**wanchain-crosschain**

cross\_send

sendTransaction.js: The class can create/send normal/lock/refund/revoke Transaction.

dbDefine

crossTransDefind.js: The db schema for norTransaction and crossTransaction.

wanchaindb: The wanchain-db operation library.

wanchainsender

sendGroup

SendFromSocket.js: Send request to api server throught socket.

SendFromWeb3.js: Send request to web3 Ipc.

webSocket

socketmessage.js: The Class socketmessage which handle socker response.

socketServer.js: The socketServer class initialized with socket url.

wanchaintrans

contract: IContract is one Class handle interface of contract.

cross\_contract: The Class hashContract extends IContract, initialized with wan\_crosschain\_contract abi. X and hashX will be generated through here.

cross\_transactions: The folder included superclass hashXSend and subclass of eth and wan which handle the contract transaction send.

interface: This folder include the amount and RawTransaction interface.

web3: To init the web3.

webSocket

messageFactory.js: Wan\_api server request interface.

ccUtil.js: The primary Class which initialized with configuration file and include those lock/refund/revoke transaction operation and event detection.

config.js: The configuration file which include contract address/functionname/abi/database configuration, etc.

monitor.js: The monitorRecord task will monitor the transaciton and update it's status.

walletCore.js: The walletCore Class will initialize the socket/db and get up a interval task monitorTask.

**File: ccUtil.js**

This file provides the chain data normal api and the cross-chain transaction api.

**async init(cfg,ethsender, wansender,cb){}**

**Parameters**:

cfg: The configuraion file.

ethsender: A valid ethSender object.

wansender: A valid wanSender object.

cb: Callback function

**async createrSender(ChainType, useWeb3=false){}**

**Parameters**:

ChainType: sender chain type, 'ETH' or 'WAN'

useWeb3: default false, if use web3 sender, plz input true.

**Returns**:

return the sender object.

**async getEthAccountsInfo(sender) {}**

**Parameters**:

sender: A valid send object, ethsender

**Returns**:

return the whole eth accounts info include balance in the eth keystore path

**createEthAddr(keyPassword){}**

**Parameters**:

keyPassword: The account password

**Returns**:

return the new eth account address, and also create the keystore file in the keystorepath in config

**async getWanAccountsInfo(sender) {}**

**Parameters**:

sender: A valid send object, wansender

**Returns**:

return the whole wan accounts info include balance in the eth keystore path

**createWanAddr(keyPassword) {}**

**Parameters**:

keyPassword: The account password

**Returns**:

return the new wan account address, and also create the keystore file in the keystorepath in config

**createTrans(sender){}**

**Parameters**:

sender: A valid send object, ethsender or wansender

**Returns**:

return Object - the sendTransaction

**getEthSmgList(sender) {}**

**Parameters**:

sender: A valid send object, ethsender

**Returns**:

Promise return Object - the storemangroup list

**async sendEthHash(sender, tx) {}**

**Parameters**:

sender: A valid send object, ethsender

tx: Object transaction, include those follow keys(from, amount, storemanGroup, cross, gas, gasprice, nonce)

**Returns**:

return the txhash of the transaction

**async sendDepositX(sender, from,gas,gasPrice,x, passwd, nonce) {}**

**Parameters**:

sender: A valid send object, ethsender

from: An address for the sending account

gas: The amount of gas to use for the transaction

gasPrice: The price of gas for this transaction in wei

x: 32 bytes hash, which stand for the unique identification x of each cross transaction

passwd: the password of the sending account

nonce: The number of transactions made by the sender prior to this one

**Returns**:

return the txhash of the transaction

**async sendEthCancel(sender, from,gas,gasPrice,x, passwd, nonce) {}**

**Parameters**:

sender: A valid send object, ethsender

from: An address for the sending account

gas: The amount of gas to use for the transaction

gasPrice: The price of gas for this transaction in wei

x: 32 bytes hash, which stand for the unique identification x of each cross transaction

passwd: the password of the sending account

nonce: The number of transactions made by the sender prior to this one

**Returns**:

return the txhash of the transaction

**async sendWanHash(sender, tx) {}**

**Parameters**:

sender: A valid send object, wansender

tx: Object transaction, include those follow keys(from, amount, storemanGroup, cross, gas, gasprice, nonce)

**Returns**:

return the txhash of the transaction

**async sendWanX(sender, from,gas,gasPrice,x, passwd, nonce) {}**

**Parameters**:

sender: A valid send object, wansender

from: An address for the sending account

gas: The amount of gas to use for the transaction

gasPrice: The price of gas for this transaction in wei

x: 32 bytes hash, which stand for the unique identification x of each cross transaction

passwd: the password of the sending account

nonce: The number of transactions made by the sender prior to this one

**Returns**:

return the txhash of the transaction

**async sendWanCancel(sender, from,gas,gasPrice,x, passwd,nonce) {}**

**Parameters**:

sender: A valid send object, wansender

from: An address for the sending account

gas: The amount of gas to use for the transaction

gasPrice: The price of gas for this transaction in wei

x: 32 bytes hash, which stand for the unique identification x of each cross transaction

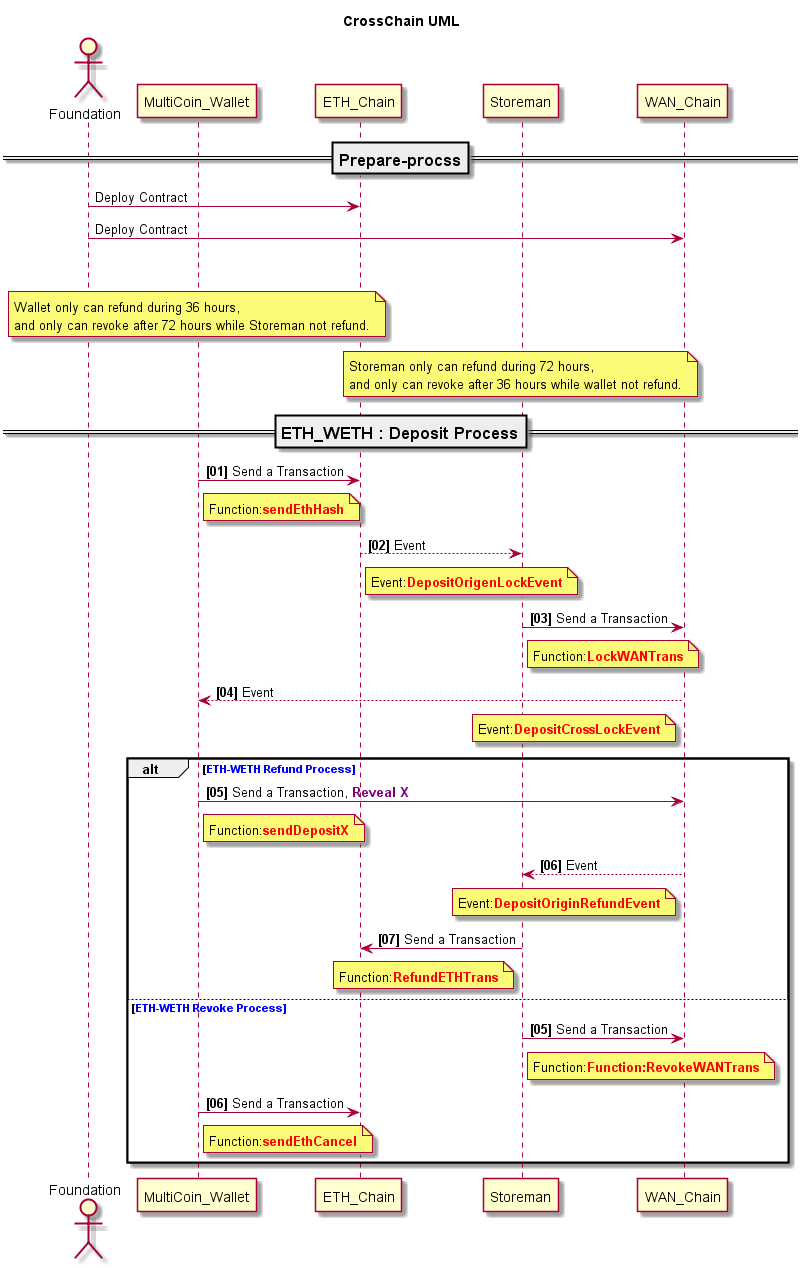
passwd: the password of the sending account

nonce: The number of transactions made by the sender prior to this one

**Returns**:

return the txhash of the transaction

**There are two transaction-direction, depoist means ETH to WETH, withdraw means WETH to ETH.**



**getDepositOrigenLockEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, ethsender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the event log of the original deposit lock transaction, the deposit lock transaction is on eth with the direction ETH-WETH

**getDepositCrossLockEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, wansender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the event log of the cross deposit lock transaction of storeman, the deposit lock transaction of storeman is on wan with the direction ETH-WETH

**getDepositOriginRefundEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, wansender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the event log of the original deposit refund transaction, the deposit refund transaction is on wan with the direction ETH-WETH

**getDepositRevokeEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, ethsender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the event log of the original deposit revoke transaction, the deposit revoke transaction is on eth with the direction ETH-WETH

**getDepositHTLCLeftLockedTime(sender, hashX){}**

**Parameters**:

sender: A valid send object, ethsender or wansender

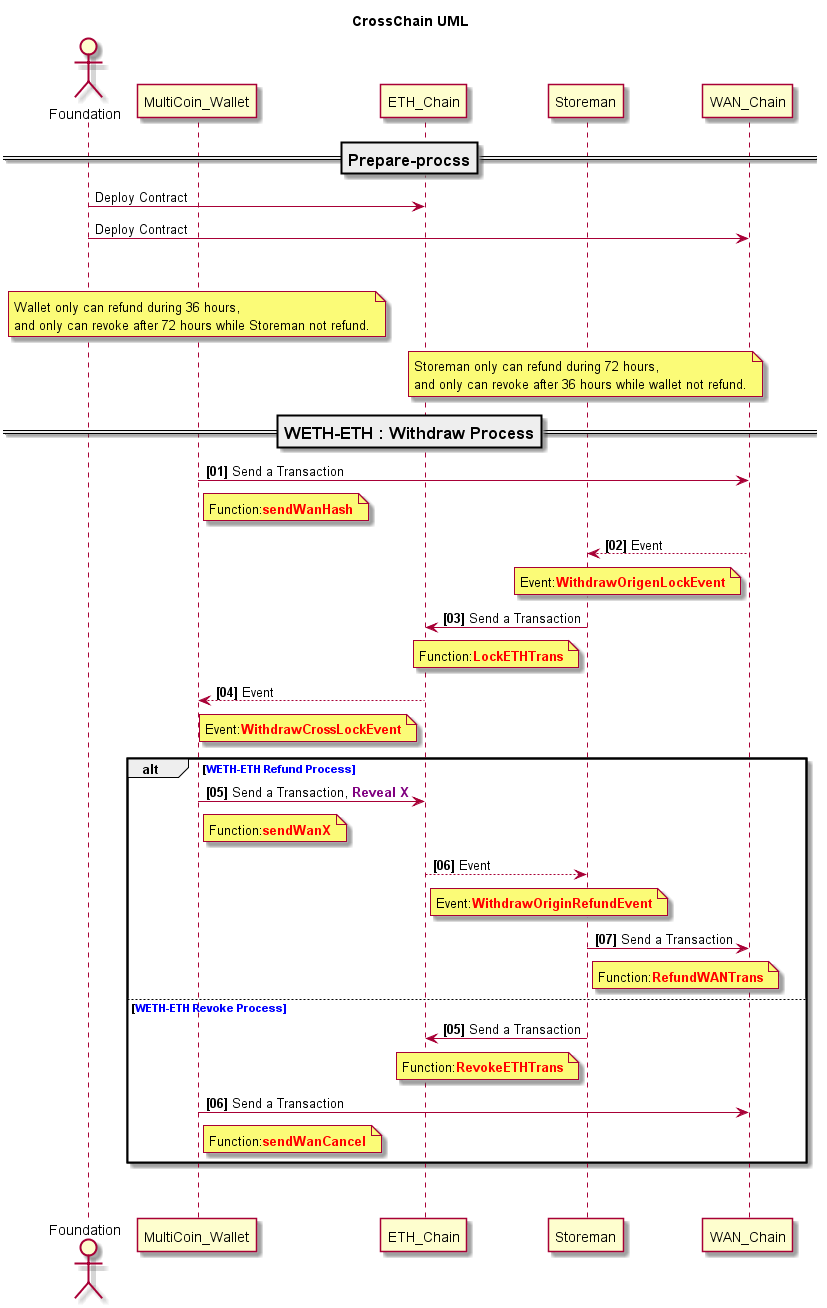
hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the deposit HTLC left locked time, the unit is second, after this revoke can be done.

If the sender is ethsender, this time is about the user deposit HTLC left locker time;

If the sender is wansender, this time is about the storeman deposit HTLC left locker time



**getWithdrawOrigenLockEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, wansender

hashX: 32 by

tes hash of X

**Returns**:

Promise return Object - the event log of the original withdraw transaction, the withdraw transaction is on wan with the direction WETH-ETH

**getWithdrawCrossLockEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, ethsender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the event log of the cross withdraw transaction of storeman, the withdraw transaction of storeman is on eth with the direction WETH-ETH

**getWithdrawOriginRefundEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, ethsender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the event log of the original withdraw refund transaction, the deposit refund transaction is on eth with the direction WETH-ETH

**getWithdrawRevokeEvent(sender, hashX) {}**

**Parameters**:

sender: A valid send object, wansender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the event log of the original withdraw revoke transaction, the deposit revoke transaction is on wan with the direction WETH-ETH

**getWithdrawHTLCLeftLockedTime(sender, hashX){}**

**Parameters**:

sender: A valid send object, ethsender or wansender

hashX: 32 bytes hash of X

**Returns**:

Promise return Object - the withdraw HTLC left locked time, the unit is second, after this revoke can be done.

If the sender is ethsender, this time is about the storeman deposit HTLC left locker time;

If the sender is wansender, this time is about the user deposit HTLC left locker time

**monitorTxConfirm(sender, txhash, waitBlocks) {}**

**Parameters**:

sender: A valid send object, ethsender or wansender

txhash: The transaction hash

waitBlocks: The wait block number to ensure this transaction is on chain

**Returns**:

Promise return Object - the receipt of the transaction by this transaction hash

**getEthLockTime(sender){}**

**Parameters**:

sender: A valid send object, ethsender or wansender

**Returns**:

Promise return Object - the definded locker time-slot in cross-chain contract

**getEthC2wRatio(sender){}**

**Parameters**:

sender: A valid send object, ethsender or wansender

**Returns**:

Promise return Object - the coin2wan ratio, 1 coin to how many WANs,such as ethereum 880\*DEFAULT\_PRECISE, and DEFAULT\_PRECISE = 10000

**getEthBalance(sender, addr) {}**

**Parameters**:

sender: A valid send object, ethsender

addr: The address to get the balance of.

**Returns**:

Promise return Object - the current balance for the given address in wei

**getWanBalance(sender, addr) {}**

**Parameters**:

sender: A valid send object, wansender

addr: The address to get the balance of.

**Returns**:

Promise return Object - the current balance for the given address in wei

**getBlockByNumber(sender, blockNumber) {}**

**Parameters**:

sender: A valid send object, ethsender or wansender

blockNumber: The block number to get the block of.

**Returns**:

Promise return Object - the block matching the block number

**getTxReceipt(sender,txhash){}**

**Parameters**:

sender: A valid send object, ethsender or wansender

txhash: The transaction hash to get the receipt of.

**Returns**:

Promise return Object - the transaction receipt matching the txhash

**getTxInfo(sender,txhash){}**

**Parameters**:

sender: A valid send object, ethsender or wansender

txhash: The transaction hash to get the transaction of.

**Returns**:

Promise return Object - the transaction matching the given transaction hash

**getTxHistory(option) {}**

**Parameters**:

option: A object contain key-value to search the transaction in local cross-chain db

**Returns**:

return the local transaction record matching the option

**getMultiEthBalances(sender, addrs) {}**

**Parameters**:

sender: A valid send object, ethsender

addrs: A array of address to get the balance of.

**Returns**:

Promise return Object - the balance of those given addresses

**getMultiWanBalances(sender, addrs) {}**

**Parameters**:

sender: A valid send object, wansender

addrs: A array of address to get the balance of.

**Returns**:

Promise return Object - the balance of those given addresses

**getMultiTokenBalance(sender, addrs) {}**

**Parameters**:

sender: A valid send object, wansender

addrs: A array of token address to get the token balance of

**Returns**:

Promise return Object - the token balance of those given addresses