比特币架构

路远

中国科学院软件研究所

本讲内容

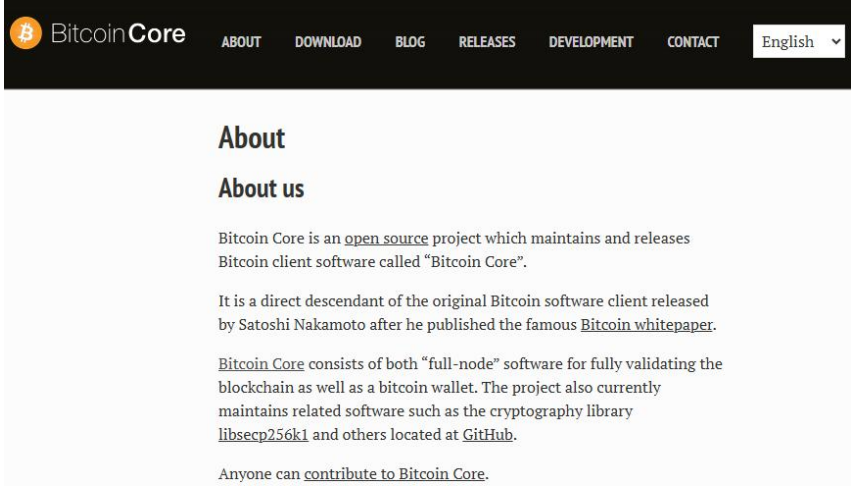
- Bitcoin Core 架构总览
  - 用户接口
  - 并发模型
  - 代码分区
  - 持久存储
  - 数据结构
- 区块链查询
  - 自建全节点
  - 第三方“浏览器”
  - 轻客户端协议

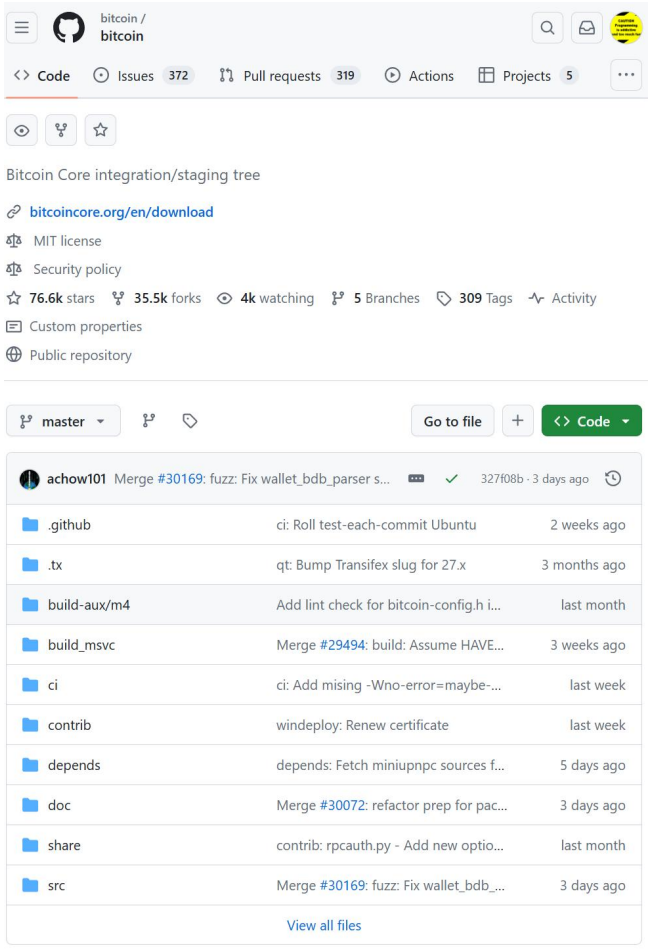
Bitcoin Core 架构总览

Bitcoin Core

比特币社区的参考实现

- **全节点参考实现**
  - 脚本VM、状态 (UTXOs) 维护、区块验证
  - P2P网络、交易中继、区块中继、交易池
- **钱包权威实现**
  - 面向终端用于的 GUI/RPC 工具
  - UTXO选择策略 (Coin Select)
- **矿工节点**
  - 新区块的组装
  - PoW软件参考实现
- **区块链应用的编程接口**
  - RPC (远程过程调用)、CLI (命令行界面)
- **各类可重用的代码库**
  - 序列化方法、脚本VM、密码库

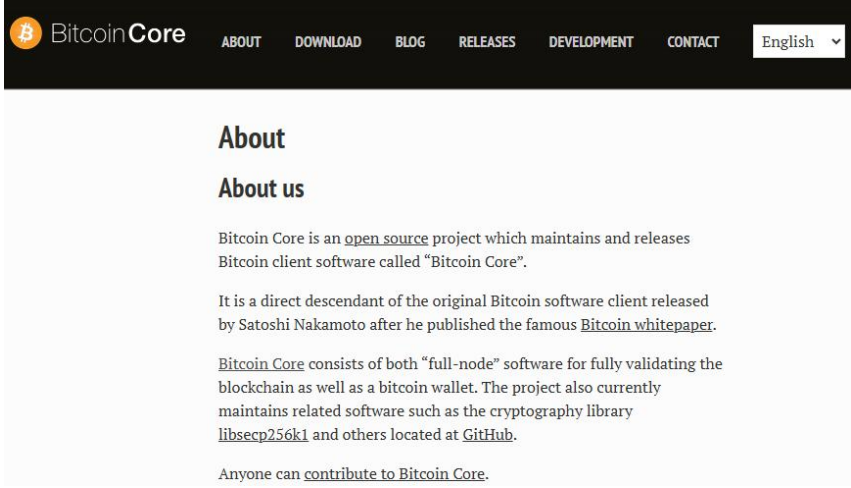


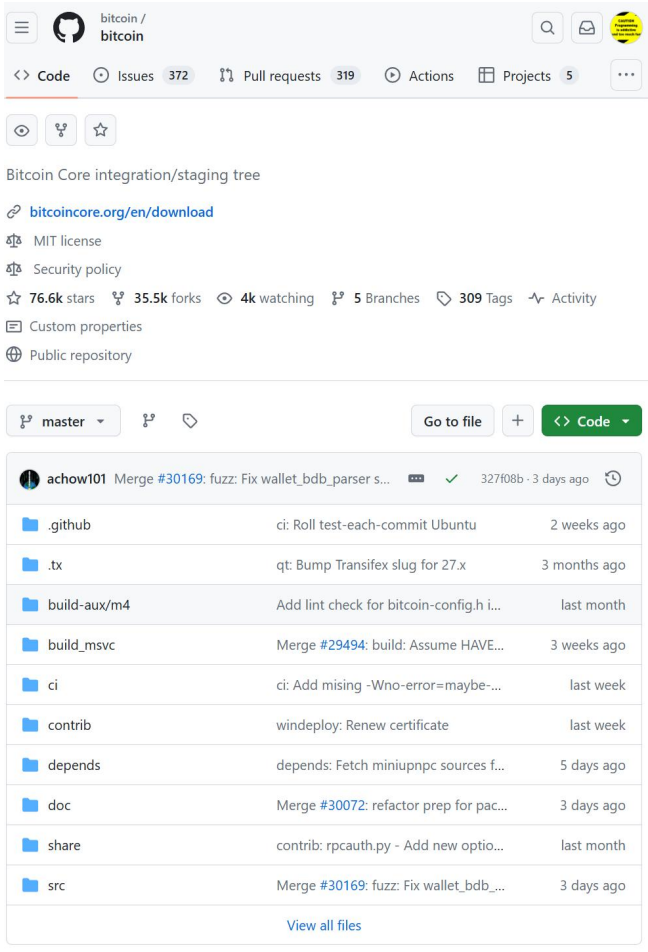


Bitcoin Core

比特币社区的参考实现

- **并发模型**
  - 如何并发地执行多项事务?
  - 比如：P2P网络、PoW挖矿 如何同时执行?
- **代码功能分区**
  - 代码架构怎样设计?
  - 比如：哪些子功能模块?
- **持久存储**
  - 数据如何进行持久化?
  - 比如：区块链如何在硬盘存储
- **数据结构**
  - 内存中的数据结构如何表示和操作
  - 比如：区块、交易的结构如何?





Bitcoin Core：用户接口

用户可选参数

用户可选参数：Bitcoin Core节点允许用户选择参数

- Bitcoin forms a TCP overlay network of nodes passing messages to one another
  - Messages defined in `src/protocol.h`
- Each node has a set of outbound and inbound peers they exchange data with
  - `-addnode=<addr>`
  - `-maxconnections=<n>`
  - `net.h MAX_OUTBOUND_CONNECTIONS, DEFAULT_MAX_PEER_CONNECTIONS`
- Peers can be manually added (`-addnode`) or are discovered from DNS seeds: DNS servers that randomly resolve to known Bitcoin nodes
- DoS protection is implemented to prevent malicious peers from disrupting the network
  - `-banscore=<n>` configures sensitivity, defaults to `100`
- SPV (simple payment verification) nodes retrieve txout proofs

用户接口

RPC接口：允许用户通过HTTP与Bitcoin Core交互

- 允许远程的用户查询区块和交易数据
- 允许远程的用户申请签名和提交交易
- 允许远程的矿池查询尚缺少PoW的区块

CLI接口：允许用户通过命令行与Bitcoin Core交互

- 允许本地的用户查询区块和交易数据
- 允许本地的用户申请签名和提交交易
- 允许本地的矿池查询尚缺少PoW的区块

用户接口

区块链相关API：

- GetBestBlockHash: 返回最优链上最近区块的哈希
- **GetBlock: 返回具有指定哈希的区块**
- GetBlockChainInfo: 返回区块链当前状态信息
- GetBlockCount: 返回本地最优链上的区块数量
- GetBlockHash: 返回本地最有区块链上指定高度区块的哈希
- GetBlockHeader: 返回指定区块头
- GetChainTips: 返回每个本地区块链的最高位区块（tip）信息
- GetDifficulty: 返回PoW难度
- GetMemPoolAncestors: 返回交易池内指定交易的所有祖先
- GetMemPoolDescendants: 返回交易池内指定交易的所有后代
- GetMemPoolEntry: 返回交易池内指定交易的池数据
- GetMemPoolInfo: 返回交易池信息
- GetRawMemPool: 返回交易池内的所有交易
- GetTxOut: 返回指定交易输出的详细信息
- GetTxOutProof: 返回一个或多个交易的证明数据
- GetTxOutSetInfo: 返回UTXO集合的统计信息
- PreciousBlock:
- PruneBlockChain: 对区块链执行剪枝操作
- VerifyChain: 验证本地区块链的每个记录
- VerifyTxOutProof: 验证交易输出证明

Blockchain RPCs

- getbestblockhash
- getblock
- getblockchaininfo
- getblockcount
- getblockfilter
- getblockhash
- getblockheader
- getblockstats
- getchaintips
- getchaintxstats
- getdifficulty
- getmempoolancestors
- getmempooldescendants
- getmempoolentry
- getmempoolinfo
- getrawmempool
- gettxout
- gettxoutproof
- gettxoutsetinfo
- preciousblock
- pruneblockchain
- savemempool
- scantxoutset
- verifychain
- verifytxoutproof

用户接口

区块链相关API：

- **GetBlock: 返回具有指定哈希的区块**

getblock

```
getblock "blockhash" ( verbosity )
```

Argument #1 - blockhash

Type: string, required

The block hash

Argument #2 - verbosity

Type: numeric, optional, default=1

0 for hex-encoded data, 1 for a json object, and 2 for json object with transaction data

0 for hex-encoded data,
1 for a json object,
2 for json object with transaction data

Blockchain RPCs

- getbestblockhash
- getblock
- getblockchaininfo
- getblockcount
- getblockfilter
- getblockhash
- getblockheader
- getblockstats
- getchaintips
- getchaintxstats
- getdifficulty
- getmempoolancestors
- getmempooldescendants
- getmempoolentry
- getmempoolinfo
- getrawmempool
- gettxout
- gettxoutproof
- gettxoutsetinfo
- preciousblock
- pruneblockchain
- savemempool
- scantxoutset
- verifychain
- verifytxoutproof

用户接口

节点控制API：

- GetInfo: 返回节点和网络信息
- Help: 返回所有可用的RPC命令，或返回指定命令的帮助信息
- Stop: 安全关闭bitcoin core的节点服务

出块相关API：

- Generate: 生成区块
- GenerateToAddress: 生成区块并将新生成的比特币转入指定地址

挖矿相关API：

- GetBlockTemplate: 返回节点模板
- GetMiningInfo: 返回挖矿相关信息
- GetNetworkHashPS: 返回估算的全网哈希速率
- PrioritiseTransaction: 交易优先权

Control RPCs ¶

- getmemoryinfo
- getrpcinfo
- help
- logging
- stop
- uptime

Generating RPCs

- generateblock
- generatetoaddress
- generatetodescriptor

Mining RPCs

- getblocktemplate
- getmininginfo
- getnetworkhashps
- prioritisetransaction
- submitblock
- submitheader

用户接口

网络相关API：

- AddNode: 添加节点
- ClearBanned: 清理禁止的节点
- DisconnectNode: 断开与指定节点旳连接
- GetAddedNodeInfo: 返回新增节点的信息
- GetConnectionCount: 返回与其他节点旳连接总数量
- GetNetTotals: 返回网络流量统计信息
- GetNetworkInfo: 返回节点的网络连接信息
- GetPeerInfo: 返回所连接其他节点旳信息
- ListBanned: 返回所有被禁止的IP或子网
- Ping: 向所有连接的节点发送p2p的pin报文
- SetBan: 管理禁止访问清单
- SetNetworkActive: 禁止/启用P2P网络

Network RPCs

- addnode
- clearbanned
- disconnectnode
- getaddednodeinfo
- getconnectioncount
- getnettotals
- getnetworkinfo
- getnodeaddresses
- getpeerinfo
- listbanned
- ping
- setban
- setnetworkactive

用户接口

裸交易(serialized hex-encoded)相关API：

- CreateRawTransaction: 创建未签名的序列化交易
- FundRawTransaction: 向裸交易添加新的UTXO
- DecodeRawTransaction: 解码指定的裸交易
- DecodeScript: 解码指定的P2SH赎回脚本
- GetRawTransaction: 返回指定的裸交易
- SendRawTransaction: 验证并发送裸交易到P2P网络
- SignRawTransaction: 签名裸交易
- .....

工具类API：

- CreateMultiSig: 创建P2SH多重签名地址
- EstimateFee: 估算交易费率
- EstimatePriority: 估算交易的优先级
- GetMemoryInfo: 返回内存使用情况
- ValidateAddress: 验证指定的地址
- VerifyMessage: 验证签名的消息

Rawtransactions RPCs

- analyzepsbt
- combinepsbt
- combinerawtransaction
- converttopsbt
- createpsbt
- createrawtransaction
- decodepsbt
- decoderawtransaction
- decodescript
- finalizepsbt
- fundrawtransaction
- getrawtransaction
- joinpsbts
- sendrawtransaction
- signrawtransactionwithkey
- testmempoolaccept
- utxoupdatepsbt

Util RPCs

- createmultisig
- deriveaddresses
- estimatesmartfee
- getdescriptorinfo
- getindexinfo
- signmessagewithprivkey
- validateaddress
- verifymessage

用户接口

钱包相关API：

- SendToAddress: 向指定地址发送比特币
- SetAccount: 将指定地址与账户关联
- SetTxFee: 设置千字节交易费率
- SignMessage: 签名消息
- SignMessageWithPrivKey: 使用指定私钥签名消息
- WalletLock: 锁定钱包
- WalletPassphrase: 输入钱包口令
- WalletPassphraseChange: 修改钱包口令
- ……

Wallet RPCs

Note: the wallet RPCs are only available if Bitcoin Core was built with wallet support, which is the default.

- abandontransaction
- abortrescan
- addmultisigaddress
- backupwallet
- bumpfee
- createwallet
- dumpprivkey
- dumpwallet
- encryptwallet
- getaddressesbylabel
- getaddressinfo
- getbalance
- getbalances
- getnewaddress
- getrawchangeaddress
- getreceivedbyaddress
- getreceivedbylabel
- gettransaction
- getunconfirmedbalance
- getwalletinfo
- importaddress
- importdescriptors
- importmulti
- importprivkey

用户接口

钱包相关API：

- **WalletLock: 锁定钱包**
- **WalletPassphrase: 输入钱包口令**
- **SendToAddress: 发送交易给指定地址**

.....

钱包API使用举例：

Set the passphrase for 2 minutes to perform a transaction: **步骤1：解锁密钥120秒 (口令解密)**

```
bitcoin-cli walletpassphrase "my pass phrase" 120
```

Perform a send (requires passphrase set): **步骤2：发送交易到指定地址**

```
bitcoin-cli sendtoaddress "bc1q09vm5lfy0j5reeulh4x5752q25uqqvz34hufdl" 1.0
```

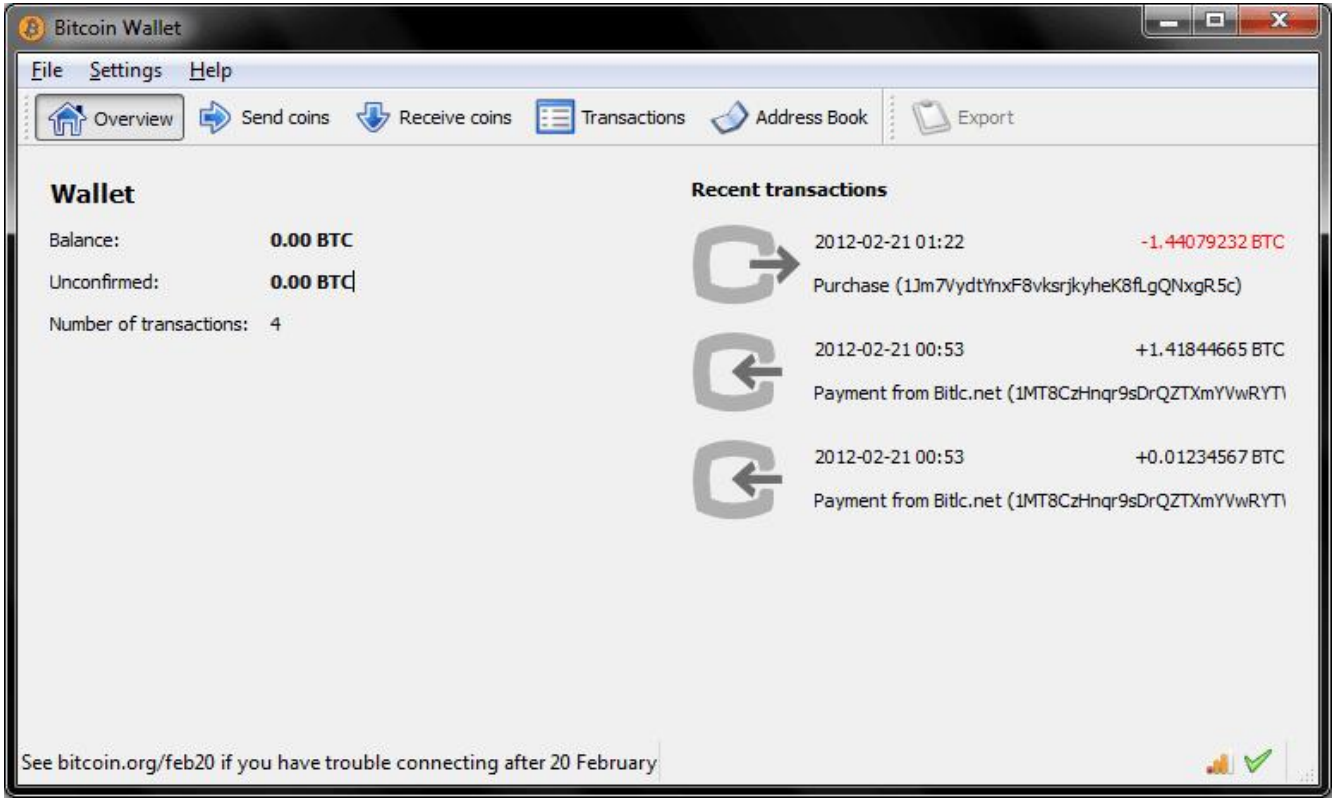
Clear the passphrase since we are done before 2 minutes is up: **步骤3：锁定钱包(在内存中删除私钥)**

```
bitcoin-cli walletlock
```

可视化QT界面

可视化QT界面

- 可视化的钱包功能
- 交易统计信息
- RPC控制台
- .....



ZeroMQ (ZMQ) 接口

基于 轻量化进程间通信框架 ZMQ 的用户接口 (默认开启)

用户可以通过ZMQ 接口，订阅：

- 新区块 (raw)：rawblock
- 新区块 (hash)：hashblock
- 新交易 (raw)：rawtx
- 新交易 (hash)：hashtx

```
# From `contrib/zmq/zmq_sub.py`

zmqSubSocket = self.zmqContext.socket(zmq.SUB)
zmqSubSocket.setsockopt_string(zmq.SUBSCRIBE, "hashblock")
msg = await zmqSubSocket.recv_multipart()
topic, body, *_ = msg

if topic == b"hashblock":
    print('saw hashblock')
    print(binascii.hexlify(body))
```

Bitcoin Core：并发模型

并发模型

Bitcoin Core 中，需要同时执行一系列的并发任务

- 采用多线程设计，基于 std::threads 或者 boost::threads
- 多线程共享内存，必要时需要使用线程锁
- 大多数线程在 init.cpp 中 (直接或间接) 的启动
- P2P网络通信采用完全单独的线程 (CConman::ThreadSocketHandler)
- 所有的状态变 顺序处理！

线程统计

Purpose	# threads	Task run
Script verification	`nproc or 16` *	`ThreadScriptCheck()`
Loading blocks	1	`ThreadImport()`
Servicing RPC calls	4*	`ThreadHTTP()`
Load peer addresses from DNS seeds	1	`ThreadDNSAddressSeed()`
Send and receive messages to and from peers	1	`ThreadSocketHandler()`
Initializing network connections	1	`ThreadOpenConnections()`
Opening added network connections	1	`ThreadOpenAddedConnections()`
Process messages from `net` -> `net_processing`	1	`ThreadMessageHandler()`

线程统计 cont.

Purpose	# threads	Task run
Tor control	1	`TorControlThread()`
Wallet notify (`-walletnotify`)	1	user-specified
txindex building	1	`ThreadSync()`
Block notify (`-blocknotify`)	1	user-specified
Upnp connectivity	1	`ThreadMapPort()`
`CScheduler` service queue (powers `ValidationInterface`)	1	`CScheduler::serviceQueue()`

Bitcoin Core：代码分区

代码分区

Name	Purpose
`net`	Handles socket networking, tracking of peers
`net_processing`	Routes P2P messages into validation calls and response P2P messages
`validation`	Defines how we update our validated state (chain, mempool)
`txmempool`	Mempool data structures
`coins & txdb`	Interface for in-memory view of the UTXO set
`script/`	Script execution and caching
`consensus/`	Consensus params, Merkle roots, some TX validation
`policy/`	Fee estimation, replace-by-fee
`indexes/`	Peripheral index building (e.g. `txindex`)
`wallet/`	Wallet db, coin selection, fee bumping
`rpc/`	Defines the RPC interface

代码分区： net

Regions > `net.{h,cpp}`

`net` is the "bottom" of the Bitcoin core stack. It handles network communication with the P2P network.

It contains addresses and statistics (`CNodeStats`) for peers (`CNode`s) that the running node is aware of.

`CConman` is the main class in this region - it manages socket connections (and network interaction more generally) for each peer, and forwards messages to the `net_processing` region (via `CConman::ThreadMessageHandler`).

代码分区： net_processing

Regions > `net_processing.{h,cpp}`

`net_processing` adapts the network layer to the chainstate validation layer. It translates network messages into calls for local state changes.

"Validation"-specific (i.e. information relating to chainstate) data is maintained per-node using `CNodeState` instances.

Much of this region is `ProcessMessage()`: a giant conditional for rendering particular network message types to calls deeper into Bitcoin, e.g.

- `NetMsgType::BLOCK` -> `validation:ProcessNewBlock()`
- `NetMsgType::HEADERS` -> `validation:ProcessNewBlockHeaders()`
- ...

Peers are also penalized here based on the network messages they send (see `Misbehaving` and its usages).

代码分区：validation

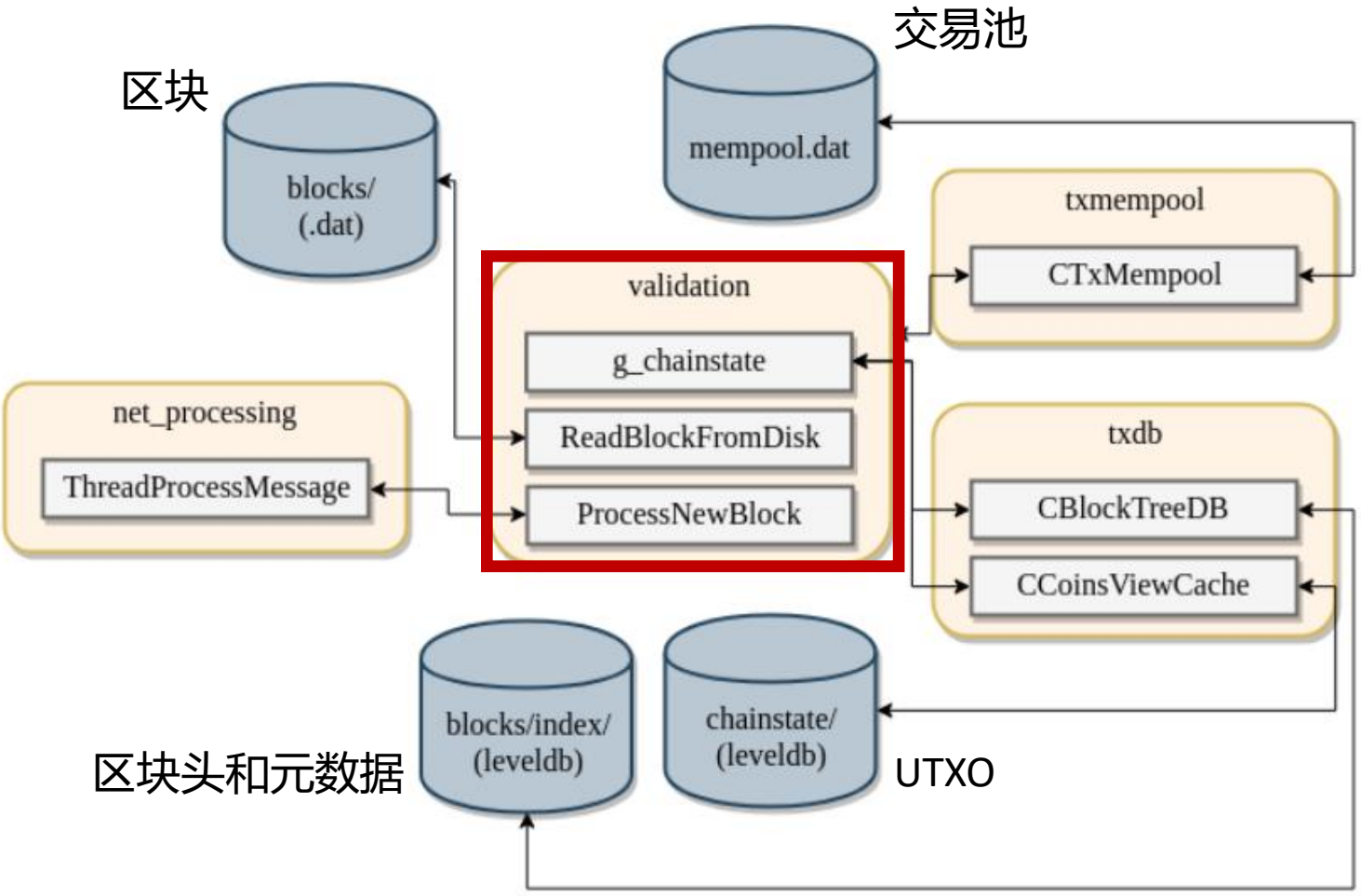
Regions > `validation.{h,cpp}`

`validation` handles modifying in-memory data structures for chainstate and transactions (mempool) on the basis of certain acceptance rules.

It both defines some of these data structures (`CChainState`, `mapBlockIndex`) as well as procedures for validating them, e.g. `CheckBlock()`.

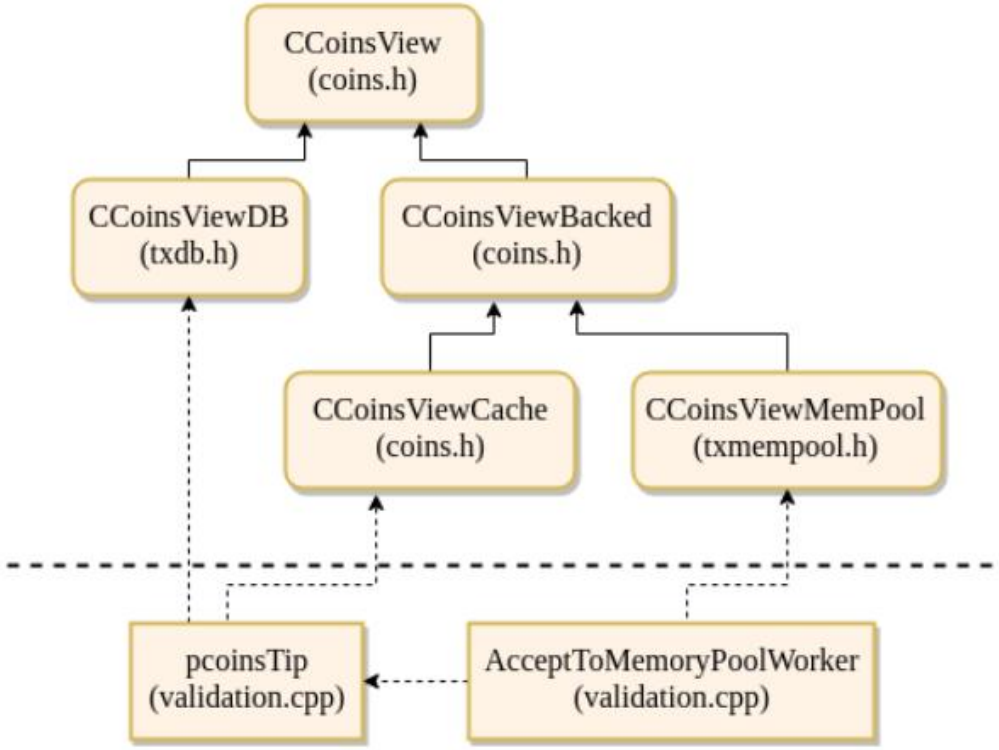
Oddly, it also contains some utility functions for marshalling data to and from disk, e.g. `ReadBlockFromDisk()`, `FlushStateToDisk()`, `{Dump,Load}Mempool()`. This is probably because `validation.{h,cpp}` is the result of refactoring `main.{h,cpp}` into smaller pieces.

代码分区：validation



代码分区：coins

CCoinsView 能够检查某个交易的输出是否能够对应到某个真实的 UTXO




```
/** Abstract view on the open txout
class CCoinsView
{
public:
    /** Retrieve the Coin (unspent transaction output) for a given outpoint.
     *  Returns true only when an unspent coin was found, which is returned in coin
     *  When false is returned, coin's value is unspecified.
     */
    virtual bool GetCoin(const COutPoint &outpoint, Coin &coin) const;

    //! Just check whether a given outpoint is unspent.
    virtual bool HaveCoin(const COutPoint &outpoint) const;

    //! Retrieve the block hash whose state this CCoinsView currently represents
    virtual uint256 GetBestBlock() const;

    ...
}
```

代码分区： script/

Regions > `script/`

The `script` subtree contains procedures for defining and executing Bitcoin scripts, as well as signing transactions (`script/sign.*`).

It also maintains data structures which cache script execution and signature verification (`script/sigcache.*`).

Script evaluation happens in `script/interpreter.cpp::EvalScript()`.

代码分区： consensus/

Regions > `consensus/`

Contains procedures for obviously consensus-critical actions like computing Merkle trees, checking transaction validity.

Contains chain validation parameters, e.g. `Params::BIP66Height, nMinimumChainWork`.

Defines BIP9 deployment description struct (`consensus/params.h:BIP9Deployment`).

`CValidationState` is defined in `consensus/validation.h` and is used broadly (but mostly in `validation`) as a small piece of state that tracks validity and likelihood of DoS when examining blocks.

代码分区： policy/

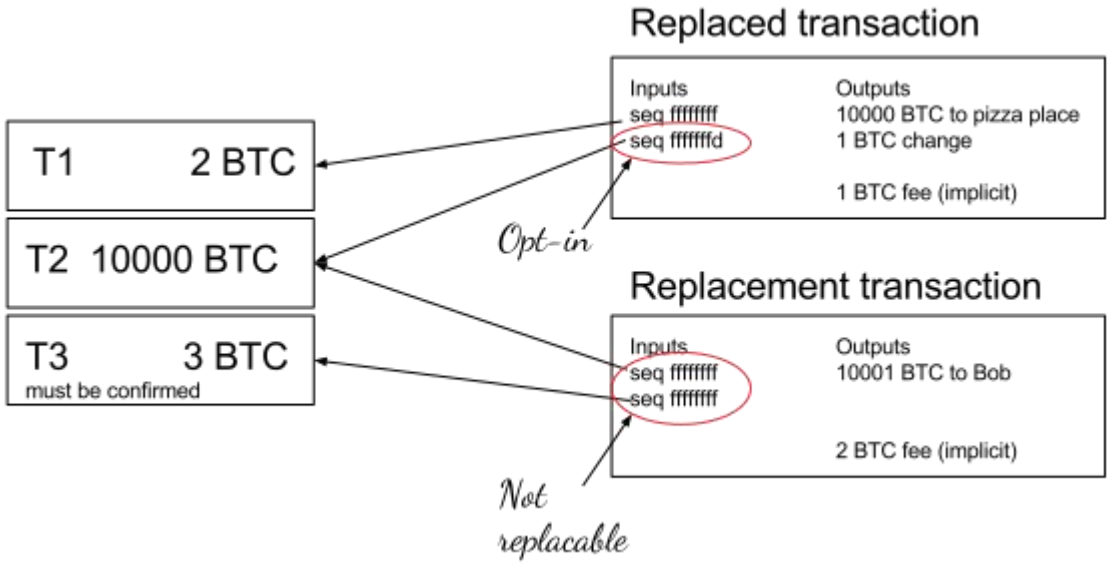
Regions > `policy/`

Policy contains logic for making various assessments about transactions (does this tx signal replace-by-fee?).

It contains logic for doing fee estimation (`policy/fees.*`).

Replace-by-fee (RBF)：

Any unconfirmed transaction can be replaced by another transaction that pays a bigger fee.



代码分区： interfaces/

Regions > `interfaces/`

Defines interfaces for interacting with the major subsystems in Bitcoin: node, wallet, GUI (eventually).

This is part of an ongoing effort by Russ Yanofsky to decompose the different parts of Bitcoin into separate systems that communicate using more formalized messages.

Eventually, we might be able to break Bitcoin Core into a few smaller repositories which can be maintained at different cadences.

代码分区： indexes/

Regions > `indexes/`

Contains optional indexes and a generic base class for adding more.

Currently only one index: `indexes/txindex` which provides a mapping of transaction ID to the `CDiskTxPos` for that transaction.

More indexes proposed, e.g. address to any related transactions (#14053 by @marcinja).

代码分区：wallet/

Regions > `wallet/`

Contains

- logic for marshalling wallet data to and from disk via BerkeleyDB.
- utilities for fee-bumping transactions.
- doing coin selection.
- RPC interface for the wallet.
- bookkeeping for wallet owners (`CWalletTx`, address book) .

代码分区：qt/

Regions > `qt/`

Contains all the code for doing the graphical user interface.

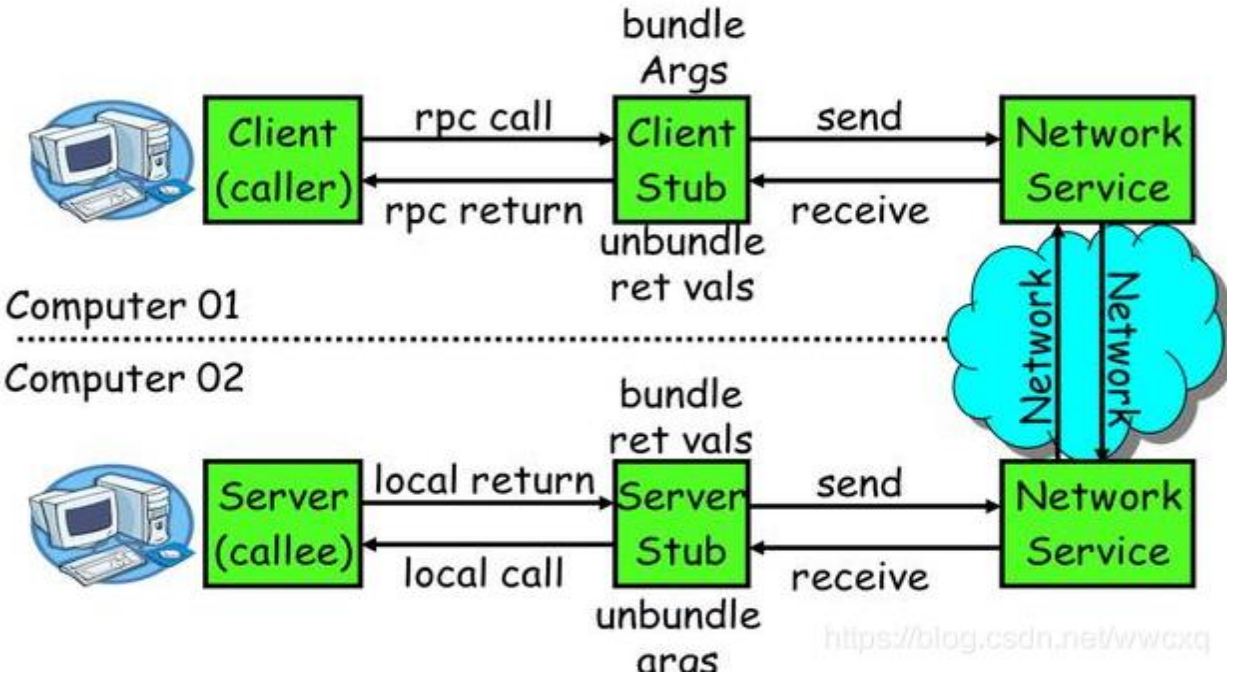
`qt/bitcoin.cpp:main()` is an alternate entrypoint for starting Bitcoin.

代码分区： rpc/

Regions > rpc/

Defines RPC interface and provides related utilities

RPC基本流程图：



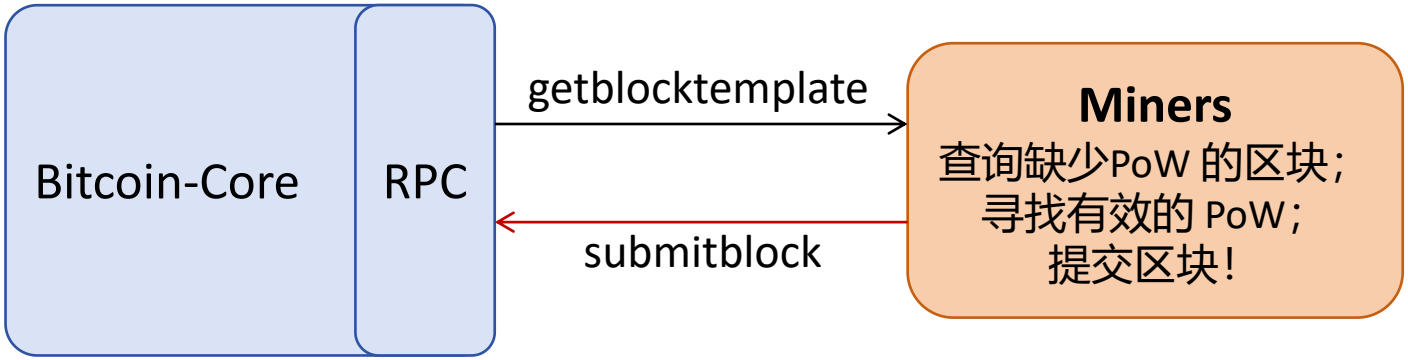
https://blog.csdn.net/wwcxq

代码分区： miner/

Regions > `miner.{h,cpp}`

Includes utilities for generating blocks to be mined (e.g. `BlockAssembler`). Used in conjunction with `rpc/mining.cpp` by miners:

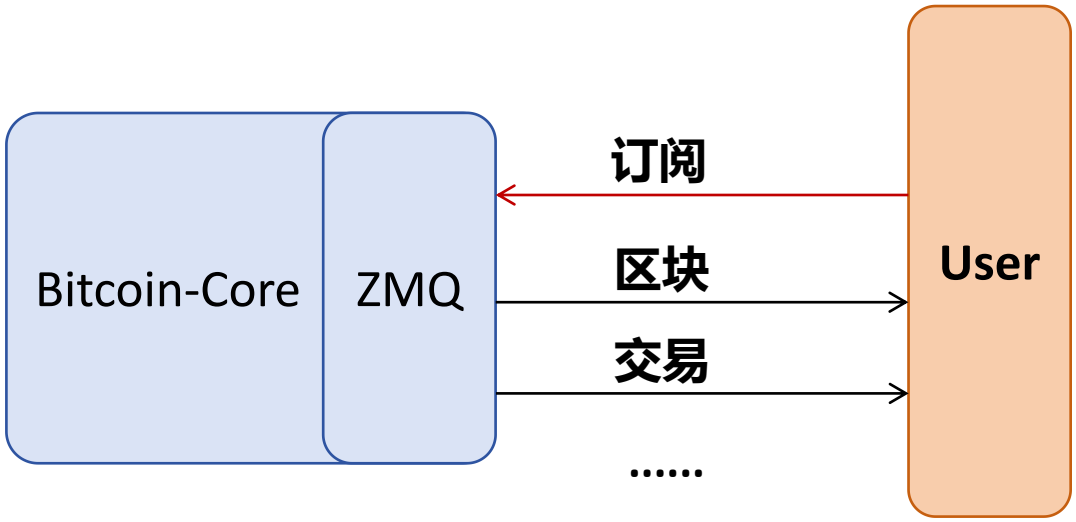
- `getblocktemplate`
- `submitblock`



代码分区： zmq/

Regions > `zmq/`

Registers events with `ValidationInterface` to forward on notifications about new blocks and transactions to ZMQ sockets.



Bitcoin Core：持久存储

存储目录

```
$ tree ~/.bitcoin/regtest/
```

```
├── banlist.dat
├── blocks
│   ├── blk00000.dat Block数据(磁盘文件)
│   ├── index Block元数据(levelDB)
│   │   ├── 000005.ldb
│   │   ├── 000006.log
│   │   ├── CURRENT
│   │   ├── LOCK
│   │   └── MANIFEST-000004
│   └── rev00000.dat
├── chainstate UTXO相关的数据(levelDB)
│   ├── 000005.ldb
│   ├── 000006.log
│   ├── CURRENT
│   ├── LOCK
│   └── MANIFEST-000004
├── debug.log
...
├── fee_estimates.dat
├── indexes
│   └── txindex
│       ├── 000003.log
│       ├── CURRENT
│       ├── LOCK
│       └── MANIFEST-000002
├── mempool.dat
├── peers.dat
└── wallets
    ├── db.log
    └── wallet.dat
```

.dat 文件存储序列化后的二进制数据结构；

序列化方法 => serialize.h

存储目录

- **`blocks/blk?????.dat`**: serialized block data
  - `validation.cpp:WriteBlockToDisk()`
  - `src/primitives/block.h:CBlock::SerializationOp()`
- **`blocks/rev?????.dat`**: "undo" data -- UTXOs added and removed by a block
  - `validation.cpp:UndoWriteToDisk()`
  - `src/undo.h:CTxUndo`

思考：为什么需要实现 rev**.dat？**

- **`mempool.dat`**: serialized list of mempool contents
  - `src/txmempool.cpp:CTxMemPool::infoAll()`
  - Dumped in `src/init.cpp:Shutdown()`
- **`peers.dat`**: serialized peers
  - `src/addrmah.h::CAddrMan::Serialize()`
- **`banlist.dat`**: banned node IPs/subnets
  - See `src/addrdb.cpp` for serialization details

LevelDB

Leveldb is a fast, sorted key value store used for a few things in Bitcoin.

It allows bulk writes and snapshots.

It is bundled with the source tree in `src/leveldb/` and maintained in bitcoin-core/leveldb.

- `blocks/index`: the complete tree of valid(ish) blocks the node has seen
  - Serializes `mapBlockIndex`, or a list of `CBlockIndex`es
  - `CBlockIndex` is a block header plus some important metadata, for example validation status of the block (`nStatus`) and position on disk (`nFile`/`nDataPos`)
- `chainstate/`: holds UTXO set
  - `COutPoint` -> `CCoinsCacheEntry`
    - Basically `(txid, index) -> Coin` (i.e. [CTxOut, is_coinbase, height])
  - `CCoinsViewCache::BatchWrite()`

Bitcoin Core：数据结构

区块相关的数据结构

Data structures > chainstate > blocks

`src/primitives/block.h:CBlockHeader` **区块头**

The block header attributes you know and love: `nVersion, hashPrevBlock, hashMerkleRoot, nTime, nBits, nNonce`.

`src/primitives/block.h:CBlock` **区块**

It's `CBlockHeader`, but with transactions attached.

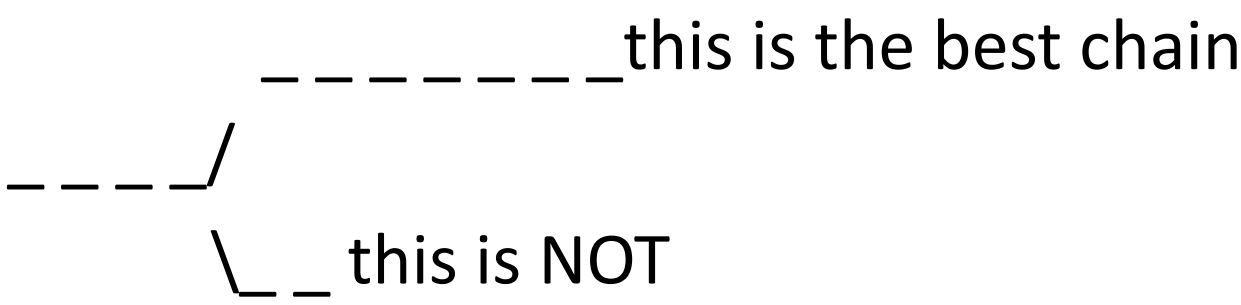
`src/chain.h:CBlockIndex` **区块索引**

A block header plus some important metadata, for example validation status of the block (`nStatus`) and position on disk (`nFile` / `nDataPos`)

The entire blockchain (and orphaned, invalid parts of the tree) is stored this way.

CChainState

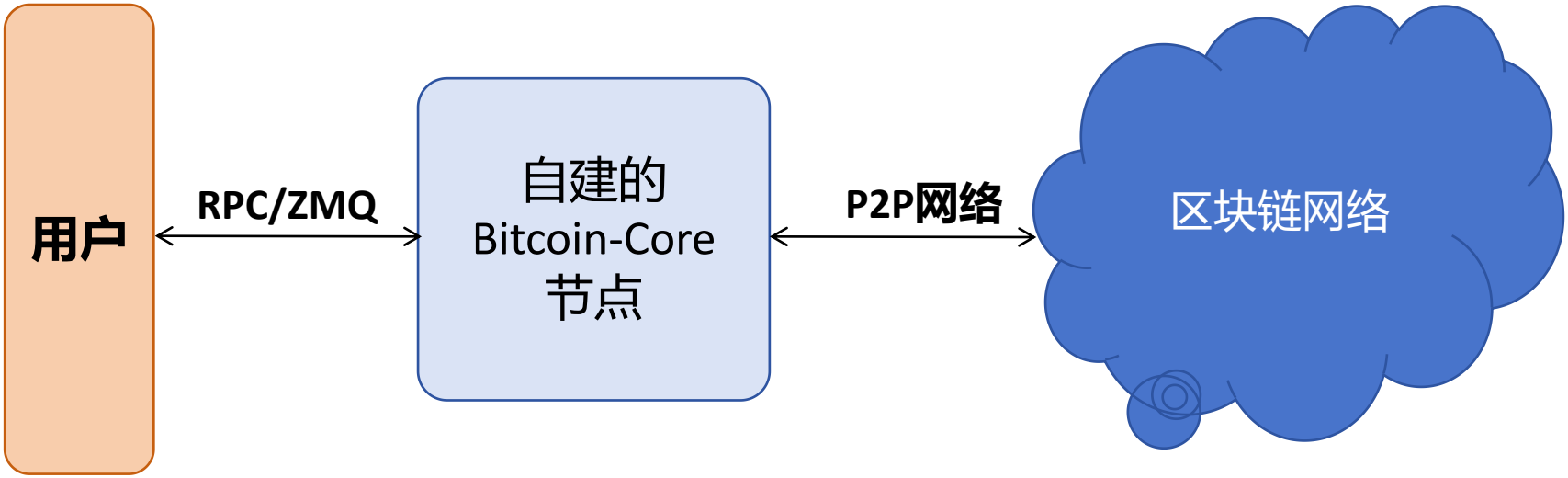
- validation.{h, cpp}中定义，维护本地视角下的 activeChain (最长的有效链)



区块链查询

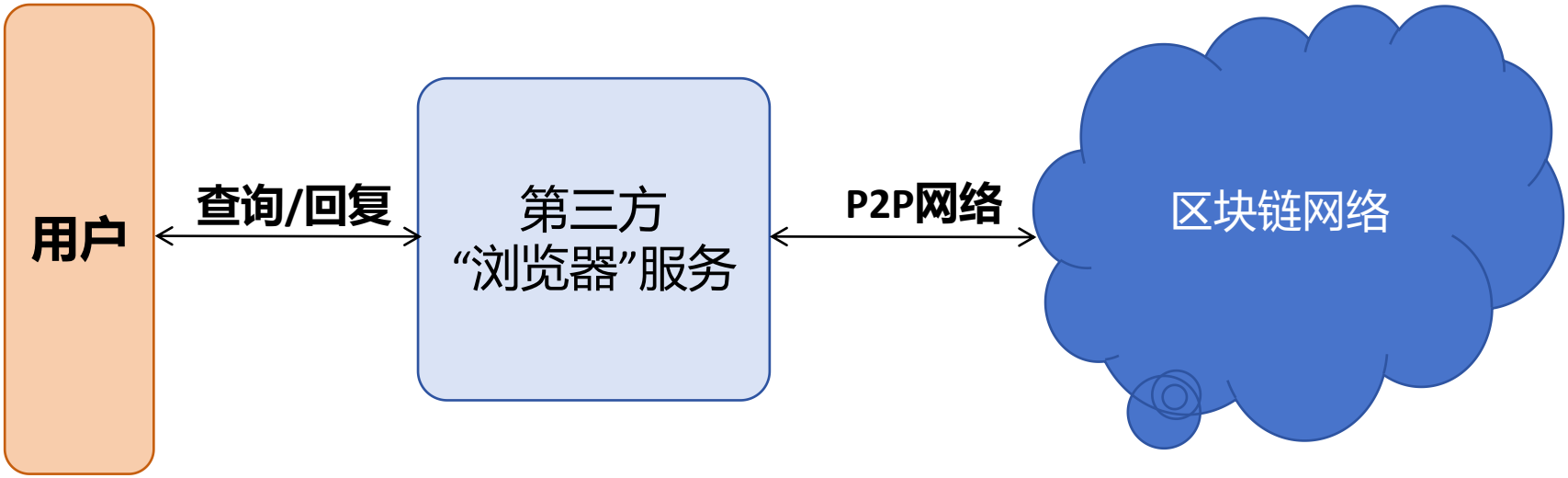
自建全节点

- 配置和运行自己的 Bitcoin Core 全节点
- 根据应用场景，开放 RPC 接口 或 ZMQ 接口



第三方“浏览器”

- 向第三方的比特币“浏览器”服务查询特定的数据



第三方“浏览器”

- 向第三方的比特币“浏览器”服务查询特定的数据

https://bitcoinexplorer.org/api/block/000000000000000001c8018d9cb3b742ef25114f27563e3fc4a1902167f9893


```
{
    "hash": "000000000000000001c8018d9cb3b742ef25114f27563e3fc4a1902167f9893",
    "confirmations": 363600,
    "height": 481824,
    "version": 536870914,
    "versionHex": "20000002",
    "merkleroot": "6438250cad442b982801ae6994edb8a9ec63c0a0ba117779fbe7ef7f07cad140",
    "time": 1503539857,
    "mediantime": 1503536701,
    "nonce": 575995682,
    "bits": "18013ce9",
    "difficulty": 888171856257.3206,
    "chainwork": "0000000000000000000000000000000000000000007eb6a652531c5ad6a4b8e9",
    "nTx": 1866,
    "previousblockhash": "000000000000000000cbeff0b533f8e1189cf09dfbebf57a8ebe349362811b80",
    "nextblockhash": "000000000000000000daf7a26d903543377d5cdddb962077e58fd11212479eea"
}
```

第三方“浏览器”

- 向第三方的比特币“浏览器”服务查询特定的数据
  - 以 Bitcoin Explorer 为例：
    - https://bitcoinexplorer.org/api/block/$HASH
    - https://bitcoinexplorer.org/api/block/$HEIGHT
    - https://bitcoinexplorer.org/api/block/header/$HASH
    - https://bitcoinexplorer.org/api/block/header/$HEIGHT
    - https://bitcoinexplorer.org/api/blocks/tip
    - https://bitcoinexplorer.org/api/tx/$TXID
    - https://bitcoinexplorer.org/api/tx/volume/24h

  ……

- 可以查询多个“浏览器”服务，缓解对单一服务过度信任的问题

轻客户端协议

- 传统 SPV 协议 ....
  - **Bad news:** Bitcoin-Core 没有 SPV 节点模式
  - **Good news:** 很多其他的 SPV 节点实现
  - 两类实现：
    - 从P2P网络，通过Bitcoin协议获得区块头 (https://github.com/cloudhead/nakamoto)
    - 从特定的服务器获得区块头 (https://github.com/spesmilo/electrum)

谢谢