P2P网络基础

节点：

比特币网络是纯P2P体系，每个节点既是服务器，也是客户端。节点既要监听网络，要连接其他节点，既要接收消息，也要发送消息。

节点允许的最大连接数为125个，可以通过参数”-maxconnections”修改。向外最多连接8个点（net.h）。

```
static const int MAX_OUTBOUND_CONNECTIONS = 8;
static const unsigned int DEFAULT_MAX_PEER_CONNECTIONS = 125;
```

系统定义了信号量(CSemaphore)，用于控制网络连接时的最大数量，信号量的最大值为连接数的最大值。

节点的信息定义在类CNode中，包含了节点需要的socket、消息、地址、ping 数据等。

系统定义了节点数组(vector<CNode*> vNodes)，包含了连接的所有节点。

当节点连接上，则把此节点添加到节点数组中；断开连接后，从节点数组中移除此节点。

网络启动和停止：

节点通过CConnman类对网络操作进行管理，在init.cpp中定义了一个全局的g_connman，在init.cpp->AppInitMain中调用CConnman的Start进行网络初始化，启动节点。调用绑定监听接口相关函数为：

`InitBinds`	调用Bind
`Bind`	调用BindListenPort
`BindListenPort`	创建socket，绑定和监听端口，把socket放入vhListenSocket

在init.cpp->Shutdown中调用CConnman的Stop关闭网络。

网络线程：

节点启动时，启动了多个线程， 用于地址、UPNP、监听、消息等。

1. socket线程

此线程主要是监听 络，连接节点，接收、发送消息。线程名为”bitcoin- net"。主函数是ThreadSocketHandler。
此线程调用select函数监听socket， 监听SOCKET数组(vhListenSocket)、节点数组(vNodes)， 为每个节点的socket 设置发送、接收fd_set。
遍历节点数组， 当节点标识断开连接(fDisconnect)， 或者没有任何引 、发 送接收消息， 则移除节点， 关闭socket， 添加到断开连接节点数组(vNodesDisconnected)
遍历监听socket数组， 当接收到数据， 且socket有效时， 接受连接， 新建节点， 添加到节点数组
遍历节点数组， 接收网络数据， 解析成消息， 添加到节点的接收消息数组 (vRecvMsg)， 转移到vProcessMsg供ThreadMessageHandler处理。当发送集合有数据时， 把节点的发送消息(vSendMsg)发送出
检查不活跃的socket连接

2. DNS节点线程

当系统启 了从DNS中加载地址时， 则创建此线程。默认是启 ， 但可以通过 设置参数”-dnsseed"来禁 。线程名为”bitcoin-dnsseed"。线程主函数是ThreadDNSAddressSeed。
在系统参数中保存了一些默认的DNS地址信息， 保存在数组中(CDNSSeedData)。
获取DNS节点数组中地址信息，查找主机的IP， 构造成CAddress， 添加到地址管理器中(CAddrMan addrman)。 addrman将会被ThreadOpenConnections线程使用
地址的时间设置为过去3 7天的随机值。

3. ADDNODE线程

此线程 于连接 户在RPC中在参数'-addnode'中指定的外部节点，线程名 为"bitcoin-addcon"。主函数是ThreadOpenAddedConnections。
把参数"-addnode"中的节点字符串添加到节点数组(vNodes)。vNodes在ThreadSocketHandler线程中被监听

4.OPENCONNECTION线程

此线程用于连接-connect或者DNS指定的地址，线程名为"bitcoin-opencon"。主函数是 ThreadOpenConnections。
当指定了参数"-connect"时，连接地址字符串。
从地址管理器中选择一个地址，连接此地址(OpenNetworkConnection)，连接后加入vNodes

5.消息处理线程

此线程用于连接-connect或者DNS指定的地址，线程名为"bitcoin-msghand"。主函数是 ThreadMessageHandler。
从CNode的vProcessMsg中接收消息，调用PeerLogicValidation::***ProcessMessages***
调用`PeerLogicValidation::SendMessages`触发消息发送（触发`CConnman::PushMessage`填充`CNode`的`vSendMsg`），最终`ThreadSocketHandler`线程发送数据
发送消息时，检验是否有命令需要发送，如:ping、addr、getheaders, getblocks、 inv、getdata 等

PeerLogicValidation：

是一个消息处理的关键类，用于发送和处理消息，实现了NetEventsInterface接口

```
bool ProcessMessages(CNode* pfrom, std::atomic<bool>& interrupt) override;
bool SendMessages(CNode* pto, std::atomic<bool>& interrupt) override;
```

net.cpp有一个全局 NetEventsInterface* m_msgproc变量，用于ThreadMessageHandler接口调用。

在RegisterValidationInterface(peerLogic.get())注册信号，用于validation.cpp中调用

发送消息过程：

1. 通过直接调用CConnman::PushMessage发送消息
2. 把消息写入CNode的对应数据字段中，由ThreadMessageHandler定时发送（100ms)

例子：广播block过程：

valication.cpp:AcceptBlock->GetMainSignals().NewPoWValidBlock

network-processing: PeerLogicValidation->NewPoWValidBlock
connman->PushMessage

相关代码文件:

netbase.cpp:
最底层网络函数接口封装。socket. send, recv等，基本上无需再改动

net.cpp:
包含CConnman和CNode的代码。调用netbase.cpp，比特币底层网络机制的核心逻辑所在

net_process.cpp:
包含PeerLogicValidation代码，属于上层比特币业务逻辑和底层net.cpp中的中间层。调用net.cpp
其中ProcessMessage是一个重要函数，包含了比特币的网络命令等逻辑

P2P网络协议

网络：

比特币网络中的节点之间可以发送、接收消息进行通信， 网络协议基于TCP协议， 支持IPV4、IPV6、TOR网络。

可以通过设置参数”-onlynet"指明只支持哪种网络。

消息:

网络络通信以消息为单位， 消息格式为:消息头 + 数据。

消息头:

消息头的类为CMessageHeader， 定义了消息头的格式。

消息头的格式:

格式	度(字节)
消息开始字符串(MessageStart)	4
命令(Command)	12
消息大小(MessageSize)	符号整型
校验和(Checksum)	符号整型

1. 消息开始字符串

定义了消息标识， 采用在正常数据中极少出现的字符， 不用大写ASCII码字 符， 任意对齐 ， 长度为4字节， 在UTF-8中无效。

在不同的环境中， 消息开始字符串不同。

环境类型	消息开始字符串
主类型(MAIN)	0xd9b4bef9
测试 络(TESTNET)	0x0709110b
回归测试(REGTEST)	0xdab6bffa

2. 命令

定义了通信中的各种命令， 长度为12个字节的字符串，由0x20~0x7F之间的字符串构成，其他字符无效。

3. 消息大小

定义了消息的大小，不包含消息头的大小，是无符号整型，最 值是32M (0x02000000)。

4. 校验和

消息的校验和，不包含消息头的数据，是无符号整型。

把消息数据经过2次SHA256算法运算得到校验和。

5. 有效性

满足以下3点才是有效的消息:

1. 消息开始字符串与环境参数定义的字符串相同。
2. 命令字符串第一个字符为0时，其余字符必须全部为0;不为0时，必须由0x20~0x7F之间的字符串构成。
3. 消息大小不能超过32M。

命令:

protocol.cpp中定义了以下命令：

```
namespace NetMsgType {
const char *VERSION="version";
const char *VERACK="verack";
const char *ADDR="addr";
const char *INV="inv";
const char *GETDATA="getdata";
const char *MERKLEBLOCK="merkleblock";
const char *GETBLOCKS="getblocks";
const char *GETHEADERS="getheaders";
const char *TX="tx";
const char *HEADERS="headers";
const char *BLOCK="block";
```

```
const char *GETADDR="getaddr";
const char *MEMPOOL="mempool";
const char *PING="ping";
const char *PONG="pong";
const char *NOTFOUND="notfound";
const char *FILTERLOAD="filterload";
const char *FILTERADD="filteradd";
const char *FILTERCLEAR="filterclear";
const char *REJECT="reject";
const char *SENDHEADERS="sendheaders";
const char *FEEFILTER="feefilter";
const char *SENDCMPCT="sendcmpct";
const char *CMPCTBLOCK="cmpctblock";
const char *GETBLOCKTXN="getblocktxn";
const char *BLOCKTXN="blocktxn";
} // namespace NetMsgType
```

getdata / inv的消息子类型：

```
enum GetDataMsg
{
    UNDEFINED = 0,
    MSG_TX = 1,
    MSG_BLOCK = 2,
    // The following can only occur in getdata. Invs always use TX or BLOCK.
    MSG_FILTERED_BLOCK = 3,  //!< Defined in BIP37
    MSG_CMPCT_BLOCK = 4,     //!< Defined in BIP152
    MSG_WITNESS_BLOCK = MSG_BLOCK | MSG_WITNESS_FLAG, //!< Defined in BIP144
    MSG_WITNESS_TX = MSG_TX | MSG_WITNESS_FLAG,       //!< Defined in BIP144
    MSG_FILTERED_WITNESS_BLOCK = MSG_FILTERED_BLOCK | MSG_WITNESS_FLAG,
};
```

版本号：

当前比特币网络协议版本号为70015（version.h）

```
static const int PROTOCOL_VERSION = 70015;
```