### lotus 节点API

1. **chian**
2. ChainNotify返回带有链头更新的引用，第一条消息保证为len==1，type=='current'。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainNotify", "params": [], "id": 3}

返回：{

"jsonrpc": "2.0",

"result": {

"Cids": [

{

"/": "bafy2bzacednghy3w2e42xqrgeisswleti2uf2dno4ffaieuge2wfrbigqwdv6"

}

],

"Blocks": [

{

"Miner": "t00",

"Ticket": {

"VRFProof": "dnJmIHByb29mMDAwMDAwMHZyZiBwcm9vZjAwMDAwMDA="

},

"ElectionProof": {

"VRFProof": ""

},

"BeaconEntries": [

{

"Round": 0,

"Data": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="

}

],

"WinPoStProof": null,

"Parents": null,

"ParentWeight": "0",

"Height": 0,

"ParentStateRoot": {

"/": "bafy2bzaceabkxvqaj6q6u2bq4nzmwkk7c4buzj76yoek3yxfptaia6hstuxik"

},

"ParentMessageReceipts": {

"/": "bafy2bzaceaa43et73tgxsoh2xizd4mxhbrcfig4kqp25zfa5scdgkzppllyuu"

},

"Messages": {

"/": "bafy2bzacecgw6dqj4bctnbnyqfujltkwu7xc7ttaaato4i5miroxr4bayhfea"

},

"BLSAggregate": null,

"Timestamp": 1592524800,

"BlockSig": null,

"ForkSignaling": 0

}

],

"Height": 0

},

"id": 3

}

1. 返回当前链头

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainHead", "params": [], "id": 3}

3.ChainGetRandomness用于对链进行随机采样。

请求：{"jsonrpc":"2.0","method":"Filecoin.ChainGetRandomness","params": [tsk types.TipSetKey, personalization crypto.DomainSeparationTag, randEpoch abi.ChainEpoch, entropy []byte], "id": 3}

4.ChainGetBlock返回给定cid 对应的块。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetBlock", "params": [‘cid’], "id": 3}

5.ChainGetTipSet返回由给定的TipSetKey指定的tipset。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetTipSet", "params": [‘TipSetKey], "id": 3}

1. ChainGetBlockMessages 返回存储在指定区块的信息

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetBlockMessages", "params": [‘cid’], "id": 3}

7.ChainGetParentReceipts返回指定块的父提示集中消息的收据。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetParentReceipts", "params": [‘cid’], "id": 3}

8.ChainGetParentMessages返回存储在指定块的父提示集中的消息。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetParentMessages", "params": [‘cid’], "id": 3}

9.ChainGetTipSetByHeight在指定的epoch回过头来查找tipset。

//如果在指定的epoch没有块，则在较高的epoch处的tipset

//将被返回。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetTipSetByHeight", "params": [abi.ChainEpoch, types.TipSetKey], "id": 3}

10.ChainReadObj从chain读取指定CID引用的ipld节点并返回原始字节。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainReadObj", "params": [cid.Cid], "id": 3}

11.ChainHasObj检查链块存储中是否存在给定的CID。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainHasObj", "params": [cid.Cid,cid.Cid], "id": 3}

12.ChainSetHead强制设置当前链头。小心使用。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainSetHead", "params": [types.TipSetKey], "id": 3}

13.ChainGetGenesis返回genesis tipset。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetGenesis", "params": [], "id": 3}

14.ChainTipSetWeight计算指定types.TipSetKey的权重

15.ChainGetNode计算指定types.TipSetKey的权重

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainTipSetWeight", "params": [types.TipSetKey], "id": 3}

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetNode", "params": [string], "id": 3}

16.ChainGetMessage从链区块库读取指定CID引用的消息。

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetMessage", "params": [cid.Cid], "id": 3}

17.// ChainGetPath returns a set of revert/apply operations needed to get from

    // one tipset to another, for example:

    //```

    //        to

    //         ^

    // from   tAA

    //   ^     ^

    // tBA    tAB

    //  ^---\*--^

    //      ^

    //     tRR

    //```

    // Would return `[revert(tBA), apply(tAB), apply(tAA)]`

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainGetPath", "params": [from types.TipSetKey,  to  types.TipSetKey], "id": 3}

18.ChainExport returns a stream of bytes with CAR dump of chain data.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.ChainExport", "params": [types.TipSetKey], "id": 3}

**二、sync**

// MethodGroup: Sync

// The Sync method group contains methods for interacting with and

// observing the lotus sync service.

1. SyncState returns the current status of the lotus sync system.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.SyncState ", "params": [], "id": 3}

2.// SyncSubmitBlock can be used to submit a newly created block to the. network through this node

请求：{ "jsonrpc": "2.0", "method": "Filecoin.SyncSubmitBlock", "params": [blk \*types.BlockMsg

], "id": 3}

3.// SyncMarkBad marks a blocks as bad, meaning that it won't ever by synced. Use with extreme caution.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.SyncMarkBad", "params": [cid.Cid], "id": 3}

4.// SyncCheckBad checks if a block was marked as bad, and if it was, returns the reason.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.SyncCheckBad", "params": [cid.Cid], "id": 3}

5.// SyncIncomingBlocks returns a channel streaming incoming, potentially not yet synced block headers.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.SyncIncomingBlocks", "params": [cid.Cid], "id": 3}

**三、Mpool**

// MethodGroup: Mpool

    // The Mpool methods are for interacting with the message pool. The message pool

    // manages all incoming and outgoing 'messages' going over the network.

1.MpoolPending returns pending mempool messages.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.MpoolPending", "params": [types.TipSetKey

], "id": 3}

2.MpoolPush pushes a signed message to mempool

请求：{ "jsonrpc": "2.0", "method": "Filecoin.MpoolPush", "params": [\*types.SignedMessage], "id": 3}

3.// MpoolPushMessage atomically assigns a nonce, signs, and pushes a message to mempool.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.MpoolPushMessage", "params": [\*types.Message], "id": 3}

4.// MpoolGetNonce gets next nonce for the specified sender.

// Note that this method may not be atomic. Use MpoolPushMessage instead.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.MpoolGetNonce", "params": [address], "id": 3}

请求：{ "jsonrpc": "2.0", "method": "Filecoin.MpoolSub", "params": [], "id": 3}

5.// MpoolEstimateGasPrice estimates what gas price should be used for a

// message to have high likelihood of inclusion in `nblocksincl` epochs.

请求：{ "jsonrpc": "2.0", "method": "Filecoin.MpoolEstimateGasPrice", "params": [nblocksincl uint64, sender address.Address, gaslimit int64, tsk types.TipSetKey], "id": 3}

四、MethodGroup: Miner

请求：{"jsonrpc":"2.0","method":"Filecoin.MinerGetBaseInfo","params": [address.Address, abi.ChainEpoch, types.TipSetKey], "id": 3}

请求：{ "jsonrpc": "2.0", "method": "Filecoin.MinerCreateBlock", "params": [\*BlockTemplate], "id": 3}

**四、wallet**

1.// WalletNew creates a new address in the wallet with the given sigType.

    WalletNew(context.Context, crypto.SigType) (address.Address, error)

请求：{ "jsonrpc": "2.0", "method": "Filecoin.WalletNew ", "params": [crypto.SigType], "id": 3}

2. // WalletHas indicates whether the given address is in the wallet.

    WalletHas(context.Context, address.Address) (bool, error)

请求：{ "jsonrpc": "2.0", "method": "Filecoin.WalletHas", "params": [address.Address], "id": 3}

3. // WalletList lists all the addresses in the wallet.

WalletList(context.Context) ([]address.Address, error)

请求：{ "jsonrpc": "2.0", "method": "Filecoin.WalletList ", "params": [], "id": 3}

4. // WalletBalance returns the balance of the given address at the current head of the chain.

WalletBalance(context.Context, address.Address) (types.BigInt, error)

请求：{ "jsonrpc": "2.0", "method": "Filecoin.WalletBalance", "params": [address.Address], "id": 3}

5. // WalletSign signs the given bytes using the given address.

    WalletSign(context.Context, address.Address, []byte) (\*crypto.Signature, error)

 请求：{ "jsonrpc": "2.0", "method": "Filecoin.WalletSign", "params": [address.Address, []byte], "id": 3}

  6. // WalletSignMessage signs the given message using the given address.

    WalletSignMessage(context.Context, address.Address, \*types.Message) (\*types.SignedMessage, error)

   请求：{ "jsonrpc": "2.0", "method": "Filecoin.WalletSignMessage", "params": [address.Address, \*types.Message], "id": 3}

7.  // WalletVerify takes an address, a signature, and some bytes, and indicates whether the signature is valid.

    // The address does not have to be in the wallet.

    WalletVerify(context.Context, address.Address, []byte, \*crypto.Signature) bool

请求：{"jsonrpc":"2.0","method":"Filecoin.WalletVerify","params": [address.Address, []byte, \*crypto.Signature], "id": 3}

 8.// WalletDefaultAddress returns the address marked as default in the wallet.

    WalletDefaultAddress(context.Context) (address.Address, error)

   请求：{"jsonrpc":"2.0","method":"Filecoin.WalletDefaultAddress","params": [], "id": 3}

9. // WalletSetDefault marks the given address as as the default one.

    WalletSetDefault(context.Context, address.Address) error

    请求：{"jsonrpc":"2.0","method":"Filecoin.WalletSetDefault","params": [address.Address], "id": 3}

10.// WalletExport returns the private key of an address in the wallet.

    WalletExport(context.Context, address.Address) (\*types.KeyInfo, error)

   请求：{"jsonrpc":"2.0","method":"Filecoin.WalletExport","params": [address.Address], "id": 3}

11. // WalletImport receives a KeyInfo, which includes a private key, and imports it into the wallet.

    WalletImport(context.Context, \*types.KeyInfo) (address.Address, error)

   请求：{"jsonrpc":"2.0","method":"Filecoin.WalletImport","params": [\*types.KeyInfo], "id": 3}

   12.// WalletDelete deletes an address from the wallet.

    WalletDelete(context.Context, address.Address) error

请求：{"jsonrpc":"2.0","method":"Filecoin.WalletDelete","params": [address.Address], "id": 3}

**五、 Other**

    // MethodGroup: Client

    // The Client methods all have to do with interacting with the storage and

    // retrieval markets as a client

    // ClientImport imports file under the specified path into filestore.

ClientImport(ctx context.Context, ref FileRef) (cid.Cid, error)

    // ClientStartDeal proposes a deal with a miner.

ClientStartDeal(ctx context.Context, params \*StartDealParams) (\*cid.Cid, error)

    // ClientGetDealInfo returns the latest information about a given deal.

ClientGetDealInfo(context.Context, cid.Cid) (\*DealInfo, error)

    // ClientListDeals returns information about the deals made by the local client.

ClientListDeals(ctx context.Context) ([]DealInfo, error)

    // ClientHasLocal indicates whether a certain CID is locally stored.

ClientHasLocal(ctx context.Context, root cid.Cid) (bool, error)

    // ClientFindData identifies peers that have a certain file, and returns QueryOffers (one per peer).

ClientFindData(ctx context.Context, root cid.Cid) ([]QueryOffer, error)

    // ClientMinerQueryOffer returns a QueryOffer for the specific miner and file.

ClientMinerQueryOffer(ctx context.Context, root cid.Cid, miner address.Address) (QueryOffer, error)

    // ClientRetrieve initiates the retrieval of a file, as specified in the order.

ClientRetrieve(ctx context.Context, order RetrievalOrder, ref \*FileRef) error

    // ClientQueryAsk returns a signed StorageAsk from the specified miner.

ClientQueryAsk(ctx context.Context, p peer.ID, miner address.Address) (\*storagemarket.SignedStorageAsk, error)

    // ClientCalcCommP calculates the CommP for a specified file, based on the sector size of the provided miner.

ClientCalcCommP(ctx context.Context, inpath string, miner address.Address) (\*CommPRet, error)

    // ClientGenCar generates a CAR file for the specified file.

    ClientGenCar(ctx context.Context, ref FileRef, outpath string) error

    // ClientUnimport removes references to the specified file from filestore

    //ClientUnimport(path string)

    // ClientListImports lists imported files and their root CIDs

    ClientListImports(ctx context.Context) ([]Import, error)

    //ClientListAsks() []Ask

**六、MethodGroup: State**

    // The State methods are used to query, inspect, and interact with chain state.

    // All methods take a TipSetKey as a parameter. The state looked up is the state at that tipset.

    // A nil TipSetKey can be provided as a param, this will cause the heaviest tipset in the chain to be used.

    // StateCall runs the given message and returns its result without any persisted changes.

StateCall(context.Context, \*types.Message, types.TipSetKey) (\*InvocResult, error)

    // StateReplay returns the result of executing the indicated message, assuming it was executed in the indicated tipset.

StateReplay(context.Context, types.TipSetKey, cid.Cid) (\*InvocResult, error)

    // StateGetActor returns the indicated actor's nonce and balance.

StateGetActor(ctx context.Context, actor address.Address, tsk types.TipSetKey) (\*types.Actor, error)

    // StateReadState returns the indicated actor's state.

StateReadState(ctx context.Context, actor address.Address, tsk types.TipSetKey) (\*ActorState, error)

    // StateListMessages looks back and returns all messages with a matching to or from address, stopping at the given height.

    StateListMessages(ctx context.Context, match \*types.Message, tsk types.TipSetKey, toht abi.ChainEpoch) ([]cid.Cid, error)

    // StateNetworkName returns the name of the network the node is synced to

StateNetworkName(context.Context) (dtypes.NetworkName, error)

    // StateMinerSectors returns info about the given miner's sectors. If the filter bitfield is nil, all sectors are included.

    // If the filterOut boolean is set to true, any sectors in the filter are excluded.

    // If false, only those sectors in the filter are included.

StateMinerSectors(context.Context, address.Address, \*abi.BitField, bool, types.TipSetKey) ([]\*ChainSectorInfo, error)

    // StateMinerProvingSet returns info about those sectors that a given miner is actively proving.

StateMinerProvingSet(context.Context, address.Address, types.TipSetKey) ([]\*ChainSectorInfo, error)

    // StateMinerProvingDeadline calculates the deadline at some epoch for a proving period

    // and returns the deadline-related calculations.

StateMinerProvingDeadline(context.Context, address.Address, types.TipSetKey) (\*miner.DeadlineInfo, error)

    // StateMinerPower returns the power of the indicated miner

StateMinerPower(context.Context, address.Address, types.TipSetKey) (\*MinerPower, error)

    // StateMinerInfo returns info about the indicated miner

StateMinerInfo(context.Context, address.Address, types.TipSetKey) (MinerInfo, error)

    // StateMinerDeadlines returns all the proving deadlines for the given miner

StateMinerDeadlines(context.Context, address.Address, types.TipSetKey) (\*miner.Deadlines, error)

    // StateMinerFaults returns a bitfield indicating the faulty sectors of the given miner

StateMinerFaults(context.Context, address.Address, types.TipSetKey) (\*abi.BitField, error)

    // StateAllMinerFaults returns all non-expired Faults that occur within lookback epochs of the given tipset

StateAllMinerFaults(ctx context.Context, lookback abi.ChainEpoch, ts types.TipSetKey) ([]\*Fault, error)

    // StateMinerRecoveries returns a bitfield indicating the recovering sectors of the given miner

StateMinerRecoveries(context.Context, address.Address, types.TipSetKey) (\*abi.BitField, error)

    // StateMinerInitialPledgeCollateral returns the initial pledge collateral for the specified miner's sector

StateMinerInitialPledgeCollateral(context.Context, address.Address, abi.SectorNumber, types.TipSetKey) (types.BigInt, error)

    // StateMinerAvailableBalance returns the portion of a miner's balance that can be withdrawn or spent

StateMinerAvailableBalance(context.Context, address.Address, types.TipSetKey) (types.BigInt, error)

    // StateSectorPreCommitInfo returns the PreCommit info for the specified miner's sector

StateSectorPreCommitInfo(context.Context, address.Address, abi.SectorNumber, types.TipSetKey) (miner.SectorPreCommitOnChainInfo, error)

    // StateSectorGetInfo returns the on-chain info for the specified miner's sector

    StateSectorGetInfo(context.Context, address.Address, abi.SectorNumber, types.TipSetKey) (\*miner.SectorOnChainInfo, error)

StatePledgeCollateral(context.Context, types.TipSetKey) (types.BigInt, error)

    // StateSearchMsg searches for a message in the chain, and returns its receipt and the tipset where it was executed

StateSearchMsg(context.Context, cid.Cid) (\*MsgLookup, error)

    // StateWaitMsg looks back in the chain for a message. If not found, it blocks until the

    // message arrives on chain, and gets to the indicated confidence depth.

StateWaitMsg(ctx context.Context, cid cid.Cid, confidence uint64) (\*MsgLookup, error)

    // StateListMiners returns the addresses of every miner that has claimed power in the Power Actor

StateListMiners(context.Context, types.TipSetKey) ([]address.Address, error)

    // StateListActors returns the addresses of every actor in the state

StateListActors(context.Context, types.TipSetKey) ([]address.Address, error)

    // StateMarketBalance looks up the Escrow and Locked balances of the given address in the Storage Market

StateMarketBalance(context.Context, address.Address, types.TipSetKey) (MarketBalance, error)

    // StateMarketParticipants returns the Escrow and Locked balances of every participant in the Storage Market

StateMarketParticipants(context.Context, types.TipSetKey) (map[string]MarketBalance, error)

    // StateMarketDeals returns information about every deal in the Storage Market

StateMarketDeals(context.Context, types.TipSetKey) (map[string]MarketDeal, error)

    // StateMarketStorageDeal returns information about the indicated deal

StateMarketStorageDeal(context.Context, abi.DealID, types.TipSetKey) (\*MarketDeal, error)

    // StateLookupID retrieves the ID address of the given address

StateLookupID(context.Context, address.Address, types.TipSetKey) (address.Address, error)

    // StateAccountKey returns the public key address of the given ID address

StateAccountKey(context.Context, address.Address, types.TipSetKey) (address.Address, error)

    // StateChangedActors returns all the actors whose states change between the two given state CIDs

    // TODO: Should this take tipset keys instead?

StateChangedActors(context.Context, cid.Cid, cid.Cid) (map[string]types.Actor, error)

    // StateGetReceipt returns the message receipt for the given message

StateGetReceipt(context.Context, cid.Cid, types.TipSetKey) (\*types.MessageReceipt, error)

    // StateMinerSectorCount returns the number of sectors in a miner's sector set and proving set

StateMinerSectorCount(context.Context, address.Address, types.TipSetKey) (MinerSectors, error)

    // StateCompute is a flexible command that applies the given messages on the given tipset.

    // The messages are run as though the VM were at the provided height.

    StateCompute(context.Context, abi.ChainEpoch, []\*types.Message, types.TipSetKey) (\*ComputeStateOutput, error)

**七、 MethodGroup: Msig**

    // The Msig methods are used to interact with multisig wallets on the

    // filecoin network

    // MsigGetAvailableBalance returns the portion of a multisig's balance that can be withdrawn or spent

MsigGetAvailableBalance(context.Context, address.Address, types.TipSetKey) (types.BigInt, error)

    // MsigCreate creates a multisig wallet

    // It takes the following params: <required number of senders>, <approving addresses>, <unlock duration>

    //<initial balance>, <sender address of the create msg>, <gas price>

MsigCreate(context.Context, int64, []address.Address, abi.ChainEpoch, types.BigInt, address.Address, types.BigInt) (cid.Cid, error)

    // MsigPropose proposes a multisig message

    // It takes the following params: <multisig address>, <recipient address>, <value to transfer>,

    // <sender address of the propose msg>, <method to call in the proposed message>, <params to include in the proposed message>

MsigPropose(context.Context, address.Address, address.Address, types.BigInt, address.Address, uint64, []byte) (cid.Cid, error)

    // MsigApprove approves a previously-proposed multisig message

    // It takes the following params: <multisig address>, <proposed message ID>, <proposer address>, <recipient address>, <value to transfer>,

    // <sender address of the approve msg>, <method to call in the proposed message>, <params to include in the proposed message>

MsigApprove(context.Context, address.Address, uint64, address.Address, address.Address, types.BigInt, address.Address, uint64, []byte) (cid.Cid, error)

    // MsigCancel cancels a previously-proposed multisig message

    // It takes the following params: <multisig address>, <proposed message ID>, <recipient address>, <value to transfer>,

    // <sender address of the cancel msg>, <method to call in the proposed message>, <params to include in the proposed message>

MsigCancel(context.Context, address.Address, uint64, address.Address, types.BigInt, address.Address, uint64, []byte) (cid.Cid, error)

    // MsigSwapPropose proposes swapping 2 signers in the multisig

    // It takes the following params: <multisig address>, <sender address of the propose msg>,

    // <old signer> <new signer>

MsigSwapPropose(context.Context, address.Address, address.Address, address.Address, address.Address) (cid.Cid, error)

    // MsigSwapApprove approves a previously proposed SwapSigner

    // It takes the following params: <multisig address>, <sender address of the approve msg>, <proposed message ID>,

    // <proposer address>, <old signer> <new signer>

MsigSwapApprove(context.Context, address.Address, address.Address, uint64, address.Address, address.Address, address.Address) (cid.Cid, error)

    // MsigSwapCancel cancels a previously proposed SwapSigner message

    // It takes the following params: <multisig address>, <sender address of the cancel msg>, <proposed message ID>,

    // <old signer> <new signer>

    MsigSwapCancel(context.Context, address.Address, address.Address, uint64, address.Address, address.Address) (cid.Cid, error)

    MarketEnsureAvailable(context.Context, address.Address, address.Address, types.BigInt) (cid.Cid, error)

    // MarketFreeBalance

**八、 MethodGroup: Paych**

    // The Paych methods are for interacting with and managing payment channels

PaychGet(ctx context.Context, from, to address.Address, ensureFunds types.BigInt) (\*ChannelInfo, error)

PaychList(context.Context) ([]address.Address, error)

PaychStatus(context.Context, address.Address) (\*PaychStatus, error)

PaychClose(context.Context, address.Address) (cid.Cid, error)

PaychAllocateLane(ctx context.Context, ch address.Address) (uint64, error)

PaychNewPayment(ctx context.Context, from, to address.Address, vouchers []VoucherSpec) (\*PaymentInfo, error)

PaychVoucherCheckValid(context.Context, address.Address, \*paych.SignedVoucher) error

PaychVoucherCheckSpendable(context.Context, address.Address, \*paych.SignedVoucher, []byte, []byte) (bool, error)

PaychVoucherCreate(context.Context, address.Address, types.BigInt, uint64) (\*paych.SignedVoucher, error)

PaychVoucherAdd(context.Context, address.Address, \*paych.SignedVoucher, []byte, types.BigInt) (types.BigInt, error)

PaychVoucherList(context.Context, address.Address) ([]\*paych.SignedVoucher, error)

    PaychVoucherSubmit(context.Context, address.Address, \*paych.SignedVoucher) (cid.Cid, error)