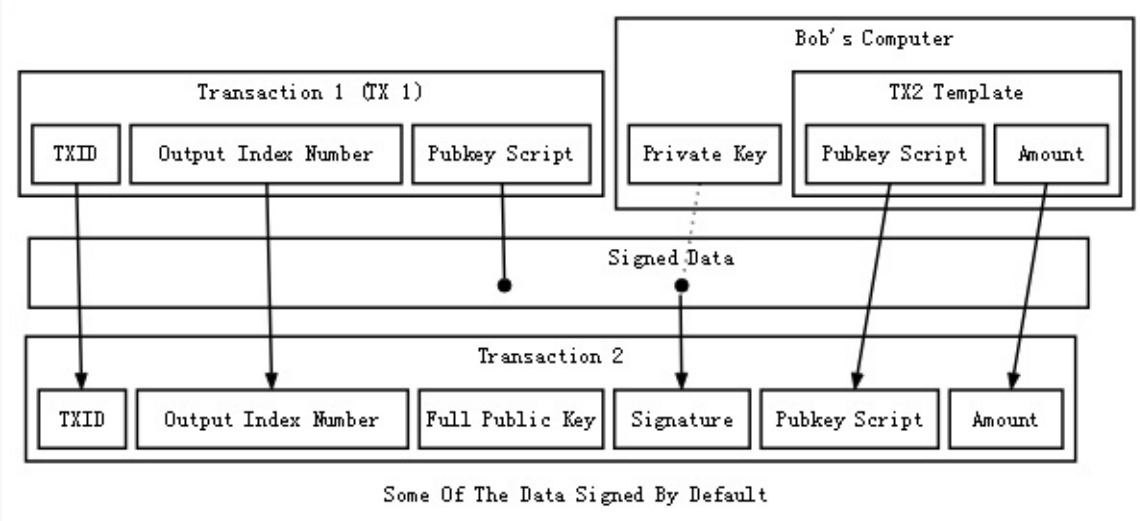


Spending A P2PKH Output

对于一个P2PKH类型的输出，Bob的签名脚本将要包含如下两条数据：

- 他的完整的公钥（非hash），因此公钥脚本可以检查它可以hash得到和Alice提供的值一样的结果。
- 一个通过ECDSA加密算法利用指定的交易数据（下面讲解）结合Bob的私钥计算得到的secp256k1签名。这使得公钥脚本可以校验Bob持有对应公钥的私钥。

Bob的secp256k1签名不仅仅证明Bob控制着自己的私钥，同时还可以防止他交易中的非签名脚本部分被攻击，以保证Bob可以在p2p网络中安全地广播自己的交易。



如上图所示，Bob签名的数据包括上次交易的txid和输出所以、上次交易的输出公钥脚本、Bob创建用于后续接受者花费这次交易输出的公钥脚本、本次转支付给下一个接收者的比特币数量。本质上来讲，除了包含公钥和secp256k1签名的签名脚本外整条交易的内容都被做了签名。

当把他的签名和公钥放到签名脚本后，Bob把这次交易通过p2p网络广播道所有的比特币矿工。每个节点和矿工在进一步广播或尝试把它添加到新区块前独立的校验交易信息。

P2PKH 脚本验证

验证过程需要执行签名脚本和公钥脚本。在一个P2PKH的输出，公钥脚本为：

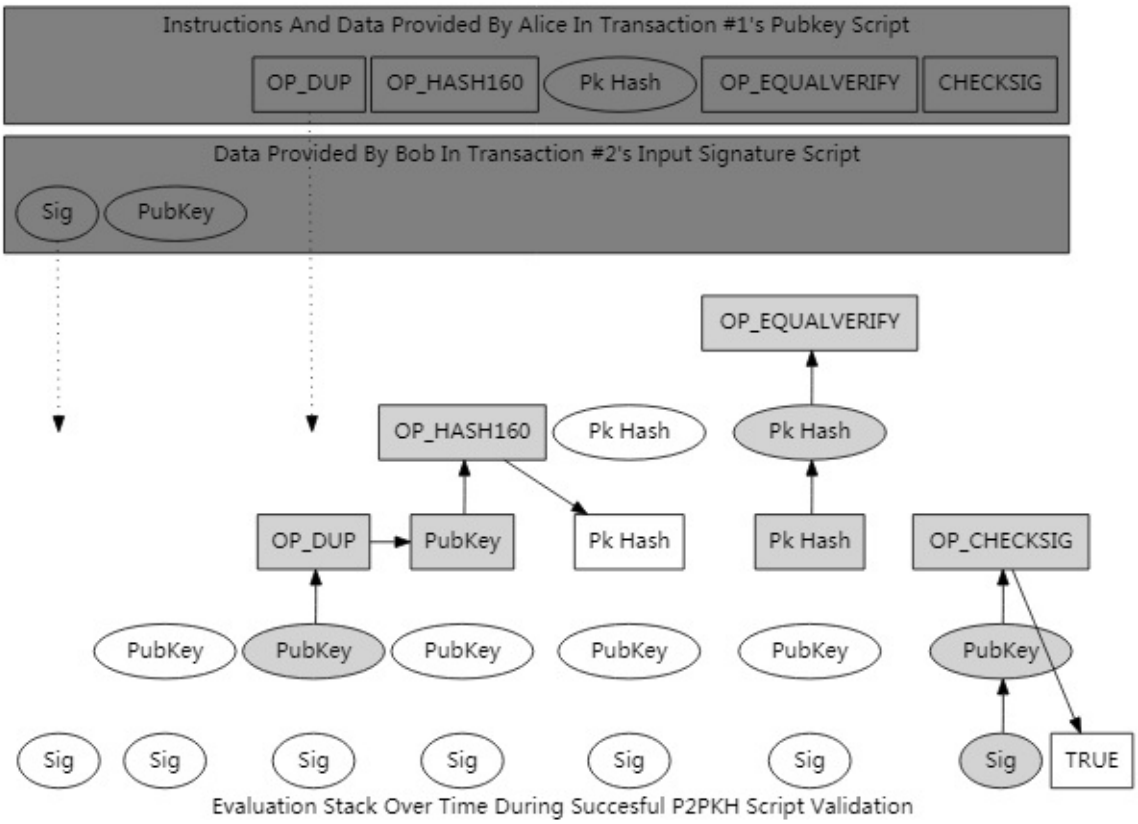
```
OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

花费者的签名脚本被执行后添加到脚本的前面。在一个P2PKH交易中，签名脚本包含secp256k1（sig）签名和完整的公钥（pubkey），创建如下的串行过程：

```
<Sig> <PubKey> OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

脚本语言是一种类Forth基于堆栈的语言，专门设计为无状态和非图灵完备的。无状态可以保证一旦一个交易被添加到区块链，没有条件能够表明它一直未被消费。非图灵完备（主要指没有循环和goto语句）可以使脚本不过于灵活便于预测，可以大大简化安全模型。

为了测试交易是否有效，签名脚本和公钥脚本中的操作一次执行一条，从Bob的签名脚本开始知道Alice的公钥脚本结束。下面的图片展示了标准P2PKH 公钥脚本的执行，图片下面是对执行过程的描述。



1. 签名（来自Bob的签名脚本）被添加到一个空栈中。由于它只是数据而已，所以不需要做其他的工作。公钥Key（也是来自签名脚本）添加在签名的上方。
2. 对于Alice的公钥脚本，首先执行最开始的 `OP_DUP` 操作。 `OP_DUP` 把当前堆栈顶部的数据复制一份再压入堆栈，这里指的是复制一份Bob提供的公钥。
3. 接下来执行 `OP_HASH160` 操作，得到堆栈顶部数据的Hash值并放在堆栈中。这里是创建的Bob的公钥的hash值。
4. 接下来Alice的公钥脚本将之前Bob在第一次交易中提供的公钥hash值入栈。到这里，堆栈顶部包含了2份Bob公钥的hash值。
5. 从这里开始变的有趣：Alice的公钥脚本执行 `OP_EQUALVERIFY` 操作。 `OP_EQUALVERIFY` 等价于顺序执行 `OP_EQUAL` 和 `OP_VERIFY` （未显示）。
   `OP_EQUAL` （未显示）价差堆栈顶部的两个元素。这里是检查由Bob提供的公钥的hash值和Alice在第一交易中获得的公钥Hash值是否相等。 `OP_EQUAL` 将栈顶参与比较的两个值出栈，并把它们的比较结果（0表示false，1表示true）压入堆栈。
   `OP_VERIFY` （未显示）检查栈顶元素。如果这个值是false，它立即结束校验过程，当前交易校验结果失败。否则它将栈顶的true元素弹出。
6. 最终，Alice的公钥脚本执行 `OP_CHECKSIG` ，通过刚刚检验过的Bob提供的公钥来检查Bob提供的用私钥产生的签名。如果签名跟公钥匹配，并且其中的内容时使用都是需要签名的数据， `OP_CHECKSIG` 把true添加到堆栈顶部。

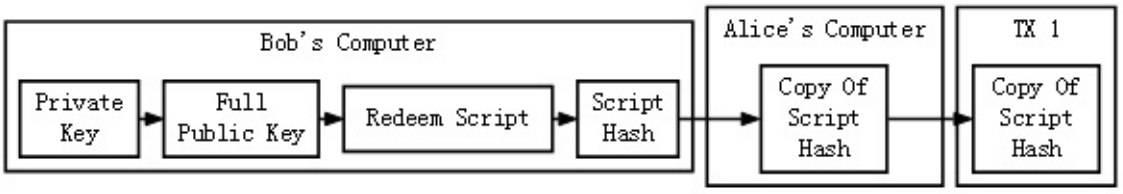
如果执行完公钥脚本后最终栈顶元素不是false，则判定交易有效（假定这里交易没有其他的问题）。

P2SH脚本

公钥脚本由支出者创建，支出者对脚本内容的兴趣其实不大。实际收款人才是真正关心脚本条件的人，并且如果他们愿意，他们可以让支出者是哟个指定的公钥脚本。不幸的是，配置的公钥脚本没有较短的比特币地址简单，并且没有标准的方式能够把他们和BIP70（后面进行讨论）广泛应用前的程序结合起来。

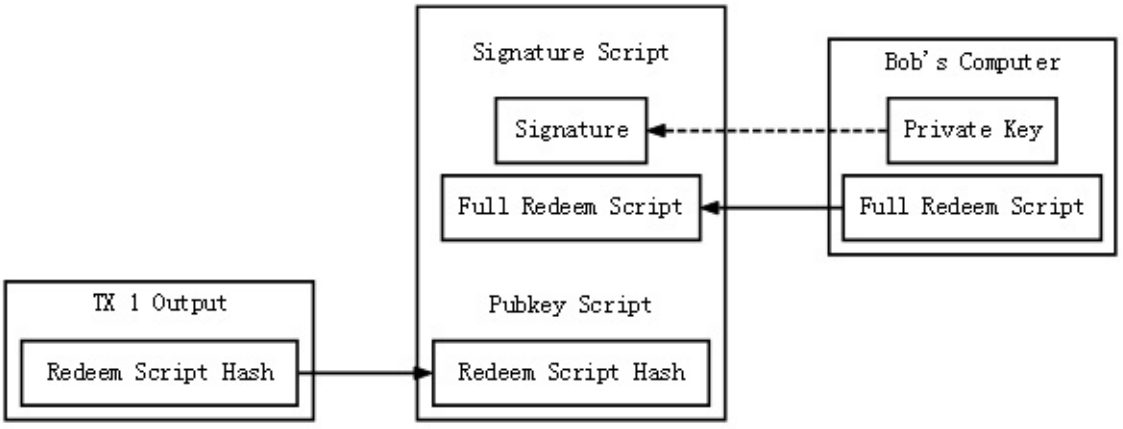
为了解决这些问题，P2SH（pay-to-script-hash，支付到脚本hash）交易在2012年产生，让支出者创建一个包含兑换脚本hash值的公钥脚本。

基本的P2SH工作流程示例如下，它看起来和P2PKH工作流几乎相同。Bob使用自己喜欢的任意的脚本作为兑换脚本，计算它的hash值，然后将兑换脚本的hash值提供给Alice。Alice创建一个P2SH版本的输出，其中包含了Bob的兑换脚本的hash值。



Creating A P2SH Redeem Script Hash To Receive Payment

当BObo想花掉这笔输出，他在签名脚本中将完整的兑换脚本和签名一起提供。p2p网络保证完整的兑换脚本hash值和Alice在她的输出中存入的hash值相同，然后就像执行公钥脚本一样执行该兑换脚本，即如果兑换脚本返回非false则允许Bob花费这笔输出。



Spending A P2SH Output

兑换脚本的hash值和公钥的hash值具有同样的特点，因此它也可以一种比特币地址格式，但是为了区分它和标准地址格式需要对它做一些小的改变。存储P2SH格式的地址和存储P2PKH格式的地址一样简单。如果P2PKH公钥hash一样，它的hash值同样会隐藏公钥值，所以二者同样安全。

标准交易

比特币在发现了几个早期版本的危险Bug之后，增加了一个测试。该测试仅仅来自指信任网络的交易，信任网络需满足1)公钥脚本和签名脚本匹配一个信任模板小集合，2)该网络的其他交易没有违反另外一个保证网络行为正常的集合的内容。这就是`IsStandard()`测试，传递给该测试的交易称为标准交易。

非标准交易是指那些未通过测试的交易，它们仍然可能被没有使用默认比特币核心设置的节点接受。如果他们被包含到区块当中，同样需要避免`IsStandard()`测试和被处理。

除了使通过广播有害交易方式攻击比特币变得更加困难外，标准交易测试还可以帮助防止“用户今天创建区块使未来增加新的交易特性更加困难”的情况。举例来说，如上面所述，每个交易都包含一个版本号，如果用户任意修改版本号的数值，那么未来使用版本号作为后向兼容特性工具的措施就会失效。

对于比特币核心0.9，标准的公钥脚本类型如下：

P2PKH（支付给公钥hash，Pay To Public Key Hash）

P2PKH是用来发送交易到一个或多个比特币地址最常用的公钥脚本格式。

```
Pubkey script: OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
Signature script: <sig> <pubkey>
```

P2SH（支付给脚本Hash，Pay To Script Hash）

P2SH用于将一笔交易发送到一个脚本hash地址。每个标准的公钥脚本都可以作为一个P2SH兑换脚本，但是实际应用中只有多签名公约脚本有效，其他的暂时都不是标准交易格式。

```
Pubkey script: OP_HASH160 <Hash160(redeemScript)> OP_EQUAL
Signature script: <sig> [sig] [sig...] <redeemScript>
```

多签名（Multisig）

尽管P2SH多签名现在通常用于多签名交易，这种基础脚本也可以在UTXO可被消费前用于请求多重签名。

在多签名公共脚本中，被称为m-ofn，m代表满足公钥的最少签名个数；n代表提供的公钥的个数。m和n应为`OP_1`到`OP_16`操作码对应的数字之一。

由于必须对原始比特币实现中的off-by-one错误保持兼容性，`OP_CHECKMULTISIG`比对应的m多消耗一个参数，所以签名脚本中对应的secp256k1 签名需要以一个额外的`OP_0`操作码开始，该值被消耗弹出但是实际不使用。

签名脚本提供签名中对应公钥需要和公钥脚本及兑换脚本以同样的顺序，`OP_CHECKMULTISIG`细节参见下面的描述：

```
Pubkey script: <m> <A pubkey> [B pubkey] [C pubkey...] <n> OP_CHECKMULTISIG
Signature script: OP_0 <A sig> [B sig] [C sig...]
```

下面是一个2-of-3的P2SH多重签名的示例：

```
Pubkey script: OP_HASH160 <Hash160(redeemScript)> OP_EQUAL
Redeem script: <OP_2> <A pubkey> <B pubkey> <C pubkey> <OP_3> OP_CHECKMULTISIG
Signature script: OP_0 <A sig> <C sig> <redeemScript>
```

公钥

公钥输出是P2PKH公钥脚本的一种简单形式，但是不如P2PKH安全，现在已基本不再使用。

```
Pubkey script: <pubkey> OP_CHECKSIG
Signature script: <sig>
```

空数据(待修正)

从比特币核心0.9.0版本开始空数据交易默认进行转发和挖掘，后来它增加了使用任意数据来确认为未花费公钥脚本，全节点不需要把这些数据保存在它们的UTXO数据库中。空数据交易不能被自动裁剪，因此鼓励在交易中使用空数据交易来扩充UTXO数据库。但是，如果可能的话把数据保存在交易的外是更好的。

在没有违反其他一致性规则的前提下，允许的空数据输出最大为公钥脚本大小10,000字节，因此任何推送数据都不能超过520字节。

默认情况下，比特币核心0.9.x到0.10.x单个数据推送可转发和挖掘的空数据交易可以达到40字节，并且只有一个空数据输出支付金额为0。

```
Pubkey Script: OP_RETURN <0 to 40 bytes of data>
(Null data scripts cannot be spent, so there's no signature script.)
```

比特币核心0.11.x把这个默认值增加到80字节，同时其他规则保持不变。

比特币核心0.12.0版本，在没有超出总字节限制的前提下，默认的转发和挖掘的空数据输出可达83字节及任意长度的数据推送。这里还是需要保证只能有1个空数据输出并且它的支付金额为0。

比特币核心配置项`-datacarriersize`允许设定你愿意转发和挖掘的最大空数据输出大小。

非标准交易

如果你不遵循输出中使用标准公钥脚本，使用默认比特币核心配置的节点和矿工将不会接受、广播，也不会打包你的交易。当并试图向你使用默认比特币核心配置伙伴节点广播你的交易时将会收到一个错误。

如果你创建一个兑换脚本，对它做hash，然后在P2SH输出中使用这个hash值，配比特币网络只能看到这个hash值，此时无论兑换脚本内容如何他们都会认为这个交易有效并接受它。这样就可以允许非标准脚本支付，从比特币核心0.11开始，几乎所有的有效兑换脚本都可以成功支付。但是使用了未标记的NOP操作符的脚本除外，这些操作符被保留待未来软分叉使用，它们只能被不遵循标准矿池协议的节点处理。

> 注意：标准交易的涉及是为了保护比特币网络，不是阻止用户犯错。创造一个比特币不可消费的标准交易是很容易的。

从比特币核心0.9.3开始，标准交易必须遵循如下条件：

- 交易必须是可终止的。要么它的锁定时间在过去（或小于等于当前区块高度），要么它所有序列号都是0xffffffff。
- 交易大小必须小于100,000字节。这大约是典型单输入单输出P2PKH交易的200倍。
- 交易中的每个签名脚本都必须小于1650字节。这各大小已经足够采用压缩公钥的15-of-15的P2SH多签名交易使用。
- 需要超过3个公钥的裸多签名交易（非P2SH）现在是非标准的。
- 交易签名脚本只能向脚本执行栈中压入数据和只有压入数据功能的执行码，不能压入其他类型的执行码。
- 当接收少于典型交易花费的1/3时，交易当中不能包含任何输入。在比特币核心采用默认中继费用时，当前对于P2PKH或P2SH输出该值为546聪。这里有各例外，标准的空数据交易只能接收0聪。

签名Hash类型

`OP_CHECKSIG` 从每个它要验证的签名中抽取一个非栈参数，允许签名者决定对交易的哪个部分签名。由于签名可以保证这些被签名的部分不被更改，这可以让签名者选择是否允许其他人修改他们的交易。

指定签名元素的选项称作hash类型，当前共有3中基本的SIGHASH类型可用：

- `SIGHASH_ALL` ，默认的，对所有的输入和输出签名，保护除了签名脚本外其他所有内容不被更改。
- `SIGHASH_NONE` ，所有输入签名，所有输出不签名，除非使用其他的签名类型进行签名保护输出，否则即为允许任何人可以更改比特币的趋向。
- `SIGHASH_SINGLE` ，仅对本输入对应的输出进行签名（拥有和输入索引相同的数字的输出

索引），确保任何人都不能更改交易中属于你的部分，但是允许其他人改变该交易中属于他们自己的部分。对应的输出必须存在或者破坏安全方案将值“1”签名。所有的输入都被包含在签名中，但是其他输入对应的输入都没有被包含在签名中，都是可被修改的。

基本的hash类型可以和 `SIGHASH_ANYONECANPAY` (any one can pay)组合，产生三种新的组合类型：

- `SIGHASH_ALL | SIGHASH_ANYONECANPAY`，签名所有的输入和当前输入，允许其他任何人添加或移除其他输入，所以任何人都能贡献额外的比特币，但是他们不能控制发送的数量和目的地。
- `SIGHASH_NONE | SIGHASH_ANYONECANPAY`，签名当前输入，允许其他任何人添加或移除其他的输入和输出，所以任何人获取当前输入的副本后都可以任意花费。
- `SIGHASH_SINGLE | SIGHASH_ANYONECANPAY`，签名当前输入和对应的输出。允许任何人添加和移除他们自己的输入。

由于每个输入都被签名，一个多输入交易可以有多种签名类型来签名交易的不同部分。比如，一个单输入交易使用None签名可以允许它的输出被把该交易放到区块链上的矿工修改。另一方面，一个辆输入的交易可以有一个签名为None，另一个签名为All，签名为all的签名者可以单独决定如何花费交易中的金额，但是没人能修改交易中的内容。

锁定时间和序列号

所有的签名hash类型都进行签名的一个字段是locktime。（在比特币核心代码中称为nLocakTime。）locktime代指交易能被添加到区块链的最早时间。

locktime允许签名者创建一个时间锁交易，该交易仅在未来有效，给交易双方可以更改决定的机会。

如果任何一个签名者改变了想法，他们可以创建一个新的非锁定交易。新的交易将使用原来locktime输入使用的相同的输出。如果新的交易在锁定时间前加入到区块链，这样将会使之前提交的locktime交易失效。

之前版本的比特币核心提供了一个特性，防止交易签名者使用上面的方法取消时间锁定交易，但是该特性的一个必须方面为了防止否认服务攻击很快被停止了。由该特性引入的为每个输入指定一个4字节序列号被保留下来。序列号意味着允许多重签名者统一更新这个交易，当他们完成对交易的更新以后，他们可以统一把所有的输入序列号设置为4字节最大值0xffffffff，一次允许该交易打破时间锁限制添加到区块链。

即使现在，设置序列号为0xffffffff（比特币核心默认）仍然可以打破时间锁；因此，如果你想使用时间锁，则至少一个输入的序列号需要设置为小于0xffffffff的值。由于序列号现在已不应用在其他地方，可以在设置时间锁的时候把序列号设置为0。

所动时间是一个无符号4字节证书，可以以两种方式解释：

- 如果小于500,000,000，锁定时间会被认为是区块高度。交易可以被添加到高度大于等于该值的区块中。
- 如果大于等于500,000,000，锁定时间被认为是unix时间戳（从UTC 1970年1月1日0时0分0秒到现在的秒数，现在已经超过1395000）格式的表示。交易可以被添加到块时间大于该锁定时间的区块中。

交易费用和找零

交易费用的支付基于签名交易总的字节大小。每字节费用是通过当前区块空间需求程度决定的，需求程度越大费用越高。交易费用被支付给比特币矿工，前面 `区块链` 一节已经提到，最终交易费用是由比特币矿工选择他们能够接受的最小交易费用决定。

这里还有一个概念称为“高优先级交易”，通过花费更多的交易费用来避免等待较长时间。

在之前 ，这种优先交易不跟正常交易费用需求一起处理。在比特币核心0.12以前，每个区块预留50KB给这些高优先级交易，但是现在已经默认把预留空间设置为0。不使用专门的优先级区域后，所有的交易按它们的每字节交易费用排列优先级，按照单字节交易费用由高到低的顺序添加到区块中直到填满区块所有空间。

从比特币核心0.9开始，为在比特币网络中广播交易指定了最小费用（当前是1000聪）。任何只支付最小交易费用的交易需要做需要等待较长时间才能被打包到稀缺的区块空间当中的准备。关于这种策略的重要意义，请查看 `验证交易支付` 一节。

由于每个交易都要花费UTXOs，而每个UTXO都只能被花费一次，因此UTXO当中的比特币必须一次花费完或作为交易费用支付给比特币矿工。很少有人持有的UTXOs中比特币的个数就是他们刚好要支付的个数，因此绝大多数交易都有一个找零输出。

找零输出是消耗一个UTXO支出后的剩余返还给支出者的常规输出。它们可以复用与UTXO相同的P2PKH公钥hash或P2SH脚本hash，但是考虑到下节将要讨论的原因，强烈建议找零输出使用一个新的hash地址。

避免**Key**重用

在交易当中，支出者和接收者都拥有彼此在交易中使用所有的公钥或地址。他们中的任何一个人都可以利用这些信息在公共区块链上追溯对方过去和未来使用相同的公钥的交易。

如果经常重用同一个公钥，这在人们使用一个地址（hash的公钥）作为比特币固定接收地址时容易出现，其他人可以很容易的追溯该地址的比特币来源和去向，以及当前该地址控制多少比特币。

上面的问题很容易解决。如果一个公钥只使用两次，一次接收，一次消费，那么用户就可以拥有很多的金融安全性。

如果更进一步，如果每次接收付款或找零时都使用一个新的公钥或不重复的比特币地址，可以跟其他技术结合（如下面将要讨论的交易联合或融合避免），使仅通过区块链本身可信地追溯用户收支过程十分困难。

避免蜜月重用还可以防止从公钥或签名对比（在特定情况下，现在更可能出现假定）来重建私钥的安全攻击。

- 非重用不重复的P2PKH和P2SH地址可以防止第一种类型的攻击。直到第一次比特币付款时才把ECDSA公钥暴露出来，除非攻击者能在一两个小时以内即区块未受妥善保护前重建私钥，否则就是无用的。
- 非重用不重复的私钥可以防止第二种类型的攻击。每个私钥只产生一个签名，因此攻击者无法获得试用一个私钥生成的一些列签名来进行比对攻击。当前的比对攻击只有当随机性不够或随机状态被破解时才会有效，比如边带攻击。

因此，无论从隐私角度，还是从安全角度，我们都建议你在创建自己的程序时避免重用公钥；如果可能，让用户优先选择使用新地址。如果你的程序要提供一个固定的接受付款的URI，可以参照下面URI一节的 `bitcoin:` 。

交易重塑

所有比特币签名hash类型都不保护签名脚本，这样就留下了一个受限的否定服务的后门，成为交易重塑。签名脚本包含secp256K1，它不能签名自己，这就给攻击者在不改变交易校验结果的情况下伪造一个非功能性修改。比如，攻击者可能在签名脚本中添加一些数据，这些数据被前一个交易的公约脚本处理掉。

尽管修改是非功能性的，不会改变交易使用的输入和输出结果，但是它们改变了交易的hash值。由于每个交易和前一个交易都是通过或hash得到的交易标识相互关联，修改以后得到的交易标识是交易创建者预料之外的。

这对大多数比特币立即添加到区块链的交易来说，这不是问题。但是对于使用该笔交易的后续交易先添加到区块链的情况就有问题了。

比特币开发者一直致力于减少标准交易中的重塑交易，其中的一个成果是BIP 141：隔离见证，该技术已经被比特币核心支持但是尚未启用。目前，新的交易不能依赖未添加到区块链的历史交易，尤其是至关重要的大宗比特币交易。

交易重塑同样影响支付追踪。比特币核心RPC接口允许用户通过txid追踪交易，但是如果由于交易内容更改导致txid改变，就可能导致交易在区块链上消失。

当前交易追踪最佳实践规定，新交易应该可以通过作为其输入的历史交易的输出追踪到，因为只要历史交易不失效该交易就不能更改。

最佳实践进一步规定，如果一个交易可能真的从网络中消失了，那么它需要重新发布，发布的方式应使之前的旧交易失效。一种可行的方式为保证重新发布的交易把丢失交易使用的交易输出全部花完。

对应英文文档

合约

合约是使用去中心的比特币系统保证商业约定的交易。比特币合约能够减少对外部机构（如法院系统）的依赖，以此可以显著在金融交易中与未知实体交涉的风险。

接下来的章节中，将会介绍现在正在使用的多个合约。由于合约处理的不只是交易，更是人与人之间的问题，因此使用一个故事将下面内容组织起来。

除了下面介绍的合约类型外，还有很多其他已经提出的合约。其中一些被整理到比特币Wiki的合约页了。

公证和仲裁

顾客Charlie想从老板Bob这里购买意见商品，但是他们互不信任，所以他们使用合约来保证Charlie可以得到他的商品并且Bob可以得到他的货款。

用一个简单的合约可以这样描述：charlie将要支付一定数量的比特币作为交易输出，而且只有当charlie和Bob都签名允许它可被再次消费时它才是可用的（作为未来交易的输入）。这意味着只有Charlie得到他的商品，Bob才能获得这些比特币；同时Charlie也不能既得到商品又拿回付款。

在发生争执时该合约很难继续执行，因此Bob和Charlie引入了仲裁者Alice的帮助，创建一个公证合约。charlie将会把他货款支付到一个交易的输出，该输出只有在3人中的2人同意后才能作为未来交易的输入。现在如果没有其他问题，charlie可以支付货款，bob也可以拿到货款；如发生了冲突，Alice可以仲裁决定谁可以得到货款。

为了创建一个多重签名的输入，他们需要给出自己的公钥，然后Bob创建下面的P2SH多重签名兑换脚本：

```
OP_2 [A's pubkey] [B's pubkey] [C's pubkey] OP_3 OP_CHECKMULTISIG
```

（把公钥压栈的操作码没有写出）。

OP_2和OP_3是把对应的数字2和3压栈。OP_2指定有两个签名需要签署；OP_3指定有3个公钥（未hash的）需要提供。这是一个2-of-3的多重签名公钥脚本，或更一般的称作m-of-n公钥脚本（m是匹配签名需要的最少个数，n是需要提供的公钥的个数）。

Bobo把兑现脚本提供给Charlie，charlie需要检查确认它的公钥和Alice的公钥都已包含在其中。然后把兑现脚本hash后创建一个P2SH兑换脚本然后把货款支付到该地址。Bob在区块脸上看到该交易被添加，就把商品交付给charlie。

不幸的事情发生了，商品在运输的过程中有轻微损坏。charlie想要全额退款，但是Bob认为10%的退款比较合理。他们找到Alice解决这个问题。Alice从charlie那里索要了照片证据以及一份Bob创建并让charlie检查过的兑换合约的副本。

通过查看证据，Alice认为40%的退款是比较合理的，因此它创建并签署了一份两输出的交易，一个支付60%的比特币到Bob的公钥，另一个把剩余40%支付到charlie的公钥。

在签名脚本中Alice添加了她的签名和和一份未hash的连在Bob创建的兑换脚本后面的新兑换脚本。它把这个不完整的交易同时提供给Bob和Charlie。他们两个人中的任一个都可以把自己的签名添加到下面的签名脚本中完成这个交易。

```
OP_0 [A's signature] [B's or C's signature] [serialized redeem script]
```

（入栈签名和兑换脚本的操作码这里没有给出。OP_0是为了规避一个早期实现off-by-one错误的变通方法，处于兼容性考虑不能省略）注意，签名脚本一定要按照出现兑换脚本的公钥相同的顺序提供，更多细节参考 `OP_CHECKMULTISIG` 的描述。）

当交易被广播到网络中，每个节点都会通过签名脚本检查charlie之前支付的P2SH输出，保证兑换脚本与之前提供的兑换脚本hash一致。然后使用两个签名作为输入，执行兑换脚本。假定兑换脚本有效，两个交易输出就会分别出现在Bob和Chlie的钱包当中成为可消费现金。

但是，如果Alice创建和签名了一个他们两人都不能接收的方案，比如把所有的比特币都给她自己，Bob和charlie可以重新找一个仲裁人签署一份新的交易，把所有的比特币使用另一个2-of-3多重签名兑换脚本hash支付出去，新的交易当中包含第二个仲裁人的公钥。这就意味着Bob和Charlie不需要为仲裁人可能投走他们的资金担心。

资源：BitRated在遵循GNU AGPL协议网站上使用html/Javascript提供了一个多重签名仲裁服务的接口。

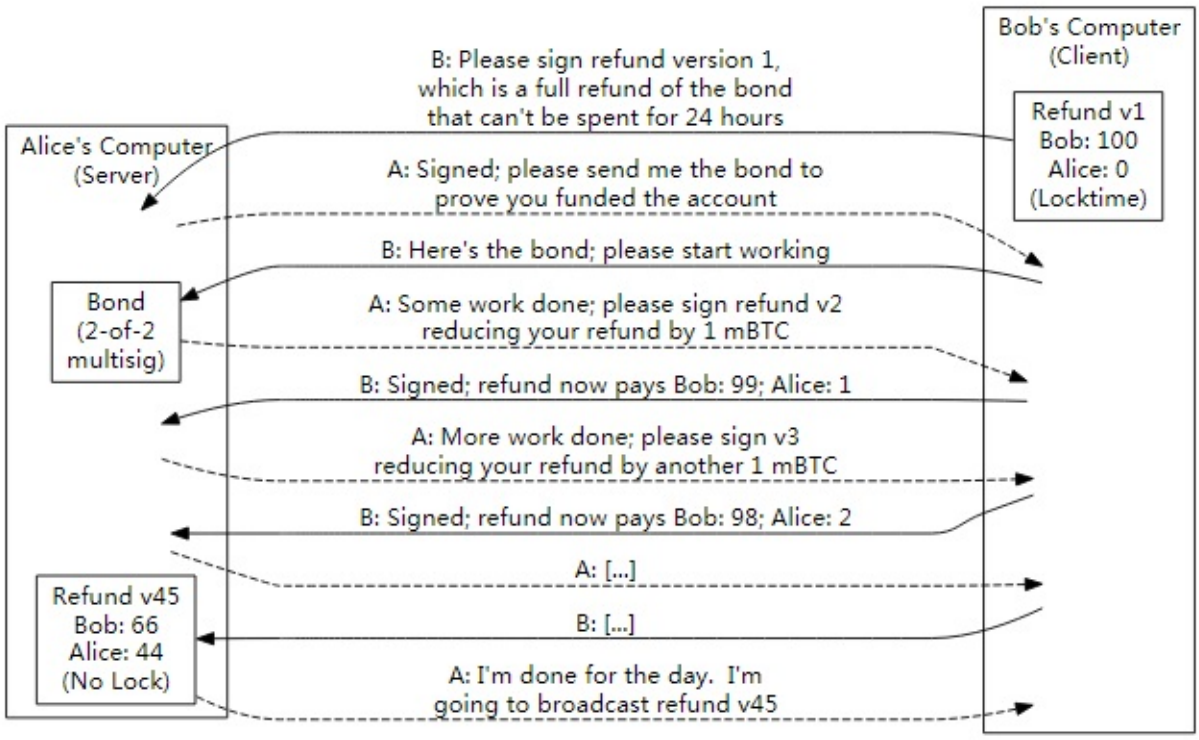
小额支付通道

Alice为Bob的论坛做兼职发言审核。每次有人在Bob繁忙的论坛上发言时，Alice都会检查发言内容是否有攻击性或者是垃圾信息。Bob经常忘记为Alice支付薪水，因此Alice要求Bob在每次审核完一个发言后Bob都立即把薪水支付给她。Bob说，这样不行，数百笔的支付交易需要花费它上千聪的交易费，因此Alice建议他们使用小额支付通道。

Bob想Alice索要她的公钥，然后创建2个交易。第一个交易使用2-of-2多重兑换签名通过P2SH输出支付100聪，多重签名同时需要Alice和Bob的签名才有效。这是债券（bond）交易。把这个交易广播到网络里是让Alice锁定这些押金，接着bob把这个交易暂时作为私有（不广播），

然后创建第二个交易。

第二个交易在24小时的锁定时间之后把第一个交易的所有输出支付给Bob，是一个退还交易。Bob不能自己签署这个退还交易，他还需要把这个交易交给Alice签名，过程如下图所示：



Alice broadcasts the bond to the Bitcoin network immediately.
She broadcasts the final version of the refund when she finishes work
or before the locktime. If she fails to broadcast before refund v1's time lock
expires, Bob can broadcast refund v1 to get a full refund.

Bitcoin Micropayment Channels (As Implemented In Bitcoinj)

Alice检查这个退还交易的锁定时间是否是未来24小时之后，如果没有问题就签署它，并把签署后的副本发送给Bob。然后她从Bob那里拿到在全交易，并检查退还交易是否是消费的债券交易的输出。然后，她把债券交易广播到网络上，以此保证Bob必须等到过期时间以后才能花费这笔资金。迄今为止，Bob除了一点交易费用外还没有其他额外花费，他也可以在24小时候广播退还交易取回所有资金。

现在，Alice每做完价值1聪的比特币的工作，他都会要求Bob创建和签署一个新的退还交易。新版本的交易将1聪的比特币支付个Alice，剩余的99聪退还给Bob；新交易并没有锁定时间，因此只要Alice愿意她可以随时签署并得到该笔支付（通常Alice不会立即使该交易生效）。

Alice和Bob这个工作-支付过程，知道Alice完成了一天的工作或时间锁即将超期。Alice签署最新版本的退还交易并广播出去，拿到自己的报酬并将剩余资金退还给Bob。第二天当Alice再次开始工作时，他们就会创建一个新的小额交易通道。

如果Alice在时间到期前没有把最终版本交易广播出去，Bob就可以把他拥有的第一个版本的交易广播并得到所有的资金。这是小额交易渠道仅适用于小额支付的原因，如果Alice的网络在时间到期前离线几个小时，她所有薪水就可能打水漂了。

上一节讨论的交易重塑是限制小额支付渠道使用的另一个原因。如果某些人利用交易重塑打破两个交易之间的联系，Alice即使什么工作也不做，也可以吧Bob的100聪押金锁定。

对于大额支付，交易费用只占总费用很小的比例，所以为了保证支付安全将多个交易立即广播更有意义。

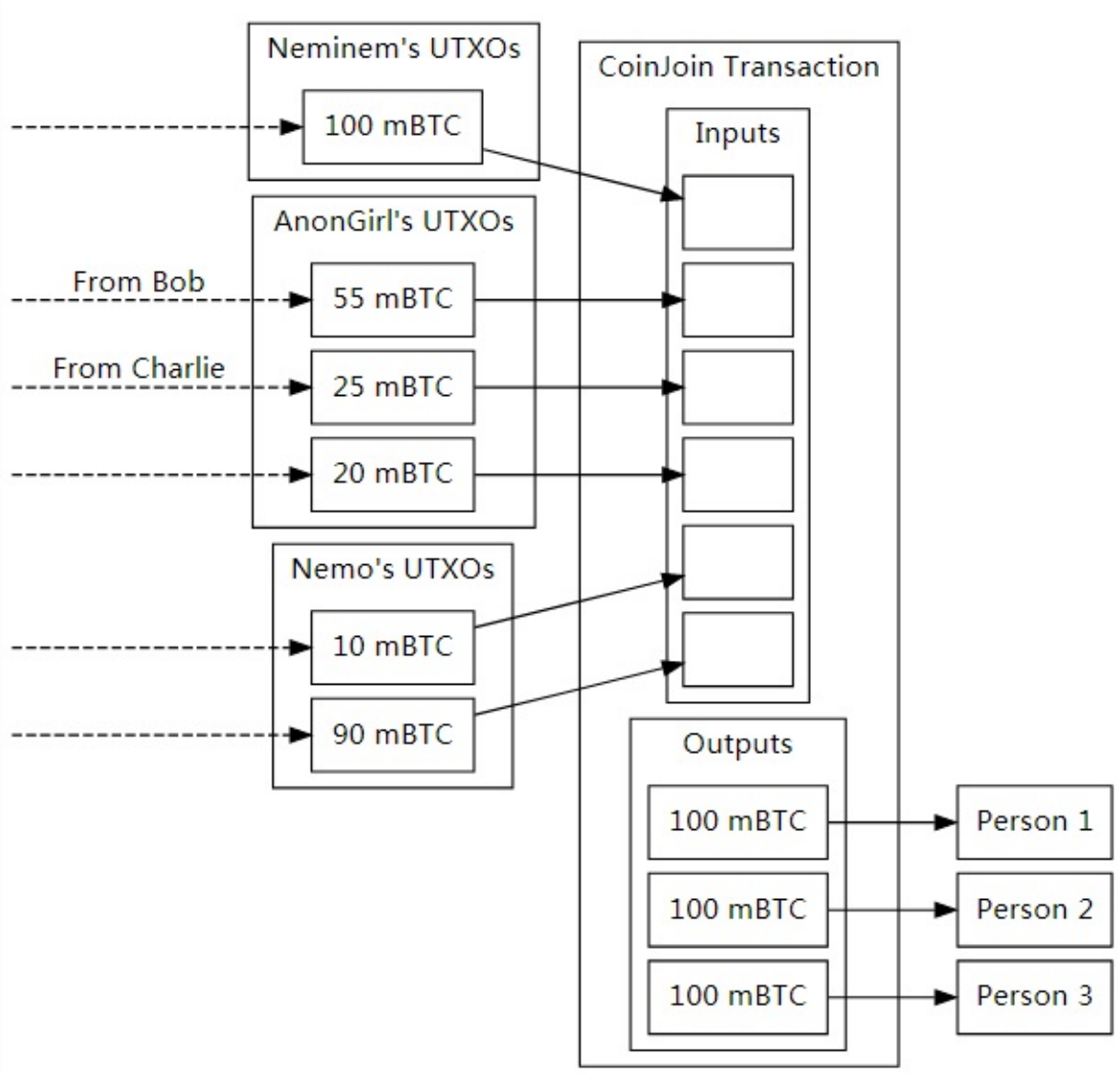
资源：bitcoinj java库提供了一套完整的设置小额交易的函数、一个实现例子，以及一个使用Apache 许可证的教程。

联合交易

Alice非常关注自己的隐私。她知道自己的每笔交易都被添加到了公共区块链上，因此当Bob和Charlie向她支付的时候他们可以轻易的最终这些比特币来获取她的支付地址、收款金额，甚至可能知道她有多少余额。

Alice不是一个罪犯，她只是想对她曾经花费了多少比特币以及现在剩余多少不要太过明显。因此她通过自己电脑上的洋葱匿名服务使用AnonGirl的账号登录到一个IRC聊天室。

在聊天室中还有Nemo和Meminem，他们共同决定彼此之间交换一部分比特币以使他们周围的人不知道哪个交易的比特币是他们的持有的。但是他们面临一个两难问题：谁首先开始向两个使用假名的陌生人支付呢？下面将要介绍的联合交易将问题简单化：他们创造一个交易同时包含所有的支出，确保没人能偷走他人的比特币。



Example CoinJoin Transaction
Only the participants know who gets which output.

他们每个人通过查找他们的UTXOs找到一个100聪交易的组合。然后他们每人生成一个新的公钥，并把选定的UTXOs信息和公钥hash交给一个引导者。这个案例中，假设AnonGirl是引导者，她创造一个交易把所有的UTXOs支出为相等的3份，流入对应贡献者的公钥Hash。

然后AnonGirl使用SIGHASH_ALL标识把输入和输出都进行签名保证没人能够修改。然后他把这份部分签名的交易交给Nemo，nemo签名自己的输入后再传递给Neminem，neminm也做同样的操作。nemin把交易光波导P2p网络，把所有的比特币都混合到同一个交易当中。

如上所示，除了他们三人外没人能够确知谁拥有那份输出，所以他们都可以模糊自己的花费。

现在Bob或charlie试图通过区块链追踪Alice的交易，他们就会发现Nemo和Neminem的交易。如果Alice多做几次联合交易，Bob和charlie就必须要从成百上千个人当中猜测哪个交易是Alice做的。

Alice的完整的比特币交易历史依然保留在区块链上，如果一个专门的侦查员能够联系到所有跟anongirl进行联合交易的人，就能够发现她比特币的原始来源或许还能知道这个人就是Alice。但是在某人只是临时查看区块链历史的情况下，Alice还是可以隐匿行踪的。

上面讨论的联合交易技术需要支付少量的交易费用。另外的一项技术是购买者联合交易，可以在节约交易费用的同时提升隐私性。

anongirl一直在聊天室里等待直到她想买点东西。她公布了她想花费比特币的想法直到有人也想买点东西，可能是从另外一个商人那里。这是他们可以在购买前把它们的输入结合在一起，把输出分别指定到不同的商家的地址，此时就没人能仅从区块链直到他们每人到底买的什么东西了。

由于他们本来就必须要在他们的交易中支付费用，anongirl和她的协作者就不需要再花费额外的费用了。但是由于他们通过组合减少了交易的开销节省了存储，他们还可以花费更少的交易费用，每人再额外节省少量交易费。

资源：去中心化联合交易的一个优质实现时coinmux，基于Apache许可证。

对应英文文档

钱包

比特币的钱包可以用来指一个钱包程序，也可以是一个钱包文件。钱包程序创建公钥接收比特币，并通过相应的私钥花费这些比特币。钱包文件为钱包程序保存交易相关的私钥和其他信息。

钱包程序

全钱包

签名钱包

离线钱包

硬件钱包

纯分布式钱包

钱包文件

私钥格式

公钥格式

对应英文文档

支付处理

暂未翻译

Payment processing encompasses the steps spenders and receivers perform to make and accept payments in exchange for products or services.

对应英文文档

运行模式

暂未翻译

Currently there are two primary methods of validating the block chain as a client: Full nodes and SPV clients. Other methods, such as server-trusting methods, are not discussed as they are not recommended.

对应英文文档

P2P网络

比特币网络协议允许全节点（peers）之间协作地维护一个区块和交易交换的p2p网络。全节点在把转发前先校验每个区块和交易。归档节点市值可以存储整个区块链的区块链并可以为其他节点提供历史区块的全节点。剪枝节点是是不保存整个区块链的全节点。许多客户端通常使用比特币网络协议与全节点相连。

一致性协议并不包括网络校验，因此比特币程序可以调整网络和协议，比如一些矿工使用的高速区块交换网络、一些为钱包提供SPV级别安全的专门交易信息服务器等。

为了为比特币p2p网络提供一个使用的例子，本节当中使用比特币核心作为全节点的代表，BitcoinJ作为SPV客户端代表。这两个程序功能都是弹性的，因此这里只介绍基本功能。同时，出于隐私考虑，例子中出现的IP地址使用保留IP代替。

伙伴发现

第一次启动时，程序不知道任何的存活节点。为了发现一些IP地址，他们查询一个或多个在比特币核心和BitcoinJ程序中硬编码的DNS名字（称为DNS种子）。查询的结果将会包含一个或多个 DNS A记录，这些A记录会伴随一些可能接受新连接的全节点IP地址。比如，使用 `dig` 命令：

```
;; QUESTION SECTION:
;seed.bitcoin.sipa.be.        IN  A

;; ANSWER SECTION:
seed.bitcoin.sipa.be.    60  IN  A  192.0.2.113
seed.bitcoin.sipa.be.    60  IN  A  198.51.100.231
seed.bitcoin.sipa.be.    60  IN  A  203.0.113.183
[...]
```

DNS种子由比特币社区的成员维护：一些人提供动通过扫描网络自动获取活跃IP地址的动态DNS种子服务器，一些人提供静态手动更新的可能不太活跃的IP的DNS种子。在一些情况下，在指定主网络端口8333或测试网络端口18333运行的节点会被自动加入DNS种子当中。

由于DNS种子不是权威认证的，一些恶意种子提供者或网络中间人攻击可能只返回被攻击者控制的种子，从而把程序孤立在攻击者自己的网络中，以此来发起伪造协议和区块。由于这些原因，程序不能完全只依靠DNS种子。

一旦一个程序接入网络中，它的伙伴就开始向它发送 `addr` （地址）信息，其中个那些带了网络当中其他伙伴的端口和地址，提供了一种完全区中心的伙伴发现方法。比特币核心把当前已知的伙伴持久化保存在一个磁盘数据库当中，后面的启动中通常直接连接到这些伙伴而不再使用DNS种子。

但是，经常有伙伴离开网络或改变IP地址，因此程序可能启动前再一次成功的连接前发起多次尝试。这会为程序引入较大的延迟才能接入网络，迫使用户在发送交易或校验支付状态时等待很久。

为了避免这种可能的延迟，对于刚激活的节点，BitcoinJ通常使用动态DNS种子获取IP地址。比特币核心通常试图在最小化延迟和避免使用多余的DNS种子使用之间达到一个平衡：如果比特币核心在它的伙伴数据库当中有相应的记录，在使用种子前，它最多花费11s尝试连接至少一个伙伴；如果在这段时间内建立起一个连接，就不需要查询任何种子了。

比特币核心和BitcoinJ都包含一个硬编码的IP和端口列表，它们指向对应版本第一次发布时点附近活跃的节点。比特币核心在所有的DNS种子服务器在60s内没有响应时将开始尝试区连接到这些节点，提供一个自动的后备选项。

对于手动后备选项，比特币核心也提供了几个命令行连接选项，包括从通过IP指定的节点链接到一系列伙伴，或和某个通过IP指定的固定的节点维持一个持续连接。使用 `-help` 参数查看文本帮助信息。BitcoinJ也可以编程支持同样的操作。

资源：Bitcoin Seeder，该程序运行在比特币核心和BitcoinJ使用的种子服务器上。比特币核心DNS Seed Policy。比特币核心和BitcoinJ使用的硬编码的IP地址使使用makeseeds脚本产生的。

连接伙伴

连接是通过发送一个 `version` 信息到远端节点实现的，其中包括版本号、区块、当前时间。远端节点使用自己的版本信息回复。这样两个节点相互发送一个 `verack` 信息证实连接已经建立。

一旦建立连接，本地节点就可以向远端节点发送 `getaddr` 和 `addr` 信息获取更多的伙伴。

为了维护和伙伴的连接，节点默认在30分钟连接失效前向伙伴节点发送一条信息。如果90分钟内都没有收到一个伙伴的信息，节点就会假定伙伴的连接已经关闭。

初始区块下载

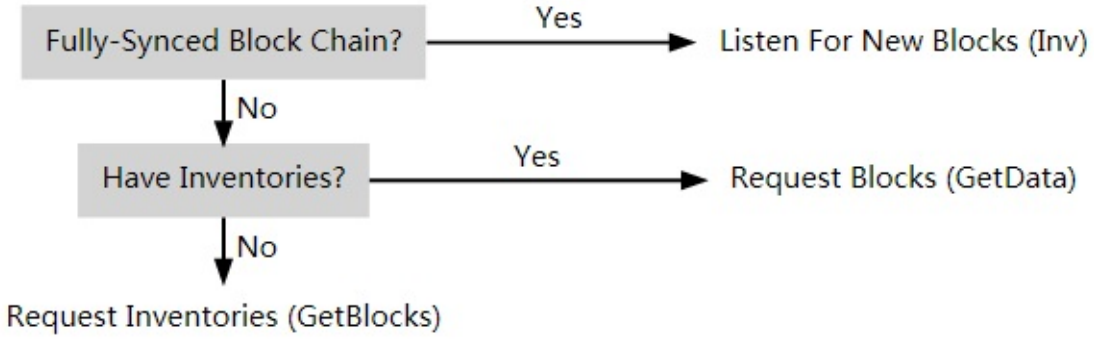
在一个全节点有能力校验一个未确认交易和最近新挖区块前，他必须下载并校验从第1个区块（硬编码的创始区块之后）开始到当前最优（可能存在分叉）区块链的所有区块。这被成为IBD（Initial Block Download，初始区块下载）或初始同步。

尽管“初始”这个词暗示这个过程只执行一次，但在任何时候有大量区块下载时候也都会使用它，比如之前接入的节点已经离线很久。这种情况下，节点可以使用IBD方式下载从它上次最后一次在线到现在的所有区块。

只要节点最优区块链上最新区块的区块头部时间距现在已经超过24小时时，比特币核心就会使用IBD方法。比特币核心0.10.0版本也会在本地最优区块链高度比本地最优头部链(header chain)的高度低144时（这也意味着本地区块链距离当前已经超过24小时）采用IBD。

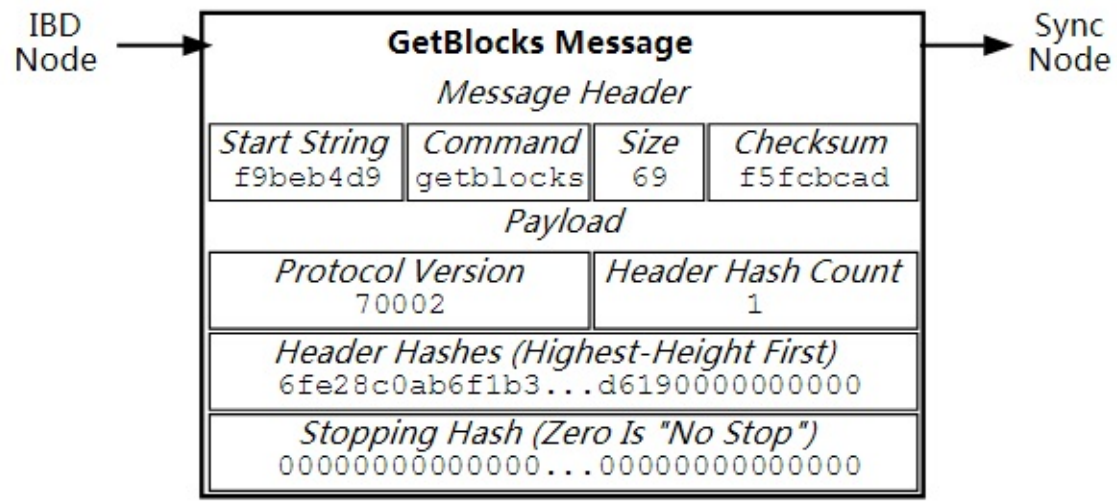
区块优先

直到0.9.3版本，比特币核心都采用一种我们称之为区块优先的简单IBD（Initial Block Download，初始区块下载）方式。它的目标是从最新区块链序列中下载完整区块。



Overview Of Blocks-First Initial Blocks Download (IBD)

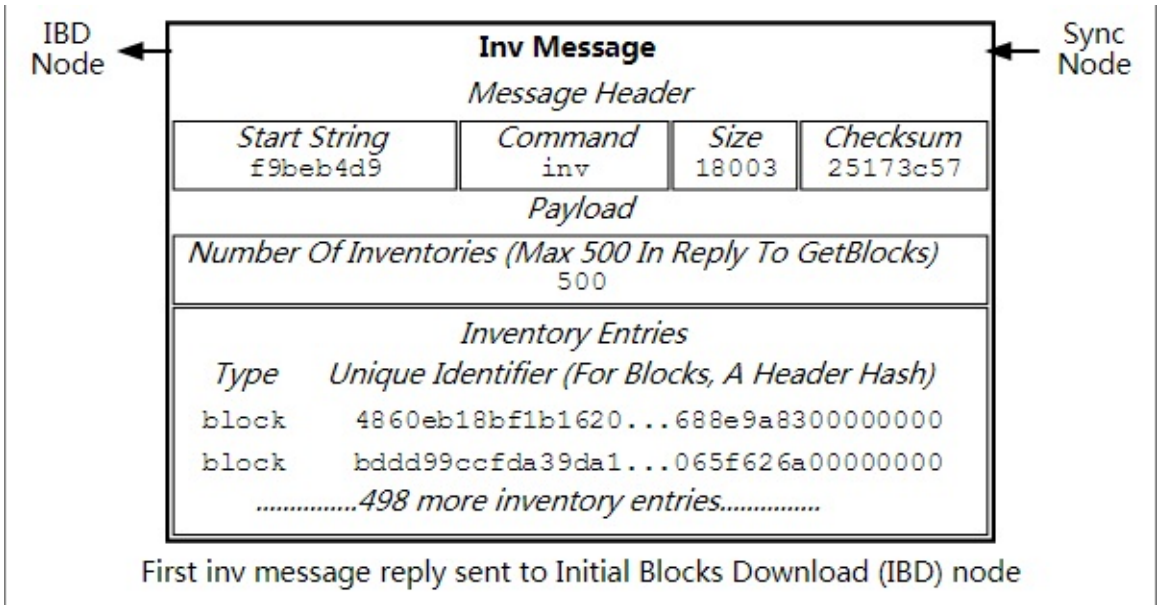
当节点第一次启动时，它的本地区块链只包含一个节点——硬编码的创始区块（区块0）。这个节点选择一个远端伙伴，成为同步节点，然后向它发送一个如下所示的 `getblocks` 消息。



First getblocks message sent from Initial Blocks Download (IBD) node

在 `getblocks` 信息的header hashs字段，新节点发送了它拥有的唯一的创始区块的头部hash值（6f32...0000，内部字节序）。此外它还把结束hash字段全部置零，以请求最大的响应。

当收到 `getblocks` 消息后，同步节点拿到头部hash在它的区块链中搜索那个hash值对应的区块。它发现区块0匹配，因此回复从区块1开始的500个区块的来列表（500是针对 `getblocks` 消息回复的最大值）。它把这些列表放在如下所示的 `inv` 消息中。

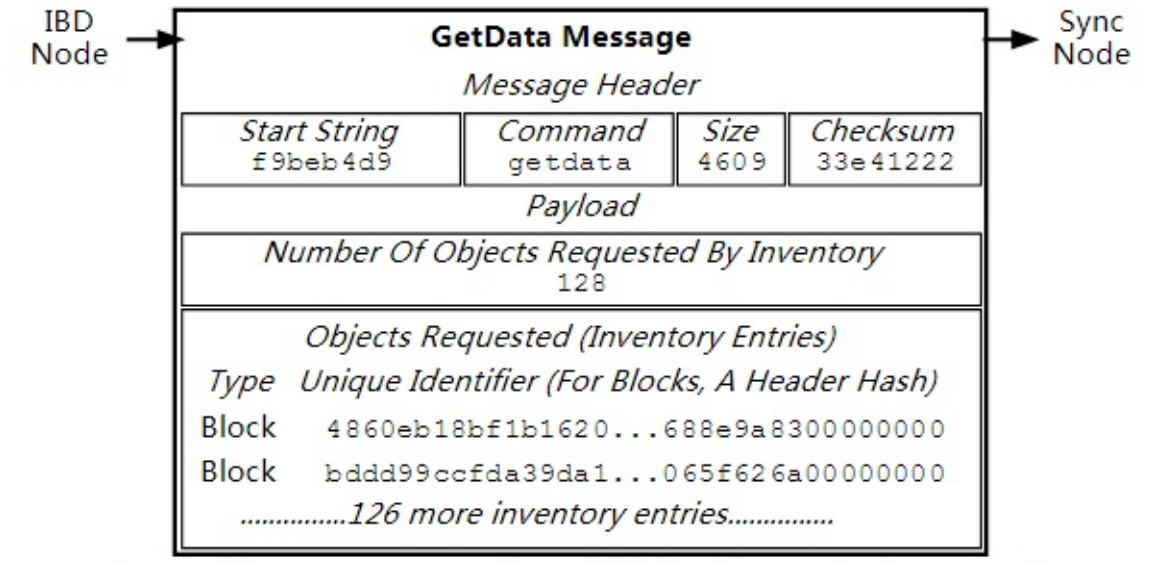


First inv message reply sent to Initial Blocks Download (IBD) node

列表项是网络上信息的唯一标识。每个列表包含一个对象实例的类型字段和唯一标识字段。对于区块，它的唯一标识是头部的hash值。

区块列表项出现在 `inv` 消息中的信息和它出现在区块链中的顺序一致，因此第一个 `inv` 消息中包含的表项是从区块1到区块501。（比如，上例中展示的区块1的头部hash为4860...0000）。

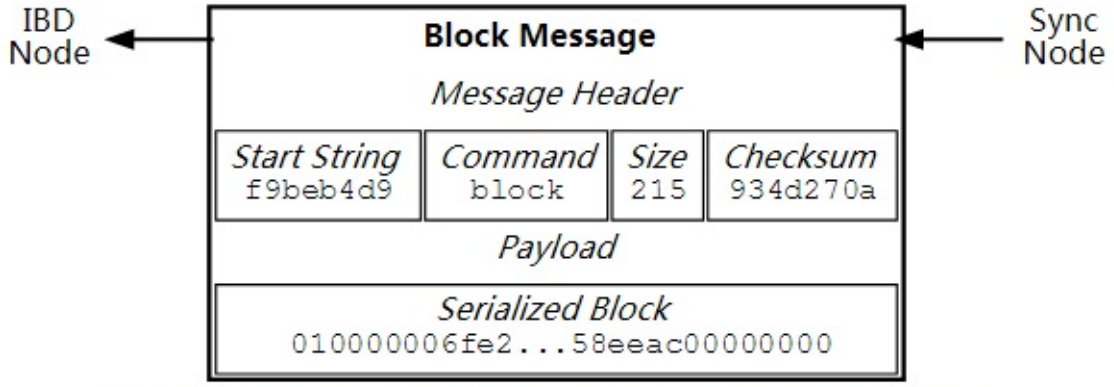
IBD节点使用受到的列表使用如下所示的 `getdata` 消息每次向同步节点请求128个区块。



First getdata message sent from Initial Blocks Download (IBD) node

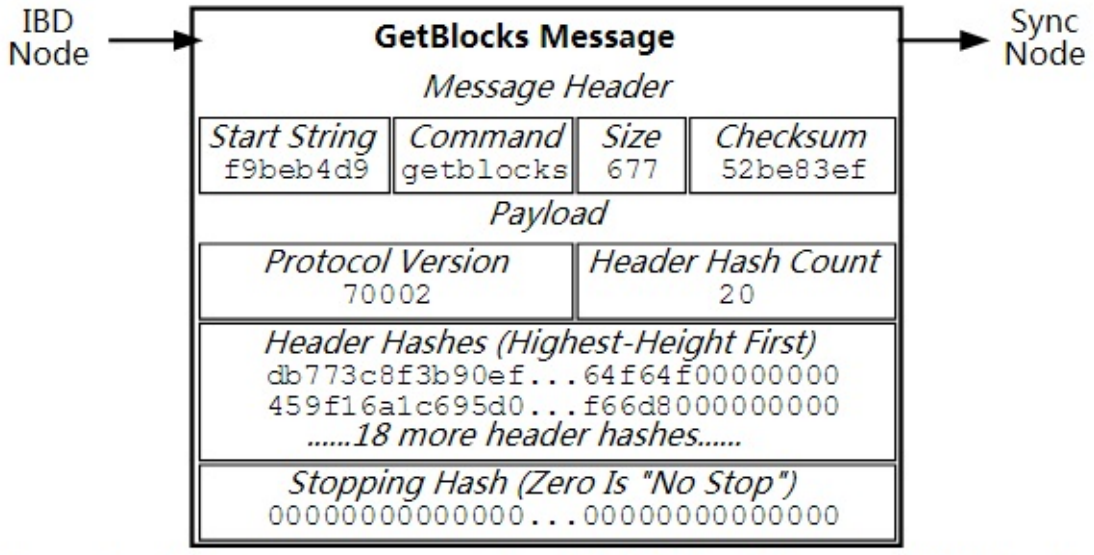
对于区块优先的节点来说顺序请求和发送区块非常重要，因为当前区块的头部要参照前面区块的hash值。这就意味着IBD节点只有收到父区块，才能完整校验当前区块。由于父区块没有收到导致无法校验的区块成为孤儿区块，下面有一小节将详细讨论。

一收到 `getdata` 消息，同步节点就会返回请求的区块。每个区块都会使用串行区块格式通过 `block` 消息分别发送。 `block` 消息的格式如下所示。



First block message sent to Initial Blocks Download (IBD) node

IBD下载每个区块、校验后，在请求下一个之前没有请求过的区块，并维护一个最大为128的待下载队列。当它接收到已有列表当中的所有区块后，将发送向同步节点再发送一个请求最多500个区块的列表 `getblocks` 消息。第二个请求如下所示，其中包含多个头部hash值。



Second getblocks message sent from Initial Blocks Download (IBD) node

一旦收到第二个 `getblocks` 消息，同步节点搜寻它的最优区块链，按照接收顺序依次逐个尝试，找到一个符合消息中头部hash的区块。如果找到一个匹配的，它将返回匹配点下一个区块开始的500个区块列表。如果除了停止hash外没有匹配的区块，它就假定两个节点相同的只有区块0，这是她就发送一个从区块1开始id `inv` 消息（和前面多次讨论的 `inv` 消息相同）。

重复的搜索搜索过程，即便IBD节点本地区块链和同步节点本地区块链分叉的情况下，仍可以让同步节点发送有用的列表。这种分叉检查在IBD节点越接近区块链的顶端时越有效。

当IBD节点收到第二个 `inv` 消息，它将使用 `getdata` 消息请求这些区块，同步节点会使用 `getblocks` 消息响应，然后IBD节点会使用另外的 `getblocks` 消息请求更多的列表。这种循环抑制重复知道IBD节点同步到当前区块链顶端。在那个时候，节点会接收正常的区块广播（下面的章节讨论）发送的区块。

区块优先的优点和缺点

区块优先最大的优势是简单，最大的劣势是IBD节点依赖单个节点下载所有区块。它们会存在如下问题：

- 速度限制：所有的请求都发送到同步节点，如果同步节点有上传带宽限制，IBD节点的下载速度也会很慢。注意：如果一个同步节点离线，比特币核心将会继续从另外一个节点下载，但是它仍然只会一次从一个同步节点下载。
- 下载重启：同步节点可以向IBD节点发送非最优（但有效）的区块链。IBD节点必须等到IBD过程接近完成时才能判断自己在非最优链上，强制IBD节点必须通过另一个节点再次重启区块下载过程。比特币核心在不同的区块高度（开发者选定）设置了多个区块链检查点，可以帮助IBD节点检查到自己正在接收一个可选的区块链历史，从而让IBD节点在下载过程的早期就重启下载。
- 磁盘错误攻击：类似于下载重启，如果同步节点同步非最优（但有效）的区块链，并被存储在IBD节点的磁盘上，不仅浪费磁盘存储空间还可能将磁盘用垃圾数据填满。
- 占用大量内存：无论是恶意为之还是意外，同步节点都可能乱序发送区块，导致产生故而区块，要等父区块收到并校验。孤儿区块在等待校验器件必须存储在内存中，这可能产生大量的内存占用。

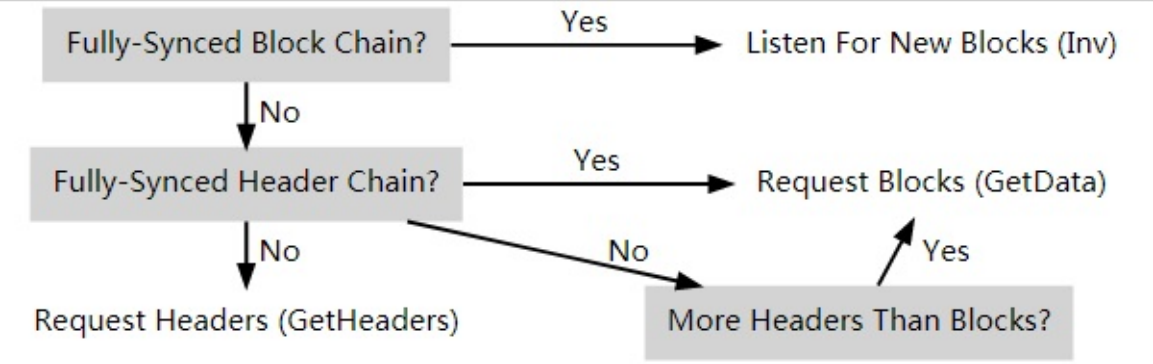
上述所有的问题已经被比特币核心0.10.0版本的块头优先IBD方法部分或全部地解决了。

资源：下面的表格总结了本节提到的信息。信息的连接可以带你查看对应信息的参考页面。

Message	getblocks	inv	getdata	block
From > To	IBD -> Sync	Sync -> IBD	IBD -> Sync	Sync -> IBD
内容	一个或多个头部hash	最多500个区块列表（唯一标识）	一个或多个区块列表	一个顺序的区块

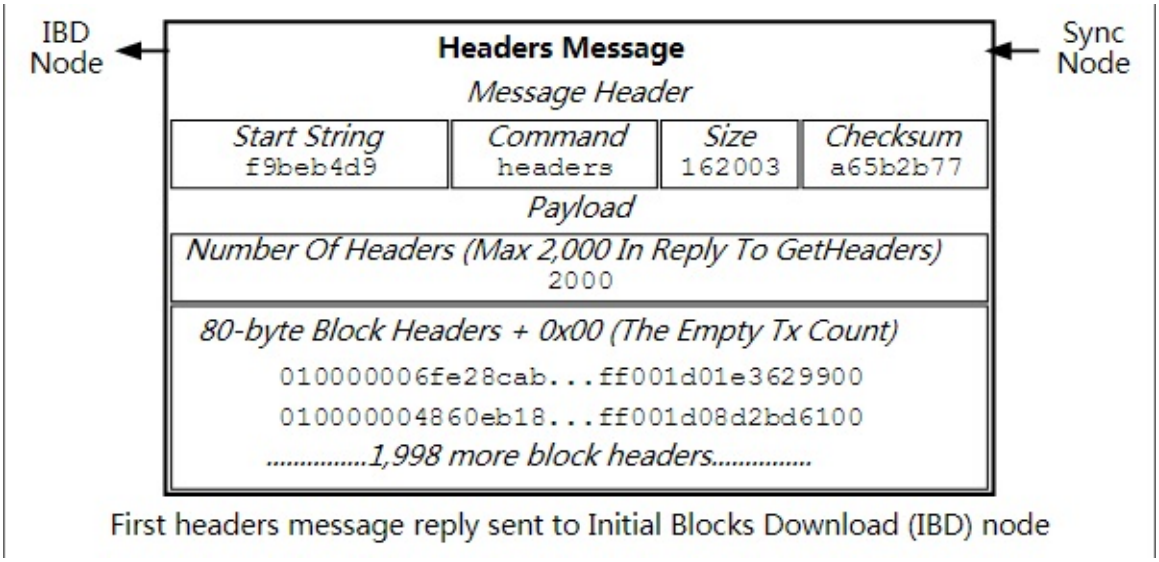
块头优先

比特币核心0.10.0使用了一种称作 `块头优先` 的IBD方法。目标是从最优块头链中下载块头，一边尽可能的校验，同时并行的下载对应的区块。这种方式解决了老的区块优先的IBD方法的许多问题。



Overview Of Headers-First Initial Blocks Download (IBD)

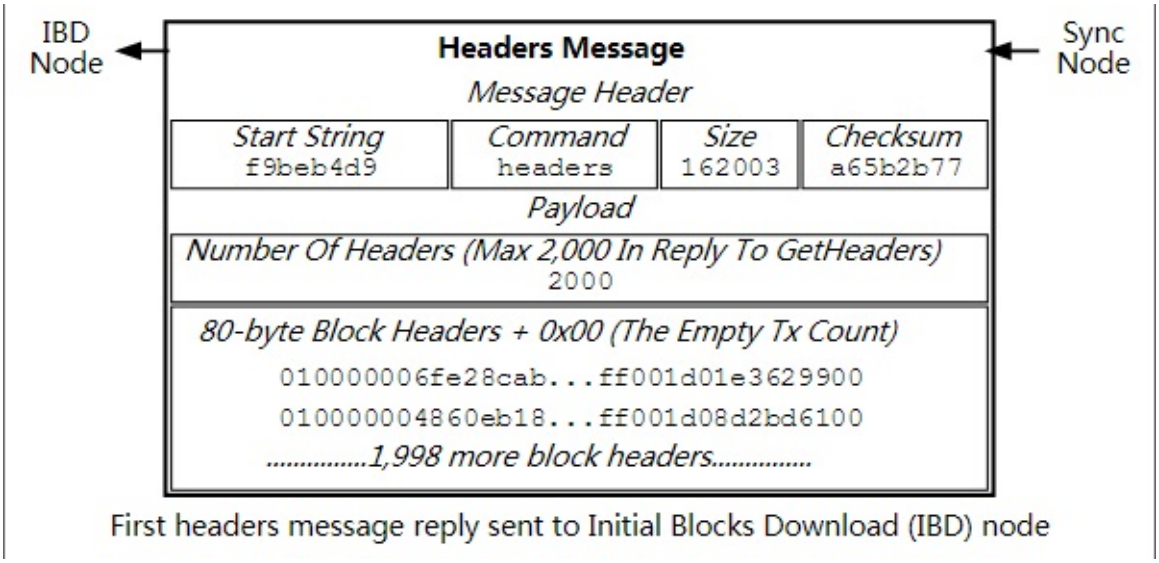
当节点第一次启动时，它的本地区块链只包含一个节点——硬编码的创始区块（区块0）。这个节点选择一个远端伙伴，成为同步节点，然后向它发送一个如下所示的 `getheaders` 消息。



First headers message reply sent to Initial Blocks Download (IBD) node

在 `getheaders` 信息的header hashs字段，新节点发送了它仅有的创世区块的头部hash值（6f32...0000，内部字节序）。此外它还把结束hash字段全部置零，以请求最大的响应。

当收到 `getheaders` 消息后，同步节点拿到头部hash在它的区块链中搜索那个hash值对应的区块。它发现区块0匹配，因此回复从区块1开始的2000个头部（回复的最大值）。它把这些头部hash放在如下所示的 `headers` 消息中。

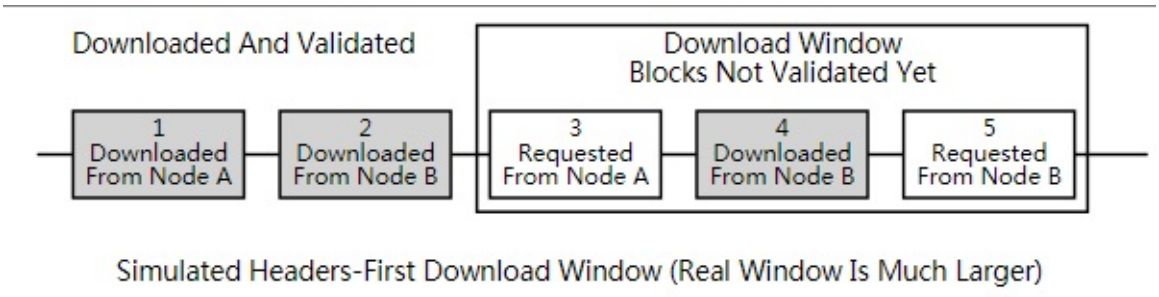


First headers message reply sent to Initial Blocks Download (IBD) node

IBD节点暂时只能部分校验区块头，包括确保所有的字段符合一致性规则，以及根据nBits属性头部hash在目标门限以下。（完整校验仍然需要从对应区块中找回所有交易）

当IBD节点部分验证区块头部以后，它可以并行进行以下两项工作：

1. 下载更多的头部。IBD节点可以发送另一个 `getheaders` 消息到同步节点，请求最优块头链上下一组2000区块头部。这些块头可以立即被校验，然后再发出下一批请求如此重复，直到从同步节点收到的头部个数小于2000，意味着同步节点已经不能提供更多的头部。这段介绍写下的时间，块头同步可以在200次循环过程内完成，大约下载32MB的数据。 一旦IBD节点从同步节点收到一个少于2000个块头的 `headers` 消息，它向周围的每个outbound 伙伴发送一个 `getheaders` 消息来获取他们的最优头部链信息。通过对比每个outbound伙伴的响应，它很容易就可以判定自己下载的块头是不是属于最优块头链。这就意味着即使不适用检查点，不诚实的同步节点也会很快被发现（IBD节点至少要连接一个诚实节点；比特币核心仍会继续提供检查点，以防一个诚实节点都找不到）。
2. 下载区块。当IBD节点不断下载块头以及块头下载完成时，它还在逐个请求并下载区块。IBD节点可以使用从头部链中计算得到的toubuhash创建一个 `getdata` 消息，通过他们的列表请求区块。它并一定非要从同步节点下载区块，任何一个全节点伙伴都可以。（虽然并不是所有的全节点都保存了全部区块）这样可以并行的获取区块，而且避免了下载速度受单个同步节点上行速度限制的问题。 为了使用多个伙伴分担负载，比特币核心每次向每个伙伴最多可以请求16个区块。考虑到一个节点最多可以有8个outboud连接，块头优先的比特币核心在IBD期间可以最多可以同时请求128个区块（和区块优先比特币核心允许一次向同步节点请求的最大值相同）



Simulated Headers-First Download Window (Real Window Is Much Larger)

比特币核心块头优先模式使用一个1024区块的滑动下载窗口最大化下载速度。窗口的当前最小高度区块是下一个将被校验的，如果比特币核心准备好校验它时它仍为到达，比特币核心将会等待最少2s的时间防止发送延迟。如果区块仍未到达，比特币核心将断开和拖延节点的连接并尝试连接另外的节点。例如，如上所示，如果在2s内仍然没有发送区块3，那么和它的连接就会被断开。 一旦IBD节点同步到区块链当前最新位置，它就开始接收下一节介绍的正常的广播区块。

资源：下面的表格总结了本节提到的信息。信息的连接可以带你查看对应信息的参考页面。

Message	getheaders	headers	getdata	block
From > To	IBD -> Sync	Sync -> IBD	IBD -> Sync	Sync -> IBD
内容	一个或多个头部hash	最多2000个区块头部	一个或多个区块列表	一个顺序的区块

区块广播

当一个矿工挖到一个新的区块，它将使用如下方式的一种将该区块广播到自己的伙伴节点：

- 自主区块推送：矿工把新区块放到一个 `block` 消息中发送给自己的全节点同伴。通过这种方式矿工可以合理的规避标准的同步方法，他知道自己所有的同伴都没有这个新发现的区块。
- 标准区块中继：矿工标准的中继节点一样，向他所有的伙伴节点（包括全节点和SPV节点）发送一个包含新区块列表信息的 `inv` 消息。通常回应有如下集中：
    - 每个块头优先同伴通过 `getheaders` 消息求区块，请求当中包含它的最优块头连中高度最高的块头的hash值，同样地请求当中会包含最优块头链中之前的区块头以便进行分叉监测。跟随这个消息后还会携带一个 `getdata` 消息请求完整区块。通过先请求区块，一个块头优先节点可以拒绝下面章节介绍的孤儿区块。
    - 每个SPV客户端通过发送一个 `getdata` 消息，通常请求merkle区块。

      矿工根据请求回复给每个节点，有的回复一个包含区块的 `block` 消息，或者一个或多个 `headers` 消息，或者merkle区块以及和包含SPV客户端布隆滤波器相关交易的 `merkleblock` 消息（消息后含0个或多个 `tx` 消息）。

默认情况下，比特币核心使用标准的区块中继方式广播区块，但是它也接受使用上面介绍的任何一种方式的广播的区块。

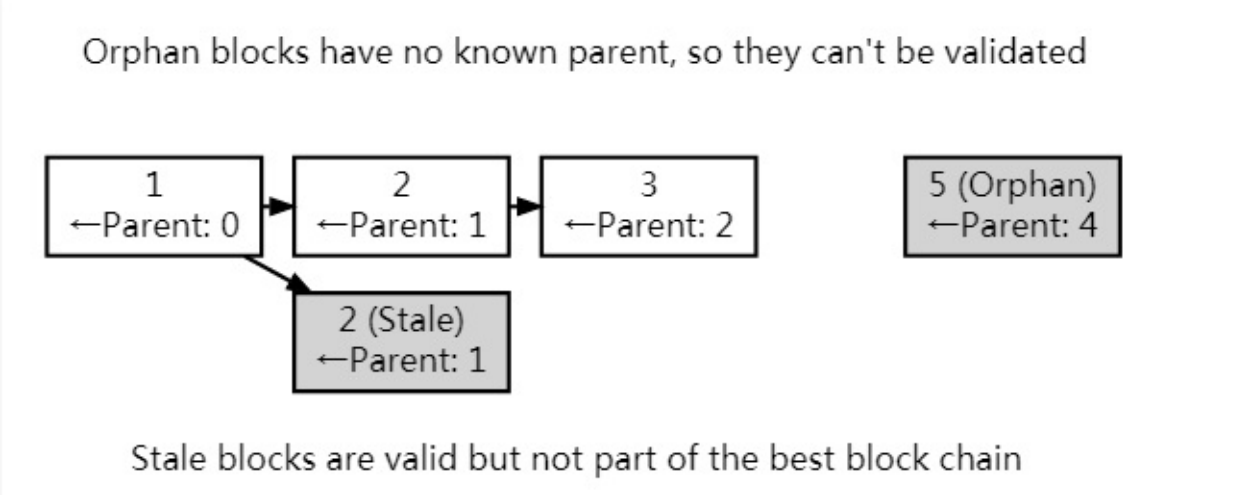
全节点校验接收到的区块，然后建议它的伙伴节点使用上面讨论的标准区块中继方法。下面浓缩的表格当中高亮了上面讨论的操作使用的消息（Relay, BF, HF 和SPV 代指中继节点、区块优先节点、块头优先节点和SPV客户端，any代指使用任意区块请求方法的节点）。

Message	inv	getdata	getheaders	headers
From -> To	Realay->Any	BF->Relay	HF->Relay	Relay->HF
负载	新区块列表	新区块列表	HF节点最优头部链一个或多个头部hash	中继链上与HF节点最优头部链相关的最多2000个头部

Message	block	merkleblock	tx
From -> To	Relay->BF/HF	Relay->SPV	Relay->SPV
负载	顺序的新区块	merkle区块过滤的新区块	新区块中满足布隆滤波器的顺序交易

孤儿区块

区块优先的节点可能会下载孤儿区块（把头部hash字段在当前区块头部的前一个节点尚未到达，本区块称为孤儿区块）。换句话说，孤儿区块没有已知的父区块（不同于废弃区块，废弃区块父区块已知，但是不属于最优区块链）。



块头优先节下载点下载到一个孤儿区块后并不对它进行校验。而是向发送孤儿区块的广播节点发送一个 `getblock` 消息，该广播节点会回复一个包含下载节点丢失区块列表（最多500条）的 `inv` 消息，下载节点将会使用一个 `getdata` 消息请求这些区块，广播节点将会通过 `block` 消息发送这些区块。下载节点将会校验这些区块，一旦前面孤儿区块的父区块被校验，它就会校验该孤儿区块。

块头优先节点通常先使用 `getheaders` 消息请求块头然后再用 `getdata` 消息请求区块的方式来避免这个复杂的过程。广播节点将会将会下发一个包含它认为下载节点更新到最新需要的所有（最多2000）区块块头的 `headers` 消息，这些头部将会指向他们的父节点，因此下载节点

收到 `blocks` 消息中包含的区块肯定不是父区块，因为它所有的符节点都是已知的（即使尚未校验）。除此之外，如果收到的 `block` 消息中包含了一个孤儿区块，块头优先节点将会立刻忽略该区块。

但是忽略孤儿区块意味着头部优先节点将会忽略矿工通过自助区块推送发送的孤儿区块。

交易广播

为了把交易发送给同伴，需要使用 `inv` 消息。如果收到一个 `getdata` 消息的回复，就使用 `tx` 消息发送交易。同伴接收到交易后，如果交易有效，也使用同样的方式转发。

内存池

全信息节点会跟踪有资格被包含到下一个区块的未证实的交易。这对于实际挖掘这些区块的矿工来说更是必要的，但是对于想要追踪那些未确认交易的伙伴节点来说也是有用的，比如为SPV节点提供为确认交易信息的节点。

由于为确认的交易在比特币中没有永久的状态，比特币核心把他们存储在非持久内存中，称为内存池(memory pool)或存储池(mempool)。当一个节点关闭后，除了存储在它钱包中的交易外，存储池中的内容就会丢失。这意味着那些从没有被矿工确认的交易，随着节点重启或内存淘汰慢慢就会消失在网络中。

那些被挖到废弃区块中的交易可能被重新加入内存池。这些重新添加的交易也可能由于被包含在新的替换区块（废弃区块重组后添加到当前最优链）中而被立即删除。比特币核心现在的情况时，从废弃链顶端开始，一个一个移除废弃链上的区块。随着每个区块被删除，其中的交易会被添加回内存池。等到所有的废弃区块都被溢出，更新的区块会一个一个被添加到区块上，知道最先的区块。随着区块的添加，它包含的交易会被从内存池中删除。

SPV节点没有内存池，因此他们不中继交易。他们不能独立的校验一个没有被包含在区块中的交易是否被双花，因此他们不知道哪个区块有资格被加入到下一个区块。

异常节点

两种类型的广播都要注意，对于那些通过发送错误信息占用带宽和计算资源的异常节点室友惩罚机制的。如果一个节点的惩罚因子大于 `-banscore=<n>` 门限值，就会被禁止 `-bantime=<n>` 指定的一段时间（秒为单位，默认是86400，即24小时）。

警告

从比特币核心v0.13.0版本已经移除。

早期版本的比特币核心孕育开发者和受信任的团体成员提交比特币警告事件通知用户严重的大范围网络问题。这个消息系统从比特币核心v0.13.0就退休了。但是内部的警告、部分检查警告和 `-alertnotify` 选项依然保留。

对应英文文档

挖矿

暂未翻译