P2P Network

这一部分讲解了比特币 P2P 网络协议（但这并不是一个说明书）。它没有说明非连续直接的 IP to IP 支付协议、BIP70 支付协议、GetBlockTemplate mining 协议或者任何从未在 Bitcoin Core 官方版本中实现的网络协议。

所有点对点通信都完全发生在 TCP 之上。

注意：除非说明中另有说明，否则本节中提到的所有多字节整数均按小端顺序传输。

常量及默认值：

以下常量和默认值取自 Bitcoin Core chainparams.cpp 源码文件。

Network	Default Port	Start String	Max nBits
Mainnet	8333	0xf9beb4d9	0x1d00ffff
Testnet	18333	0x0b110907	0x1d00ffff
Regtest	18444	0xfabfb5da	0x207fffff

注意：上面的 testnet start string 和 nBits 是对于 testnet3 的；原始的 testnet 使用了一个不同的字符串和更高的（不那么难的）nBits。

命令行参数可以更改节点侦听的端口（请参阅-help）。Start string 是在所有消息在比特币网络上开始发送时出现的硬编码常量；他们也可能出现在像 Bitcoin Core 区块数据库的数据文件中。上述的 nBits 是按大端顺序的；他们以小端的顺序在网络上发送。

Bitcoin Core 的 chainparams.cpp 还包括对程序有用的其他常量，例如不同网络的创世区块的哈希。

协议版本

下表列出了一些值得注意的 P2P 网络协议版本，其中最新版本列在第一位。

截至 Bitcoin Core 0.14.2，最新的协议版本是 70015。

版本	初始发布	重要更改

版本	初始发布	重要更改
70015	Bitcoin Core 0.13.2 (Jan 2017)	• New banning behavior for invalid compact blocks #9026in v0.14.0, Backported to v0.13.2 in #9048.
70014	Bitcoin Core 0.13.0 (Aug 2016)	BIP152: • Added `sendcmpct`, `cmpctblock`, `getblocktxn`, `blocktxn` messages • Added `MSG_CMPCT_BLOCK` inventory type to `getdata` message.
70013	Bitcoin Core 0.13.0 (Aug 2016)	BIP133: • Added `feefilter` message. • Removed `alert` message system. See Alert System Retirement
70012	Bitcoin Core 0.12.0 (Feb	BIP130: • Added `sendheaders` message.

版本	初始发布	重要更改
	2016)	
70011	Bitcoin Core 0.12.0 (Feb 2016)	BIP111: • `filter*` messages are disabled without NODE_BLOOM after and including this version.
70002	Bitcoin Core 0.9.0 (Mar 2014)	• Send multiple `inv` messages in response to a `mempool` message if necessary BIP61: • Added `reject` message
70001	Bitcoin Core 0.8.0 (Feb 2013)	• Added `notfound` message. BIP37: • Added `filterload` message. • Added `filteradd` message. • Added `filterclear` message. • Added `merkleblock` message. • Added relay field to `version` message • Added `MSG_FILTERED_BLOCK` inventory type to `getdata` message.
60002	Bitcoin	BIP35: • Added `mempool` message.

版本	初始发布	重要更改
	Core 0.7.0 (Sep 2012)	• Extended `getdata` message to allow download of memory pool transactions
60001	Bitcoin Core 0.6.1 (May 2012)	BIP31: • Added nonce field to `ping` message • Added `pong` message
60000	Bitcoin Core 0.6.0 (Mar 2012)	BIP14: • Separated protocol version from Bitcoin Core version
31800	Bitcoin Core 0.3.18 (Dec 2010)	• Added `getheaders` message and `headers` message.
31402	Bitcoin Core	• Added time field to `addr` message.

版本	初始发布	重要更改
	0.3.15 (Oct 2010)	
311	Bitcoin Core 0.3.11 (Aug 2010)	• Added `alert` message.
209	Bitcoin Core 0.2.9 (May 2010)	• Added checksum field to message headers, added `verack` message, and added starting height field to `version` message.
106	Bitcoin Core 0.1.6 (Oct 2009)	• Added transmitter IP address fields, nonce, and User Agent (subVer) to `version` message.

Message Headers

网络协议中的所有消息使用相同的容器格式，它提供了一个所需的多字段消息头和一个可选的有效载荷。消息头格式是：

字节数	名称	数据类型	秒速
4	start string	char[4]	Magic bytes indicating the originating network; used to seek to next message when stream state is unknown.
12	command name	char[12]	ASCII string which identifies what message type is contained in the payload. Followed by nulls (0x00) to pad out byte count; for example: `version\0\0\0\0\0`.
4	payload size	uint32_t	Number of bytes in payload. The current maximum number of bytes (`MAX_SIZE`) allowed in the payload by Bitcoin Core is 32 MiB—messages with a payload size larger than this will be dropped or rejected.
4	checksum	char[4]	*Added in protocol version 209.* First 4 bytes of SHA256(SHA256(payload)) in internal byte order. If payload is empty, as in `verack` and `getaddr` messages, the checksum is always 0x5df6e0e2 (SHA256(SHA256(<empty string>))).

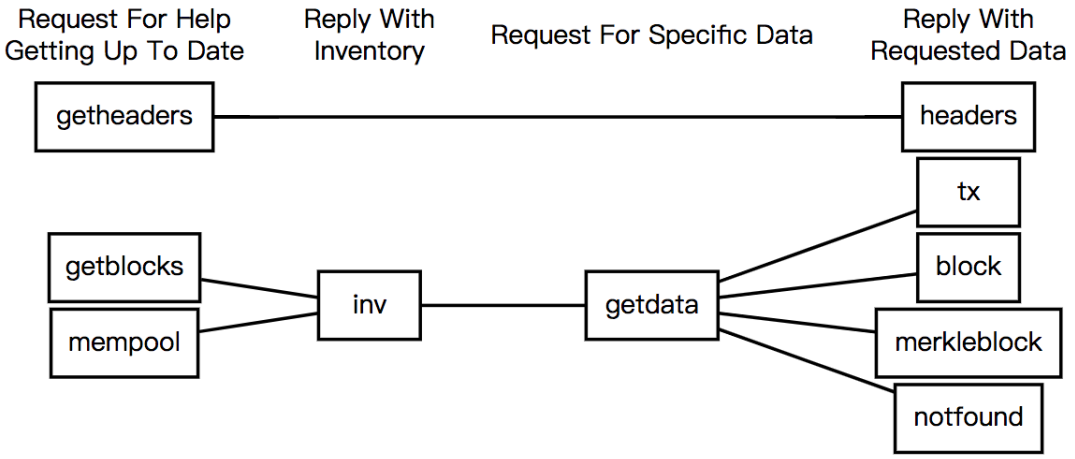
以下示例是来自没有负载的 verack 消息的 mainnet 消息头的注释十六进制输出。

```
f9beb4d9 .................. Start string: Mainnet
76657261636b000000000000 ... Command name: verack + null padding
00000000 .................. Byte count: 0
```

```
5df6e0e2 ................... Checksum: SHA256(SHA256(<empty>))
```

数据消息

以下网络消息全部请求或提供与交易和区块有关的数据。



Overview Of P2P Protocol Data Request And Reply Messages

许多数据消息使用 inventories 作为交易和块的唯一标识符。Inventories 有一个简单的 36 字节结构：

字节数	名称	数据类型	描述
4	type identifier	uint32_t	The type of object which was hashed. See list of type identifiers below.
32	hash	char[32]	SHA256(SHA256()) hash of the object in internal byte order.

当前可用类型标识符有：

Type Identifier	Name	Description
1	`MSG_TX`	The hash is a TXID.

Type Identifier	Name	Description
2	MSG_BLOCK	The hash is of a block header.
3	MSG_FILTERED_BLOCK	The hash is of a block header; identical to MSG_BLOCK. When used in a getdata message, this indicates the response should be a merkleblock message rather than a block message (but this only works if a bloom filter was previously configured).**Only for use in getdata messages.**

类型标识符零和大于 3 的被保留并将用于以后的实现。Bitcoin Core 忽略了所有这些未知类型的 inventories。

区块

block 消息以在序列化区块部分中描述的格式传输单个序列化的区块。到该章节获取十六进制输出的例子。它可以以两个不同的原因发送：

1. GetData 响应：节点将始终发送它以响应一个请求 MSG_BLOCK 类型 inventory 的区块的 getdata 消息，(假设该节点使得该区块可用)。
2. Unsolicited：一些矿工将发送未经请求的广播他们新挖出的区块的 block 信息给他们的所有对等点。许多采矿池做同样的事情，虽然有些可能被错误地配置以从多个节点发送区块，可能不止一次地将同一区块发送给某些对等点。

GetBlock

getblocks 消息请求一个 inv 消息，该消息提供从区块链中的特定点开始的区块头哈希。它允许一个第一次被断开或启动的对等点获得它所需要的数据来请求它没有看到的区块。

已经断开连接的对等点可能在其本地存储的区块链中具有旧的区块，所以 getblocks 消息允许请求方在其本地链上的不同高度处向接收方提供多个头散列。

这允许接收方在该列表内查找它们共同拥有的最后一个头哈希 ,并回复所有后续头哈希。

注意：接收方本身可能会响应一个包含旧区块头哈希的 inv 消息。轮询所有的对等点以找到最好的区块链是由请求方来决定的。

如果接收方没有在列表中找到一个共同的头哈希 , 它将假定最后一个公共区块是创世区块（区块 0）, 所以它将在包含从区块 1 开始的头部哈希的 inv 消息中进行回复（第一个区块在创世区块之后)。

字节数	名称	数据类型	描述
4	version	uint32_t	The protocol version number; the same as sent in the `version` message.
Varies	hash count	compactSize uint	The number of header hashes provided not including the stop hash. There is no limit except that the byte size of the entire message must be below the `MAX_SIZE` limit; typically from 1 to 200 hashes are sent.
Varies	block headerhashes	char[32]	One or more block header hashes (32 bytes each) in internal byte order. Hashes should be provided in reverse order of block height, so highest-height hashes are listed first and lowest-height hashes are listed last.
32	stop hash	char[32]	The header hash of the last header hash being requested; set to all zeroes to request an `inv` message with all subsequent headerhashes (a maximum of 500 will be sent as a reply to this message; if you need more than 500, you

字节数	名称	数据类型	描述
			will need to send another `getblocks` message with a higher-heightheader hash as the first entry in block headerhash field).

下述十六进制输出注释展示了一个 getblocks 消息。(消息头被省略。)

```
71110100 ............................ Protocol version: 70001
02 ................................. Hash count: 2

d39f608a7775b537729884d4e6633bb2
105e55a16a14d31b0000000000000000 ... Hash #1

5c3e6403d40837110a2e8afb602b1c01
714bda7ce23bea0a0000000000000000 ... Hash #2

0000000000000000000000000000000000
0000000000000000000000000000000000 ... Stop hash
```

GetData

getdata 消息请求来自另一个节点的一个或多个数据对象。这些对象是由一个 inventory 所请求的，请求节点通常事先通过一个 inv 消息来接收这个清单。

对 getdata 消息的响应可以是一个 tx 消息、block 消息、merkleblock 消息或者 notfound 消息。

此消息不能用于请求任意数据，例如历史交易不再存在于内存池或中继集合中。如果完整的节点已经在区块数据库中修剪旧的交易，那么它们甚至可能无法提供旧区块。出于这个原因，getdata 消息通常应该只用于通过发送一个 inv 消息来向先前通告它的那个节点请求数据。

getdata 消息的格式和最大限制与 inv 消息相同；只有消息头不同。

GetHeaders

在协议版本 31800 中添加。

getheaders 消息请求一个 headers 消息，该消息提供从区块链中的特定点开始的区块头。它允许一个第一次断开或启动的对等点获得它还没有看到的头。

getheaders 消息几乎与 getblocks 消息相同，只有一点区别：对 getblocks 消息的 inv 回复将包含不超过 500 个区块头哈希；回复 getheaders 消息的 headers 将包括多达 2,000 个区块头。

Headers

在协议版本 31800 中添加。

headers 消息将区块头发送到先前用 getheaders 消息请求特定头的节点。一个 headers 消息可以为空。

字节数	名称	数据类型	描述
Varies	count	compactSize uint	Number of block headers up to a maximum of 2,000. Note: headers-first sync assumes the sending node will send the maximum number of headers whenever possible.
Varies	headers	block_header	Block headers: each 80-byte block headeris in the format described in the block headers section with an additional 0x00 suffixed. This 0x00 is called the transaction count, but because the headers message doesn't include any transactions, the transaction count is always zero.

下述十六进制输出注释展示了一个 headers 消息。（消息头已被忽略。）

```
01 ................................ Header count: 1
```

```
02000000 ............................ Block version: 2

b6ff0b1b1680a2862a30ca44d346d9e8

910d334beb48ca0c0000000000000000 ... Hash of previous block's header

9d10aa52ee949386ca9385695f04ede2

70dda20810decd12bc9b048aaab31471 ... Merkle root

24d95a54 ............................ Unix time: 1415239972

30c31b18 ............................ Target (bits)

fe9f0864 ............................ Nonce


00 ................................. Transaction count (0x00)
```

Inv

inv 消息（inventory 消息）传输一个或多个发送方已知对象的 inventory。它可以被未经请求的主动发送来宣布新的交易或区块，或者它可以被发送以回复 getblocks 消息或 mempool 消息。

接收方可以将一个 inv 消息中的 inventories 与已经看到的 inventories 进行比较，然后使用后续消息来请求未见的对象。

字节数	名称	数据类型	描述
Varies	count	compactSize uint	The number of inventory entries.
Varies	inventory	inventory	One or more inventory entries up to a maximum of 50,000 entries.

下述注释十六进制输出展示了一个有两个 incentory 条目的 inv 消息。（消息头已被忽略。）

```
02 ................................. Count: 2
```

```
01000000 ............................ Type: MSG_TX
de55ffd709ac1f5dc509a0925d0b1fc4
42ca034f224732e429081da1b621f55a ... Hash (TXID)

01000000 ............................ Type: MSG_TX
91d36d997037e08018262978766f24b8
a055aaf1d872e94ae85e9817b2c68dc7 ... Hash (TXID)
```

Mempool

在协议版本 60002 中添加。

mempool 消息请求接收节点验证为有效但尚未出现在区块中的交易的 TXID。也就是在接收节点的内存池中的交易。mempool 消息的响应是一个或多个 inv 消息，其中包含通常 inventory 格式的 TXID。

当程序第一次连接到网络时，发送 mempool 消息是非常有用的。完整的节点可以使用它来快速收集网络上可用的大部分或全部未确认的交易 ;这对于试图为他们的交易费用而收集交易的矿工尤其有用。SPV 客户端可以在发送 mempool 之前设置过滤器，从而只接收匹配该过滤器的交易；这允许一个最近启动的客户端获得与其钱包有关的大部分或全部未确认的交易。

mempool 消息的 inv 响应最多只是一个节点的网络视图 ,而不是网络上未确认交易的完整列表。以下是该列表可能不完整的一些其他原因：

- 在 Bitcoin Core 0.9.0 之前，对 mempool 消息的响应只是一个 inv 消息。一个 inv 消息被限制为 50,000 个 inventory，因此一个具有大于 50,000 个条目的内存池的节点不会发送所有内容。Bitcoin Core 的更新版本根据需要发送尽可能多的 inv 消息来引用其完整的内存池。
- mempool 消息当前不与 filterload 消息的 BLOOM_UPDATE_ALL 和 BLOOM_UPDATE_P2PUBKEY_ONLY 标志完全兼容。Mempool 交易不像区块内交易那样排序，所以一个花费一个支出的交易（tx2）可以出现在包含该支出的交易（tx1）之前，这意味着自动的过滤器更新机制将直到第二次出现的事务(tx1)被看见之后才会操作—缺失了先出现的交易(tx2)。在 Bitcoin Core 问题＃2381 中已经提出，交易在被过滤器处理之前就应该被排序。

在 mempool 消息中没有有效载荷。有关不含负载的消息的示例，请参阅消息头部分。

MerkleBlock

如 BIP37 所述，在协议版本 70001 中添加。

merkleblock 消息是对使用 inventory 类型 MSG_MERKLEBLOCK 来请求一个区块的 getdata 消息的回复。这只是回复的一部分 :如果找到任何匹配的交易，它们将作为 tx 消息单独发送。

如果一个过滤器先前已经用 filterload 消息设置，那么 merkleblock 消息将包含所请求块中与过滤器相匹配的任何交易的 TXID，以及有必要将那些交易连接到区块头到 merkle 根到区块 merkle 树的任何部分。该消息还包含区块头的一份完整拷贝，以允许客户端对其进行散列并确认其 proof of work。

字节数	名称	数据类型	描述
80	block header	block_header	The block header in the format described in the block header section.
4	transaction count	uint32_t	The number of transactions in the block(including ones that don't match the filter).
Varies	hash count	compactSize uint	The number of hashes in the following field.
Varies	hashes	char[32]	One or more hashes of both transactions and merkle nodes in internal byte order. Each hash is 32 bytes.
Varies	flag byte count	compactSize uint	The number of flag bytes in the following field.
Varies	flags	byte[]	A sequence of bits packed eight in a byte

字节数	名称	数据类型	描述
			with the least significant bit first. May be padded to the nearest byte boundary but must not contain any more bits than that. Used to assign the hashes to particular nodes in the merkle tree as described below.

下述注释的十六进制输出展示了一个与下列例子对应的 merkleblock 消息 。（消息头已被忽略。）

```
01000000 ........................... Block version: 1
82bb869cf3a793432a66e826e05a6fc3
7469f8efb7421dc88067010000000000 ... Hash of previous block's header
7f16c5962e8bd963659c793ce370d95f
093bc7e367117b3c30c1f8fdd0d97287 ... Merkle root
76381b4d ........................... Time: 1293629558
4c86041b ........................... nBits: 0x04864c * 256**(0x1b-3)
554b8529 ........................... Nonce

07000000 ........................... Transaction count: 7
04 ................................. Hash count: 4

3612262624047ee87660be1a707519a4
43b1c1ce3d248cbfc6c15870f6c5daa2 ... Hash #1
019f5b01d4195ecbc9398fbf3c3b1fa9
bb3183301d7a1fb3bd174fcfa40a2b65 ... Hash #2
41ed70551dd7e841883ab8f0b16bf041
76b7d1480e4f0af9f3d4c3595768d068 ... Hash #3
20d2a7bc994987302e5b1ac80fc425fe
```

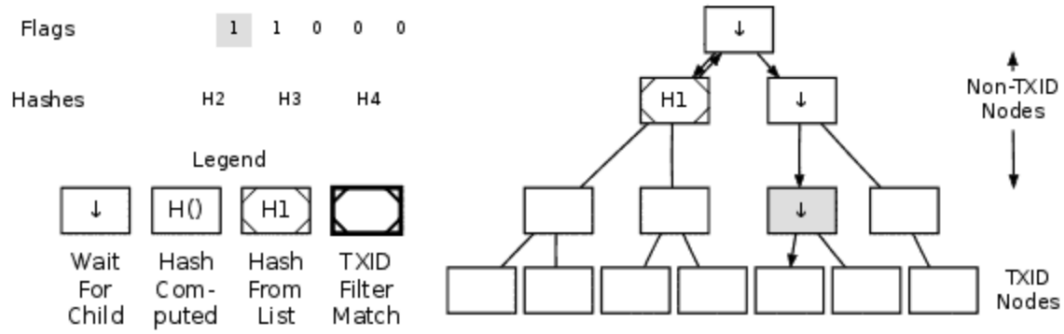
```
25f8b63169ea78e68fbaaefa59379bbf ... Hash #4


01 ................................ Flag bytes: 1

1d ................................ Flags: 1 0 1 1 1 0 0 0
```

注意：当完全解码时，上面的 merkleblock 消息提供了匹配过滤器的单个交易的 TXID。在该输出被取出的网络流量输出中，属于该 TXID 的完整交易在 merkleblock 消息之后立即被作为 tx 消息发送出去。

解析 MerkleBlock 消息

如上面注释的十六进制输出所示，merkleblock 消息提供了三种特殊的数据类型：一个交易计数、一个散列表以及一个单比特标志列表。

你可以使用交易计数来构造一个空的 merkle 树。我们将树中的每个条目称为一个节点；底部是 TXID 节点—这些节点的散列是 TXIDs；其余节点（包括 merkle 根）是非 TXID 节点—它们实际上可能与一个 TXID 具有相同的散列，但我们要将它们区别对待。



gif 地址：https://bitcoin.org/img/dev/animated-en-merkleblock-parsing.gif

按照它们在 merkleblock 消息中出现的顺序保持散列和标志（flag）。当我们说“下一个标志”或“下一个散列”时，我们指的是列表中的下一个标志或散列，即使这是我们目前使用的第一个。

从 merkle 根节点和第一个标志开始。下表描述了如何基于正在处理的节点是 TXID 节点还是非 TXID 节点来评估标志。一旦将一个标志应用到一个节点，不要将另一个标志应用到同一个节点，或再次使用同一个标志。

Flag	TXID Node	Non-TXID Node
0	Use the next hash as this node'sTXID, but this transaction didn't match the filter.	Use the next hash as this node's hash. Don't process any descendant nodes.
1	Use the next hash as this node'sTXID, and mark this transaction as matching the filter.	The hash needs to be computed. Process the left child node to get its hash; process the right child node to get its hash; then concatenate the two hashes as 64 raw bytes and hash them to get this node's hash.

任何时候你第一次开始处理节点，评估下一个标志。在任何其他时候都不要使用标志。

处理子节点时，可能需要处理其子节点（原始节点的孙辈节点）或下一级节点，然后再返回到父节点。这是期望的—首先保持处理的深度，直到你到达一个 TXID 节点或一个标志为 0 的非 TXID 节点。

当你处理一个 TXID 节点或一个标志为 0 的非 TXID 节点后，停止处理标志并开始上升树。当你上升的时候，为你现有的两个子散列或你现有的唯一子散列计算任何节点的散列。请参阅 merkle 树章节部分以获取哈希指导。如果你到达一个只有左边的散列已知的节点，根据需要下降到它的右边的子节点（如果有的话）和以后的后代节点。

但是，如果您发现一个节点的左侧和右侧子节点都具有相同的哈希，则失败。这与 CVE-2012-2459 有关。

继续下降和上升，直到获得足够的信息来获取 merkle 根节点的散列。如果在达到该条件之前用光了标志或哈希，则失败。然后执行以下检查（顺序无关紧要）：

- 如果在哈希列表中有未使用的哈希，则失败。
- 如果有未使用的标志位则会失败—除了填充到下一个完整字节所需的最小位数外。
- 如果 Merkle 根节点的散列与区块头中的 merkle 根不相同，则失败。

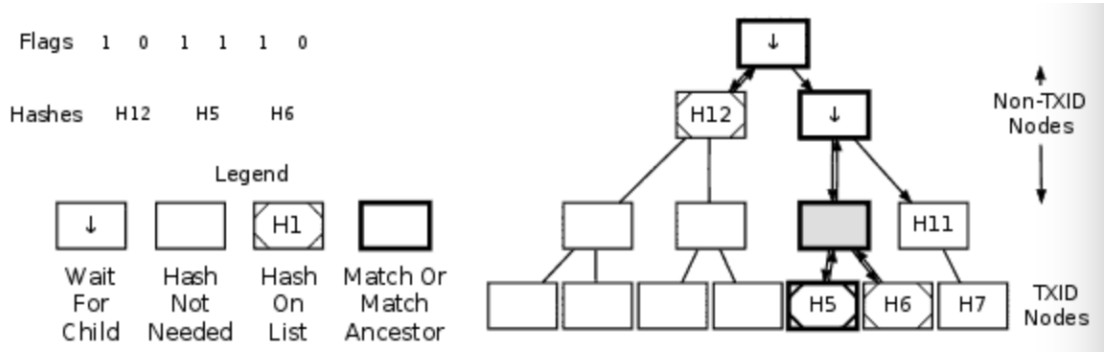
· 区块头无效时失败。请记住确保头散列小于或等于由 nBits 头字段编码的目标阈值。你的程序当然也应该试图确保头属于最好的区块链，并且用户知道这个区块有多少个确认。

有关解析 merkleblock 消息的详细示例，请参阅相应的 merkle 区块示例部分。

创建 MerkleBlock 消息

在理解如何解析已经创建的消息之后，理解如何创建 merkleblock 消息会更容易，所以我们建议您首先阅读上面的解析部分。

在底行创建一个带有 TXID 和在顶行计算到 Merkle 根的所有其他散列的完整的 Merkle 树。对于匹配过滤器的每个交易，跟踪其 TXID 节点及其所有祖先节点。



gif 地址：https://bitcoin.org/img/dev/animated-en-merkleblock-creation.gif

开始使用 merkle 根节点处理树。下表描述了如何根据节点是否为匹配节点，匹配的祖先节点，或者既不是匹配节点也不是匹配祖先节点，来处理 TXID 节点和非 TXID 节点。

	TXID Node	Non-TXID Node
Neither Match Nor Match Ancestor	Append a 0 to the flag list; append this node's TXIDto the hash list.	Append a 0 to the flag list; append this node'shash to the hash list. Do not descend into its child nodes.
Match Or Match	Append a 1 to the flag list; append	Append a 1 to the flag list; process the left child node. Then, if the node has a

	TXID Node	Non-TXID Node
Ancestor	this node's TXIDto the hash list.	right child, process the right child. Do not append a hash to the hash list for this node.

任何时候你第一次开始处理一个节点，一个标志应该被附加到标志列表中。除非处理完成，将标记列表填充到字节边界，否则绝不要在其他时间在列表上放置标记。

处理子节点时，可能需要处理其子节点（原始节点的孙辈节点）或下一级节点，然后再返回到父节点。这是期望的—首先保持处理深度，直到你到达一个 TXID 节点或一个既不是 TXID 也不是匹配祖先的节点。

在处理完一个 TXID 节点或一个既不是 TXID 也不是匹配祖先的节点之后，停止处理并开始上升树，直到找到一个尚未处理的右子节点。下降到这个右子节点，并处理它。

在按照上表中的说明完整处理 merkle 根节点后，处理即完成。将标志列表填充到字节边界，并使用本小节开头附近的模板构造 merkleblock 消息。

NotFound

在协议版本 70001 中添加。

notfound 消息是对 getdata 消息的回复，该消息请求接收节点不具有可用于中继的对象。（不希望节点中继历史交易，这些交易不再存在于内存池或中继集合中。节点也可能从较早的区块中删除已用完的交易，使它们无法发送这些区块）。

notfound 消息的格式和最大限制与 inv 消息相同；只是消息头不同。

Tx

tx 消息以原始交易处理格式传输单个交易。它可以在各种情况下发送;

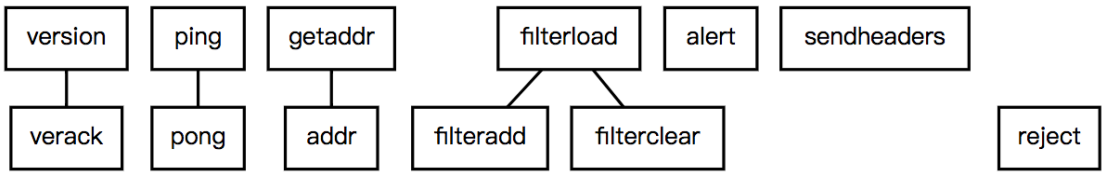
- 交易回应 Bitcoin Core 和 BitcoinJ 将发送它以回复一个请求交易的 inventory 类型为 MSG_TX 的 getdata 消息。
- MerkleBlock 响应：Bitcoin Core 将发送它以回复一个请求 merkle 区块的 inventory 类型为 MSG_MERKLEBLOCK 的 getdata 消息。（这是发送一个 merkleblock 消息的补充。）在这种情况下，每个 tx 消息提供来自该区块的匹配交易。

· Unsolicited：BitcoinJ 会发送一个 tx 消息来主动发起它的交易。

对于原始交易（raw transaction）格式的示例十六进制输出，请参阅原始交易部分。

Control Message

以下网络消息都有助于控制两个对等点之间的连接，或者允许他们相互通知关于网络其余部分的信息。



Overview Of P2P Protocol Control And Advisory Messages

请注意，几乎没有任何控制消息以任何方式进行身份验证，这意味着它们可能包含不正确的或蓄意有害的信息。另外，这部分还没有涵盖 Tor 网络上的 P2P 协议操作。

Addr

addr（IP 地址）消息中继网络上的对等点的连接信息。每个想要接受传入连接的对等点创建一个提供其连接信息的 addr 消息，然后将该消息发送给其对等点的 unsolicited。其中一些对等点将这些信息发送给他们的对等点（也是 unsolicited），其中一些将其进一步分发，允许为已经在网络上的任何程序分散对等点的发现。

addr 消息也可以被发送以响应 getaddr 消息。

字节数	名称	数据类型	描述
Varies	IP address count	compactSize uint	The number of IP address entries up to a maximum of 1,000.
Varies	IP addresses	network IP address	IP address entries. See the table below for the format of a Bitcoin network IP

字节数	名称	数据类型	描述
			address.

每个封装的网络 IP 地址当前使用如下结构：

字节数	名称	数据类型	描述
4	time	uint32	*Added in protocol version 31402.* A time in Unix epoch time format. Nodesadvertising their own IP address set this to the current time. Nodes advertising IP addresses they've connected to set this to the last time they connected to that node. Other nodes just relaying the IP address should not change the time.Nodes can use the time field to avoid relaying old `addr` messages. Malicious nodes may change times or even set them in the future.
8	services	uint64_t	The services the node advertised in its `version`message.
16	IP address	char	IPv6 address in **big endian byte order**. IPv4 addresses can be provided as IPv4-mapped IPv6 addresses
2	port	uint16_t	Port number in **big endian byte order**. Note that Bitcoin Core will only connect to nodes with non-standard port numbers as a last resort for

字节数	名称	数据类型	描述
			finding peers. This is to prevent anyone from trying to use the network to disrupt non-Bitcoin services that run on other ports.

以下带注释的十六进制输出显示了 addr 部分消息。（消息头已被省略，实际 IP 地址已被 RFC5737 保留的 IP 地址替代。）

```
fde803 ............................. Address count: 1000

d91f4854 ........................... Epoch time: 1414012889
0100000000000000 ................... Service bits: 01 (network node)
00000000000000000000ffffc0000233 ... IP Address: ::ffff:192.0.2.51
208d ............................... Port: 8333

[...] .............................. (999 more addresses omitted)
```

Alert

在协议版本 311 中添加。在协议版本 70013 中删除，并在 Bitcoin Core 0.13.0 中发布。

传统的 p2p 网络警报消息系统已经停用；但是，内部警报，分区检测警告和 -alertnotify 选项功能仍然存在。有关详细信息，请参阅 Alert System Retirement。

FeeFilter

如 BIP133 所述，在协议版本 70013 中添加。

feefilter 消息是对接收方的请求，不将任何交易 inv 消息中继给发送方，其中交易费率低于 feefilter 消息中指定的费率。

在 Bitcoin Core 0.12.0 引入 mempool 限制之后，Bitcoin Core 0.13.0 引入了 feefilter。Mempool 限制提供保护，防止攻击和费率低的垃圾交易，不太可能被纳入挖矿区块。feefilter 消息允许一个节点通知它的对等点，它将不会接受低于指定费率的交易进入其 mempool，因此对等点可以跳过对低于该费率的交易的中继 inv 消息，直接到该节点。

字节数	名称	数据类型	描述
8	feerate	uint64_t	The fee rate (in satoshis per kilobyte) below which transactions should not be relayed to this peer.

接收方可以选择忽略消息，不过滤交易 inv 消息。

费用过滤器是对 bloom 过滤器的补充。如果 SPV 客户端加载 bloom 过滤器并发送 feefilter 消息，则只有通过两个过滤器才能中继交易。

但请注意 ,feefilter 对区块传播没有影响或对 getdata 消息没有响应。例如，如果一个节点通过发送一个 inv 类型为 MSG_FILTERED_BLOCK 的 getdata 消息来向其对等节点请求一个 merkleblock，并且之前已经向该对等点发送了一个 feefilter，那么对等节点应该响应一个包含与 bloom 过滤器匹配的所有交易的 merkleblock，即使他们低于 feefilter 费率。

如果存在，则从 mempool 消息生成的 inv 消息将受到费用过滤器的限制。

下面带注释的十六进制输出显示了一个 feefilter 消息。(消息头已被省略。)

```
7cbd000000000000 ... satoshis per kilobyte: 48,508
```

FilterAdd

如 BIP37 所述，在协议版本 70001 中添加。

filteradd 消息告诉接收方将单个元素添加到先前设置的 bloom 过滤器(比如一个新的公共密钥) 中。元素直接发送给接收方；对等点然后使用在 filterload 消息中设置的参数来将该元素添加到 bloom 过滤器。

由于该元素直接发送到接收方 ,所以元素没有模糊，也没有由 bloom 过滤器提供的 plausible-deniablility 的隐私。希望保持更高隐私性的客户端应该自行重新计算 bloom 过滤器 ,并使用重新计算的 bloom 过滤器发送新的 filterload 消息。

字节数	名称	数据类型	描述
Varies	element bytes	compactSize uint	The number of bytes in the following element field.
Varies	element	uint8_t[]	The element to add to the current filter. Maximum of 520 bytes, which is the maximum size of an element which can be pushed onto the stack in a pubkey or signature script. Elements must be sent in the byte order they would use when appearing in a raw transaction; for example, hashes should be sent in internal byte order.

注意：除非先前使用 filterload 消息设置了过滤器，否则 filteradd 消息将不会被接受。

下面带注释的十六进制输出显示了一个添加 TXID 的 filteradd 消息。(消息头已被忽略。) 此 TXID 出现在 merkleblock 消息中用于示例十六进制输出的同一个区块中；如果 merkleblock 消息在发送此 filteradd 消息后重新发送，则返回六个哈希而不是四个。

```
20 ................................ Element bytes: 32
fdacf9b3eb077412e7a968d2e4f11b9a
9dee312d666187ed77ee7d26af16cb0b ... Element (A TXID)
```

FilterClear

如 BIP37 所述，在协议版本 70001 中添加。

filterclear 消息告诉接收方去除先前设置的 bloom 过滤器。这也消除了将 version 消息中的中继字段设置为 0 的影响，允许未经过滤的访问 inv 消息，宣布新的交易。

Bitcoin Core 在代替过滤器加载 filterload 之前不需要 filterclear 消息。在 filterclear 消息之前，它也不需要 filterload 消息。

filterclear 消息中没有有效载荷。有关负载的消息示例，请参阅消息头部分。

FilterLoad

如 BIP37 所述，在协议版本 70001 中添加。

filterload 消息告诉接收方过滤所有中继的交易，并通过提供的过滤器请求 merkle 区块。这允许客户端接收与他们的钱包相关的交易，以及可提供 plausible-deniablility 隐私的可配置假阳性交易率。

Bytes	Name	Data Type	Description
Varies	nFilterBytes	compactSize uint	Number of bytes in the following filter bit field.
Varies	filter	uint8_t[]	A bit field of arbitrary byte-aligned size. The maximum size is 36,000 bytes.
4	nHashFuncs	uint32_t	The number of hash functions to use in this filter. The maximum value allowed in this field is 50.
4	nTweak	uint32_t	An arbitrary value to add to the seed value in the hash function used by the bloom filter.
1	nFlags	uint8_t	A set of flags that control how outpointscorresponding to a matched pubkey script are added to the filter. See the table in the Updating A Bloom Filtersubsection below.

下面带注释的十六进制输出显示了一个 filterload 消息。(消息头已被省略。) 有关如何创建此有效负载的示例，请参阅 filterload 示例。

```
02 ......... Filter bytes: 2

b50f ....... Filter: 1010 1101 1111 0000

0b000000 ... nHashFuncs: 11

00000000 ... nTweak: 0/none

00 ......... nFlags: BLOOM_UPDATE_NONE
```

初始化 Bloom 过滤器

过滤器有两个核心参数：位字段的大小和对每个数据元素运行的散列函数的数量。以下 BIP37 中公式将允许你根据你计划插入过滤器的元素数量（n）和你希望保持 plausible-deniability 的假阳性率（p）自动选择适当的值。

- 位字段的大小（以字节为单位）（nFilterBytes），最大为 36,000：
  `(-1 / log(2)**2 * n * log(p)) / 8`
- 散列函数使用（nHashFuncs），最大值为 50：
  `nFilterBytes * 8 / n * log(2)`

请注意，过滤器匹配部分交易（交易元素），所以假阳性率是相对于被检查的元素的数量而不是被检查的交易的数量的。每个正常的交易至少有四个可匹配的元素（在下面的比较小节中有描述），因此，假阳性率为 1%的过滤器至少可以匹配所有交易的 4%。

根据 BIP37 的规定，上述公式和限值提供了对含有 20000 个条目的 Bloom 过滤器的支持，假阳性率小于 0.1%，或者 10000 个物品，假阳性率小于 0.0001%。

一旦位字段的大小已知，位字段应该被初始化为全零。

Populated A Bloom Filter

bloom 过滤器使用 1 到 50 个独一无二的散列函数（由 nHashFuncs 字段为每个过滤器指定的数字）填充。不是使用多达 50 个不同的散列函数实现，而是为每个函数使用一个唯一的 seed 值。

seed 是 nHashNum * 0xfba4c795 + nTweak，像 uint32_t，其中的值是：

- nHashNum 是这个散列函数的序列号，从第一个散列迭代的 0 开始，增加到最后一个散列迭代的 nHashFuncs 字段的值（减 1）。
- 0xfba4c795 是一个经过优化的常量，用于为 nHashNum 的不同值在 seed 中创建大的差异。
- nTweak 是由客户端设置的每个过滤器常量，以要求使用任意一组散列函数。

如果从上面的公式得到的 seed 大于四个字节，则必须将其截断为四个最重要的字节（例如，0x8967452301 & 0xffffffff→0x67452301 ）。

实际使用的散列函数实现是 32 位的 Murmur3 散列函数。

警告：Murmur3 散列函数具有单独的 32 位和 64 位版本，对同一输入产生不同的结果。Bitcoin bloom 过滤器只使用 32 位 Murmur3 版本。

要散列的数据可以是 bloom 过滤器可以匹配的任何交易元素。请参阅下一小节了解针对筛选器检查的交易元素列表。可以匹配的最大元素是 520 字节的脚本数据推送，所以数据不能超过 520 字节。

下面 Bitcoin Core 的 bloom.cpp 例子结合了上述所有步骤来创建哈希函数模板。seed 是第一个参数；要散列的数据是第二个参数。结果是一个 uint32_t 的模，位字段的大小，单位为比特。

```
MurmurHash3(nHashNum * 0xFBA4C795 + nTweak, vDataToHash) % (vData.size() * 8)
```

要添加到过滤器的每个数据元素都由 nHashFuncs 散列函数的数来散列。每次运行散列函数时，结果都将是位字段中一个位的索引号（nIndex ）。该位必须设置为 1。例如，如果过滤器位字段为 00000000 并且结果为 5，则修改后的过滤器位字段为 00000100（第一位为位 0 ）。

预计有时在填充比特字段时，相同的索引号会被多次返回；这不会影响算法—在一个比特设置为 1 之后，它不会再变回 0。

在所有的数据元素被添加到过滤器之后，每组 8 位被转换成小端字节。这些字节是过滤器字段的值。

比较交易元素和一个 Bloom 过滤器

要将任意数据元素与 Bloom 过滤器进行比较，使用与创建 Bloom 过滤器相同的参数进行散列。具体来说，它是被散列 nHashFuncs 次，每次使用过滤器中提供的相同 nTweak，并且结果输出是过滤字段中提供的位字段的大小的模。在执行每个散列之后，检查过滤器以查看该索引位置处的位是否被设置。例如，如果散列的结果是 5 并且过滤器是 01001110，则该位被认为是设置过的。

如果每个散列的结果都指向一个位，则过滤器匹配。如果任何结果指向未设置位，则过滤器不匹配。

以下交易元素与 bloom 过滤器进行比较。所有元素将按区块中使用的字节顺序散列（例如，TXID 将以内部字节顺序 ）。

- TXID：交易的 SHA256(SHA256())散列。
- Outpoints：使用这个交易的输入部分的每个 36 字节 outpoint 单独比较过滤器。
- 签名脚本数据（Signature Script Data）：将来自此交易的签名脚本中的数据推送操作码推送到堆栈上的每个元素单独与过滤器进行比较。这包括在 P2SH redeem 脚本中使用的数据元素。
- PubKey 脚本数据（PubKey Script Data）：通过该交易的任何 pubkey 脚本中的数据推送操作码将每个元素推送到堆栈上，并将其单独与过滤器进行比较。（如果一个 pubkey 脚本元素与过滤器匹配，则如果设置了 BLOOM_UPDATE_ALL 标志，过滤器将立即更新；如果 pubkey 脚本采用 P2PKH 格式且与过滤器匹配，则在设置了 BLOOM_UPDATE_P2PUBKEY_ONLY 标志后，将立即更新过滤器。有关详细信息，请参阅下面的小节。）

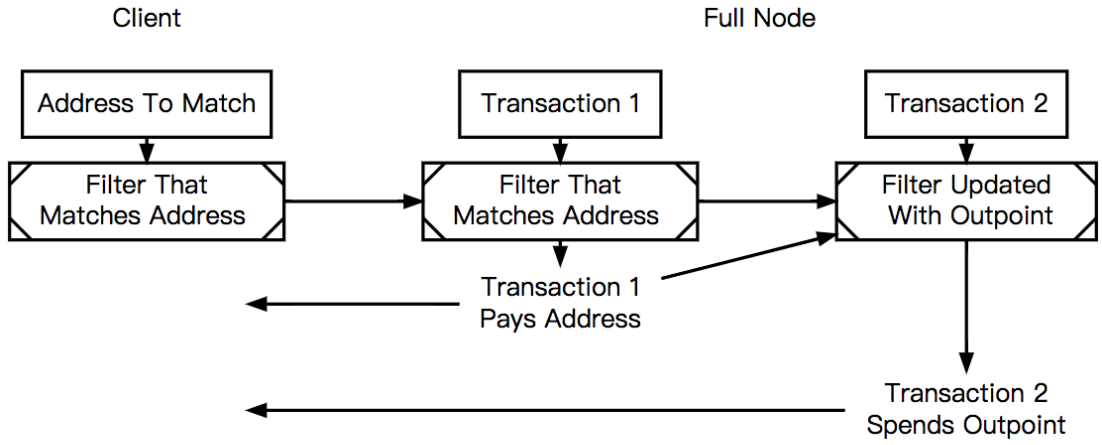
以下注释的一个交易的十六进制输出来自原始交易格式部分；过滤器将检查的元素将以粗体显示。请注意，此交易的 TXID（01000000017b1eab [...]）也将被检查，并且下面的 outpoint TXID 和索引号将被检查为单个 36 字节元素。

```
01000000 ................................... Version


01 ......................................... Number of inputs
|
| 7b1eabe0209b1fe794124575ef807057
| c77ada2138ae4fa8d6c4de0398a14f3f ......... Outpoint TXID
| 00000000 ................................... Outpoint index number
|
| 49 ......................................... Bytes in sig. script: 73
| | 48 ....................................... Push 72 bytes as data
| | | 30450221008949f0cb400094ad2b5eb3
| | | 99d59d01c14d73d8fe6e96df1a7150de
| | | b388ab8935022079656090d7f6bac4c9
| | | a94e0aad311a4268e082a725f8aeae05
| | | 73fb12ff866a5f01 ..................... Secp256k1 signature
|
```

```
| ffffffff ..................................... Sequence number: UINT32_MAX

01 ........................................... Number of outputs
| f0ca052a01000000 ........................... Satoshis (49.99990000 BTC)
|
| 19 ......................................... Bytes in pubkey script: 25
| | 76 ....................................... OP_DUP
| | a9 ....................................... OP_HASH160
| | 14 ...................................... Push 20 bytes as data
| | | cbc20a7664f2f69e5355aa427045bc15
| | | e7c6c772 ............................... PubKey hash
| | 88 ....................................... OP_EQUALVERIFY
| | ac ....................................... OP_CHECKSIG

00000000 ..................................... locktime: 0 (a block height)
```

更新 Bloom 过滤器

客户端经常要追踪与其钱包有关的消耗的输出（outpoints）的输入，因此可以设置 filterload 字段 nFlags 以允许过滤节点在找到匹配时更新过滤器。当过滤节点看到支付与过滤器匹配的 pubkey ,address 或其他数据元素的 pubkey 脚本时，过滤节点立即使用与该 pubkey 脚本相对应的 outpoint 来更新过滤器。



Automatically Updating Bloom Filters To Track Relevant Transactions

如果一个输入稍后消耗了那个 outpoint，那么过滤器将匹配它，允许过滤节点告诉客户端其交易输出中的一个已经被消耗了。

nFlags 字段有三个允许值：

值	名称	描述
0	BLOOM_UPDATE_NONE	The filtering node should not update the filter.
1	BLOOM_UPDATE_ALL	If the filter matches any data element in a pubkey script, the corresponding outpoint is added to the filter.
2	BLOOM_UPDATE_P2PUBKEY_ONLY	If the filter matches any data element in a pubkey script and that script is either a P2PKH or non-P2SH pay-to-multisigscript, the corresponding outpoint is added to the filter.

另外，由于即使添加了附加元素，过滤器大小也保持不变，所以假阳性率增加。每个假阳性都可能导致另一个元素被添加到过滤器中，从而创建一个反馈回路，可以（在某个点之后）使过滤器无用。因此，使用自动过滤器更新的客户端需要监视实际的假阳性率，并在费率过高时发送新的过滤器。

GetAddr

getaddr 消息请求来自接收节点的 addr 消息，最好是具有大量其他接收节点的 IP 地址的消息。发送节点可以使用这些 IP 地址来快速更新其可用节点的数据库，而不是等待 unsolicited 的 addr 消息超时到达。

getaddr 消息中没有有效载荷。 有关负载的消息示例，请参阅消息头部分。

Ping

ping 消息有助于确认接收方仍处于连接状态。如果在发送 ping 消息时遇到 TCP / IP 错误(例如连接超时),则发送节点可以假定接收节点已断开连接。ping 消息的响应是 pong 消息。

在协议版本 60000 之前，ping 消息没有有效载荷。从协议版本 60001 及所有更高版本开始，消息包含一个字段即 nonce。

字节数	名称	数据类型	描述
8	nonce	uint64_t	*Added in protocol version 60001 as described by BIP31.* Random nonce assigned to this `ping` message. The responding `pong` message will include this nonce to identify the `ping` message to which it is replying.

下面带注释的十六进制输出显示了一条 ping 消息。(消息头已被省略。)

```
0094102111e2af4d ... Nonce
```

Pong

如 BIP31 所述，在协议版本 60001 中添加。

pong 消息回复 ping 消息，向 pinging 节点证明 ponging 节点仍然存活。在默认情况下，Bitcoin Core 将在 20 分钟内断开与 ping 消息没有响应的任何客户端的连接。

为了允许节点跟踪延迟，pong 消息发送回应的 ping 消息中收到的相同的 nonce。

pong 消息的格式与 ping 消息相同，只是消息头不同。

Reject

如 BIP61 所述，在协议版本 70002 中加入。

Reject 消息通知接收方，它之前的消息之一已被拒绝。

字节数	名称	数据类型	描述
Varies	message bytes	compactSize uint	The number of bytes in the following message field.
Varies	message	string	The type of message rejected as ASCII text *without null padding*. For example: "tx", "block", or "version".
1	code	char	The reject message code. See the table below.
Varies	reason bytes	compactSize uint	The number of bytes in the following reason field. May be oxoo if a text reason isn't provided.
Varies	reason	string	The reason for the rejection in ASCII text. This should not be displayed to the user; it is only for debugging purposes.
Varies	extra data	*varies*	Optional additional data provided with the rejection. For example, most rejections of `tx` messages or `block` messages include the hash of the rejected transaction or block header. See the code table below.

下表列出了消息拒绝代码。代码与他们回复的消息类型相关联；例如，交易中有一个 0x10 拒绝代码，区块中有一个 0x10 拒绝代码。

Code	In Reply To	Extra Bytes	Extra Type	Description

Code	In Reply To	Extra Bytes	Extra Type	Description
0x01	*any message*	0	N/A	Message could not be decoded. Be careful of `reject` message feedback loops where two peers each don't understand each other's `reject` messages and so keep sending them back and forth forever.
0x10	`block` message	32	char[32]	Block is invalid for some reason (invalid proof-of-work, invalid signature, etc). Extra data may include the rejected block's header hash.
0x10	`tx` message	32	char[32]	Transaction is invalid for some reason (invalid signature, output value greater than input, etc.). Extra data may include the rejected transaction's TXID.
0x11	`block` message	32	char[32]	The block uses a version that is no longer supported. Extra data may include the rejected block's headerhash.
0x11	`version` message	0	N/A	Connecting node is using a protocol version that the rejecting nodeconsiders obsolete

Code	In Reply To	Extra Bytes	Extra Type	Description
				and unsupported.
0x12	`tx` message	32	char[32]	Duplicate input spend (double spend): the rejected transaction spends the same input as a previously-received transaction. Extra data may include the rejected transaction's TXID.
0x12	`version` message	0	N/A	More than one `version` messagereceived in this connection.
0x40	`tx` message	32	char[32]	The transaction will not be mined or relayed because the rejecting nodeconsiders it non-standard—a transaction type or version unknown by the server. Extra data may include the rejected transaction's TXID.
0x41	`tx` message	32	char[32]	One or more output amounts are below the dust threshold. Extra data may include the rejected transaction's TXID.
0x42	`tx` message		char[32]	The transaction did not have a large enough fee or priority to be relayed or mined. Extra data may include the rejected

Code	In Reply To	Extra Bytes	Extra Type	Description
				transaction's TXID.
0x43	`block` message	32	char[32]	The block belongs to a block chainwhich is not the same block chain as provided by a compiled-in checkpoint. Extra data may include the rejected block's header hash.

下述注释十六进制输出展示了一个 reject 消息。（消息头已被忽略。）

```
02 ............................... Number of bytes in message: 2
7478 ............................. Type of message rejected: tx
12 ............................... Reject code: 0x12 (duplicate)
15 ............................... Number of bytes in reason: 21
6261642d74786e732d696e707574732d
7370656e74 ....................... Reason: bad-txns-inputs-spent
394715fcab51093be7bfca5a31005972
947baf86a31017939575fb2354222821 ... TXID
```

SendHeaders

sendheaders 消息告诉接收方使用一个 headers 消息而不是 inv 消息发送新的区块公告。

sendheaders 消息中没有有效载荷。 有关负载的消息示例，请参阅消息头部分。

VerAck

在协议版本 209 中添加。

verack 消息确认先前收到的 version 消息，通知连接节点它可以开始发送其他消息。verack 消息没有有效载荷；有关没有负载的消息示例，请参阅消息头部分。

Version

version 消息在连接开始时向接收节点提供关于发送节点的信息。在两个对等点交换 version 消息之前，不会接受其他消息。

如果一个 version 消息被接受，接收节点应该发送一个 verack 消息—但是在通过首先发送 version 消息来初始化它的一半连接之前，应该没有节点发送一个 verack 消息。

字节数	名称	数据类型	必须／可选	描述
4	version	int32_t	Required	The highest protocol version understood by the transmitting node. See the protocol version section.
8	services	uint64_t	Required	The services supported by the transmitting nodeencoded as a bitfield. See the list of service codes below.
8	timestamp	int64_t	Required	The current Unix epoch timeaccording to the transmitting node'sclock. Because nodes will reject blocks with timestamps more than two hours in the future, this field can help other nodes to determine that their clock is wrong.
8	addr_recv services	uint64_t	Required	The services supported by the receiving node as perceived by the transmitting node. Same format as the 'services' field above. Bitcoin Core will

字节数	名称	数据类型	必须／可选	描述
				attempt to provide accurate information.BitcoinJ will, by default, always send 0.
16	addr_recv IP address	char	Required	The IPv6 address of the receiving node as perceived by the transmitting node in **big endian byte order**. IPv4 addresses can be provided as IPv4-mapped IPv6 addresses. Bitcoin Core will attempt to provide accurate information.BitcoinJ will, by default, always return ::ffff:127.0.0.1
2	addr_recv port	uint16_t	Required	The port number of the receiving nodeas perceived by the transmitting nodein **big endian byte order**.
8	addr_trans services	uint64_t	Required	*Added in protocol version 106.* The services supported by the transmitting node. Should be identical to the 'services' field above.
16	addr_trans IP address	char	Required	*Added in protocol version 106.* The IPv6 address of the transmitting node in **big endian byte order**. IPv4 addresses can be provided as IPv4-mapped IPv6 addresses. Set to ::ffff:127.0.0.1 if unknown.

字节数	名称	数据类型	必须／可选	描述
2	addr_trans port	uint16_t	Required	*Added in protocol version 106.* The port number of the transmitting node in **big endian byte order**.
8	nonce	uint64_t	Required	*Added in protocol version 106.* A random nonce which can help a node detect a connection to itself. If the nonce is 0, the nonce field is ignored. If the nonce is anything else, a node should terminate the connection on receipt of a `version` messagewith a nonce it previously sent.
Varies	user_agent bytes	compact Size uint	Required	*Added in protocol version 106.* Number of bytes in following user_agent field. If 0x00, no user agent field is sent.
Varies	user_agent	string	Required if user_agent bytes > 0	*Added in protocol version 106. Renamed in protocol version 60000.* User agent as defined by BIP14. Previously called subVer.
4	start_he	int32_t	Require	*Added in protocol version 209.*

字节数	名称	数据类型	必须／可选	描述
	ight		d	The height of the transmitting node'sbest block chain or, in the case of an SPV client, best block header chain.
1	relay	bool	Optional	*Added in protocol version 70001 as described by BIP37.* Transaction relay flag. If 0x00, no `inv` messages or `tx` messagesannouncing new transactions should be sent to this client until it sends a `filterload` message or `filterclear` message. If the relay field is not present or is set to 0x01, this nodewants `inv` messages and `tx` messagesannouncing new transactions.

下述服务标识符已被分配。

值	名称	描述
0x00	*Unnamed*	This node is not a full node. It may not be able to provide any data except for the transactions it originates.
0x01	NODE_NETWORK	This is a full node and can be asked for full blocks. It should implement all protocol features available in its self-reported protocol version.

以下带注释的十六进制输出显示了一个 version 信息。（消息头已被省略，实际 IP 地址已被 RFC5737 保留的 IP 地址替代。）

```
72110100 ........................... Protocol version: 70002
0100000000000000 ................... Services: NODE_NETWORK
bc8f5e5400000000 ................... Epoch time: 1415483324

0100000000000000 ................... Receiving node's services
00000000000000000000ffffc61b6409 ... Receiving node's IPv6 address
208d ............................... Receiving node's port number

0100000000000000 ................... Transmitting node's services
00000000000000000000ffffcb0071c0 ... Transmitting node's IPv6 address
208d ............................... Transmitting node's port number

128035cbc97953f8 ................... Nonce

0f ................................. Bytes in user agent string: 15
2f5361746f7368693a302e392e332f ..... User agent: /Satoshi:0.9.3/

cf050500 ........................... Start height: 329167
01 ................................. Relay flag: true
```