

F:\git\java\mar3\filemonitor\target\account-ledger-module\account-ledger-module-0.doc

0:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\account-ledger\src\main\java\io\nuls\account\ledger\constant\AccountLedgerConstant.java  
package io.nuls.account.ledger.constant;

import io.nuls.kernel.constant.NulsConstant;

public interface AccountLedgerConstant extends NulsConstant {

    short MODULE\_ID\_ACCOUNTLEDGER = 9;  
    /\*\*  
    \* 200NULS  
    \*/  
    long MAX\_VALUE = 20000000000L;  
}

1:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\account-ledger\src\main\java\io\nuls\account\ledger\constant\AccountLedgerErrorCode.java  
\*/

package io.nuls.account.ledger.constant;

import io.nuls.kernel.constant.KernelErrorCode;

/\*\*  
\* @author: Niels Wang  
\*/  
public interface AccountLedgerErrorCode extends KernelErrorCode {  
  
    // ErrorCode ACCOUNT\_NOT\_EXIST = ErrorCode.init("90001");  
    // ErrorCode ADDRESS\_ERROR = ErrorCode.init("90004");  
    // ErrorCode SUCCESS = ErrorCode.init("90010");  
    // ErrorCode FAILED = ErrorCode.init("90011");  
    // ErrorCode PARAMETER\_ERROR = ErrorCode.init("90012");  
    // ErrorCode IO\_ERROR = ErrorCode.init("90013");  
    // ErrorCode SOURCE\_TX\_NOT\_EXSITS = ErrorCode.init("90014");  
    // ErrorCode UNKNOW\_ERROR = ErrorCode.init("90015");  
}

```
2:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\account-  
ledger\src\main\java\io\nuls\account\ledger\model\CoinDataResult.java  
*/
```

```
package io.nuls.account.ledger.model;
```

```
import io.nuls.kernel.model.Coin;
```

```
import io.nuls.kernel.model.Na;
```

```
import java.util.List;
```

```
/**
```

```
 * coinDatacoin
```

```
 *
```

```
 * @author Vivi
```

```
 */
```

```
public class CoinDataResult {
```

```
    private boolean enough;
```

```
    private List<Coin> coinList;
```

```
    /**
```

```
     *
```

```
     */
```

```
    private Coin change;
```

```
    private Na fee;
```

```
    /**
```

```
     * UTXO000000001,000000010,000000011
```

```
     * */
```

```
    private byte signType;
```

```
    public List<Coin> getCoinList() {
```

```
        return coinList;
```

```
    }
```

```
    public void setCoinList(List<Coin> coinList) {
```

```
        this.coinList = coinList;
```

```
    }
```

```
    public Na getFee() {
```

```

        return fee;
    }

    public void setFee(Na fee) {
        this.fee = fee;
    }

    public boolean isEnough() {
        return enough;
    }

    public void setEnough(boolean enough) {
        this.enough = enough;
    }

    public Coin getChange() {
        return change;
    }

    public void setChange(Coin change) {
        this.change = change;
    }

    public int getSignType() {
        return signType;
    }

    public void setSignType(byte signType) {
        this.signType = signType;
    }
}

```

```

3:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\account-
ledger\src\main\java\io\nuls\account\ledger\model\MultipleAddressTransferModel.java
*/

```

```

package io.nuls.account.ledger.model;

```

```

/**
 *
 */

```

```

public class MultipleAddressTransferModel {

```

```

//
private byte[] address;
//
private long amount;

public long getAmount() {
    return amount;
}

public void setAmount(long amount) {
    this.amount = amount;
}

public byte[] getAddress() {
    return address;
}

public void setAddress(byte[] address) {
    this.address = address;
}
}

```

4:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\account-ledger\src\main\java\io\nuls\account\ledger\model\TransactionInfo.java

```

package io.nuls.account.ledger.model;

import io.nuls.kernel.model.NulsDigestData;

/**
 * @author Facjas
 */
public class TransactionInfo {

    public static byte CONFIRMED = 1;
    public static byte UNCONFIRMED = 0;

    private NulsDigestData txHash;

    private long blockHeight;

    private long time;

```

```
private byte[] addresses;
```

```
private int txType;
```

```
private byte status;
```

```
private String info;
```

```
/**
```

```
 * contract address
```

```
 */
```

```
private byte[] contractAddress;
```

```
/**
```

```
 * contract token symbol
```

```
 */
```

```
private String symbol;
```

```
public NulsDigestData getTxHash() {
```

```
    return txHash;
```

```
}
```

```
public void setTxHash(NulsDigestData txHash) {
```

```
    this.txHash = txHash;
```

```
}
```

```
public long getBlockHeight() {
```

```
    return blockHeight;
```

```
}
```

```
public void setBlockHeight(long blockHeight) {
```

```
    this.blockHeight = blockHeight;
```

```
}
```

```
public long getTime() {
```

```
    return time;
```

```
}
```

```
public void setTime(long time) {
```

```
    this.time = time;
```

```
}
```

```
public byte[] getAddresses() {  
    return addresses;  
}
```

```
public void setAddresses(byte[] addresses) {  
    this.addresses = addresses;  
}
```

```
public int getTxType() {  
    return txType;  
}
```

```
public void setTxType(int txType) {  
    this.txType = txType;  
}
```

```
public byte getStatus() {  
    return status;  
}
```

```
public void setStatus(byte status) {  
    this.status = status;  
}
```

```
public String getInfo() {  
    return info;  
}
```

```
public void setInfo(String info) {  
    this.info = info;  
}
```

```
public byte[] getContractAddress() {  
    return contractAddress;  
}
```

```
public void setContractAddress(byte[] contractAddress) {  
    this.contractAddress = contractAddress;  
}
```

```
public String getSymbol() {  
    return symbol;  
}
```

```

    }

    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }

    public int compareTo(long thatTime) {
        if(this.time > thatTime) {
            return -1;
        } else if(this.time < thatTime) {
            return 1;
        }
        return 0;
    }
}

```

5:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\account-ledger\src\main\java\io\nuls\account\ledger\module\AbstractAccountLedgerModule.java  
package io.nuls.account.ledger.module;

```

import io.nuls.account.ledger.constant.AccountLedgerConstant;
import io.nuls.kernel.module.BaseModuleBootstrap;

public abstract class AbstractAccountLedgerModule extends BaseModuleBootstrap {
    public AbstractAccountLedgerModule() {
        super(AccountLedgerConstant.MODULE_ID_ACCOUNTLEDGER);
    }
}

```

6:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\account-ledger\src\main\java\io\nuls\account\ledger\service\AccountLedgerService.java  
\*/

```

package io.nuls.account.ledger.service;

import io.nuls.account.ledger.model.MultipleAddressTransferModel;
import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.account.model.Account;
import io.nuls.account.model.Balance;
import io.nuls.account.ledger.model.CoinDataResult;
import io.nuls.account.model.MultiSigAccount;

```

```
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.script.Script;
import io.nuls.kernel.script.TransactionSignature;
```

```
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.Set;
```

```
public interface AccountLedgerService {
```

```
    /**
     * save a confirmed tx to account ledger.
     * save if the tx is relative to local accounts, or do nothing.
     *
     * @param tx transaction to save
     * @return return the tx count saved,
     */
    Result<Integer> saveConfirmedTransaction(Transaction tx);
```

```
    /**
     * save a unconfirmed tx to account ledger.
     * save if the tx is relative to local accounts, or do nothing.
     *
     * @param tx transaction to save
     * @return return the tx count saved,
     */
    Result<Integer> verifyAndSaveUnconfirmedTransaction(Transaction tx);
```

```
    /**
     * save a tx to account ledger.
     * save if the tx is relative to local accounts, or do nothing.
     *
     * @param txs transactions to save
     * @return return the tx count saved,
```



```
*/  
Result<Integer> saveConfirmedTransactionList(List<Transaction> txs);
```

```
/**  
 * get an .unconfirmed transaction  
 *  
 * @param hash transaction hash  
 * @return return the unconfirmed tx  
 */  
Result<Transaction> getUnconfirmedTransaction(NulsDigestData hash);
```

```
/**  
 * get all.unconfirmed transactions  
 *  
 * @return return all the unconfirmed txs  
 */  
Result<List<Transaction>> getAllUnconfirmedTransaction();
```

```
/**  
 * rollbackTransaction a tx in account ledger  
 * save if the tx is relative to local accounts, or do nothing  
 *  
 * @param tx transaction to rollbackTransaction  
 * @return return the tx count rollbacked  
 */  
Result<Integer> deleteTransaction(Transaction tx);
```

```
/**  
 * rollbackTransaction a tx list in account ledger.  
 * save if the tx is relative to local accounts, or do nothing  
 *  
 * @param txs transactions to rollbackTransaction  
 * @return return the tx count rollbacked  
 */  
Result<Integer> rollbackTransactions(List<Transaction> txs);
```

```
/**  
 * get the balance of an local account.  
 *  
 * @param address account address  
 * @return return balance of account, return 0 if account is not a local account
```

\* @throws NulsException NulsException

\*/

Result<Balance> getBalance(byte[] address);

/\*\*

\* get usable coinData

\*

\* @param address account address

\* @param amount amount want to use

\* @param size size of transaction ,to calc the fee

\* @param price price 1/KB

\* @return return balance of account, return 0 if account is not a local account

\* @throws NulsException NulsException

\*/

CoinDataResult getCoinData(byte[] address, Na amount, int size, Na price) throws  
NulsException;

/\*\*

\* A transfers NULS to B

\*

\* @param from address of A

\* @param to address of B

\* @param values NULS amount

\* @param password password of A

\* @param remark remarks of transaction

\* @param price Unit price of fee

\* @return Result

\*/

Result transfer(byte[] from, byte[] to, Na values, String password, byte[] remark, Na price);

/\*\*

\* create and send a dapp transaction

\* @param from from address

\* @param password

\* @param data

\* @param remark

\* @return

\*/

Result dapp(byte[] from, String password, byte[] data, byte[] remark);

/\*\*

\* calculation fee of transaction

```

*
* @param from address of A
* @param to address of B
* @param values NULS amount
* @param remark remarks of transaction
* @param price Unit price of fee
* @return Result
*/

```

Result transferFee(byte[] from, byte[] to, Na values, String remark, Na price);

```

/**
* create a transaction by inputs data and outputs data
*
* @param inputs used utxos
* @param outputs new utxos
* @param remark remarks of transaction
* @return Result
*/

```

Result createTransaction(List<Coin> inputs, List<Coin> outputs, byte[] remark);

```

/**
*
*
* @param tx tx
* @param ecKey ecKey
* @return Transaction
* @throws IOException IOException
*/

```

Transaction signTransaction(Transaction tx, ECKey ecKey) throws IOException;

```

/**
*
*
* @param tx tx
* @return Result
*/

```

Result broadcast(Transaction tx);

```

// /**
// * get local address list
// *
// * @return true if a address is a local address

```

```

// */
Result unlockCoinData(Transaction tx, long newLockTime);

Result rollbackUnlockTxCoinData(Transaction tx);

/**
 * load the local ledger of a account when an account imported
 *
 * @param address address
 * @return true if a address is a local address
 */
Result importLedgerByAddress(String address);

/**
 * @param address address
 * @return Result
 */
Result<List<TransactionInfo>> getTxInfoList(byte[] address);

/**
 * @param address address
 * @return Result
 */
Result<List<Coin>> getLockedUtxo(byte[] address);

/**
 * delete unconfirmed transactions of an account
 *
 * @param address address
 * @return Result
 */
Result<Integer> deleteUnconfirmedTx(byte[] address);

/**
 * ()
 * Calculate the maximum amount of a transaction (not exceeding the maximum transaction
data size) based on the account
 *
 * @param address
 * @param tx
 * @param price
 * @return

```

\*/

Result<Na> getMaxAmountOfOnce(byte[] address, Transaction tx, Na price);

/\*\*

\* ()

\* Calculate the maximum amount of a transaction (not exceeding the maximum transaction data size) based on the account

\*

\* @param address

\* @param tx

\* @param price

\* @return

\*/

Result<Na> getMultiMaxAmountOfOnce(byte[] address, Transaction tx, Na price,int size);

/\*\*

\*

\* @param fromModelList

\* @param toModelList

\* @param password

\* @param amount

\* @param remark

\* @param price

\* @return

\*/

Result multipleAddressTransfer(List<MultipleAddressTransferModel> fromModelList,  
List<MultipleAddressTransferModel> toModelList, String password,Na amount, String remark, Na  
price);

/\*\*

\*

\*

\* @param address

\* @param password

\* @param price

\* @return

\*/

Result changeWhole(byte[] address, String password, Na price);

/\*\*

\* Get the fee for the transfer transaction

```

*
* @param address
* @param amount
* @param size
* @param price
* @return
*/

```

Na getTxFee(byte[] address, Na amount, int size, Na price);

```

/**
*
*
* @param address
* @param price
* @return
*/

```

Result estimateFee(byte[] address, Na price);

```

/**
* utxo--
*
* @param address
* @return
*/

```

Result getAvailableTotalUTXO(byte[] address);

```

/**
* A transfers NULS to B
* @param fromAddr
* @param signAddr
* @param outputs
* @param values    NULS amount
* @param password  password of A
* @param remark    remarks of transaction
* @param price     Unit price of fee
* @param pubkeys
* @param m
* @param txdata
* @return Result
*/

```

Result transferP2SH(byte[] fromAddr, byte[] signAddr , List<MultipleAddressTransferModel> outputs, Na values, String password, String remark, Na price, List<String>pubkeys, int m, String

txdata);

/\*\*

\* get usable coinData CoinData

\*

\* @param address account address

\* @param amount amount want to use

\* @param size size of transaction ,to calc the fee

\* @param price price 1/KB

\* @return return balance of account, return 0 if account is not a local account

\* @throws NulsException NulsException

\*/

CoinDataResult getMutilCoinData(byte[] address, Na amount, int size, Na price);

/\*\*

\* A transfers NULS to B

\*

\* @param fromAddr

\* @param signAddr

\* @param outputs

\* @param password password of A

\* @param remark remarks of transaction

\* @return Result

\*/

Result createP2shTransfer(String fromAddr, String signAddr,  
List<MultipleAddressTransferModel> outputs, String password, String remark);

/\*\*

\* A transfers NULS to B

\*

\* @param signAddr

\* @param password password of A

\* @param txdata

\* @return Result

\*/

Result signMultiTransaction(String signAddr,String password,String txdata);

/\*\*

\* A transfers NULS to B

\*

\* @param tx

```

    * @param transactionSignature
    * @param account
    * @param password password
    * @return Result
    */
    Result txMultiProcess(Transaction tx, TransactionSignature transactionSignature, Account
account, String password);

    /**
     * A transfers NULS to B
     *
     * @param multiSigAccount
     * @return Result
     */
    Script getRedeemScript(MultiSigAccount multiSigAccount);

    /**
     * A transfers NULS to B
     *
     * @param utxoList      UTXO
     * @return Result
     */
    Result getSignatureType(List<String> utxoList);
}

```

7:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-  
base\src\main\java\io\nuls\account\ledger\base\manager\BalanceCacheEntity.java

```

    */

package io.nuls.account.ledger.base.manager;

import io.nuls.account.model.Balance;

    /**
     * author Facjas
     * date 2018/6/12.
     */
    public class BalanceCacheEntity {
        private Balance balance;
        long lowestLockHeigh;
        long earlistLockTime;
    }

```



```

public Balance getBalance() {
    return balance;
}

public void setBalance(Balance balance) {
    this.balance = balance;
}

public long getLowestLockHeigh() {
    return lowestLockHeigh;
}

public void setLowestLockHeigh(long lowestLockHeigh) {
    this.lowestLockHeigh = lowestLockHeigh;
}

public long getEarlistLockTime() {
    return earlistLockTime;
}

public void setEarlistLockTime(long earlistLockTime) {
    this.earlistLockTime = earlistLockTime;
}
}

```

8:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\main\java\io\nuls\account\ledger\base\manager\BalanceManager.java  
\*/

```

package io.nuls.account.ledger.base.manager;

import io.nuls.account.ledger.base.util.CoinComparator;
import io.nuls.account.ledger.constant.AccountLedgerErrorCode;
import io.nuls.account.ledger.storage.service.LocalUtxoStorageService;
import io.nuls.account.model.Account;
import io.nuls.kernel.model.Address;
import io.nuls.account.model.Balance;
import io.nuls.account.service.AccountService;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.core.tools.log.Log;
import io.nuls.db.model.Entry;
import io.nuls.kernel.constant.NulsConstant;

```

```
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.utils.AddressTool;
```

```
import java.util.*;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
```

```
/**
 *
 */
```

@Component

```
public class BalanceManager {
```

```
    @Autowired
```

```
    private LocalUtxoStorageService localUtxoStorageService;
```

```
    @Autowired
```

```
    private AccountService accountService;
```

```
    private Map<String, BalanceCacheEntity> balanceMap = new HashMap<>();
```

```
    private Lock lock = new ReentrantLock();
```

```
/**
 *
 */
```

```
    public void initAccountBalance() {
        balanceMap.clear();
```

```
        Collection<Account> accounts = accountService.getAccountList().getData();
```

```
        if (accounts == null) {
            return;
```

```
        }
```

```
        for (Account account : accounts) {
```

```

    try {
        calBalanceByAddress(account.getAddress().getAddressBytes());
    } catch (NulsException e) {
        Log.info("getbalance of address[" + account.getAddress().getBase58() + "] error");
    }
}

/**
 *
 */
public Result<Balance> getBalance(Address address) {
    return getBalance(address.getAddressBytes());
}

/**
 *
 */
public Result<Balance> getBalance(byte[] address) {
    lock.lock();
    try {
        if (address == null || address.length != Address.ADDRESS_LENGTH) {
            return Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR);
        }

        String addressKey = AddressTool.getStringAddressByBytes(address);
        BalanceCacheEntity entity = balanceMap.get(addressKey);
        Balance balance = null;
        if (entity == null || (entity.getEarlistLockTime() > 0L && entity.getEarlistLockTime() <=
TimeService.currentTimeMillis())) {
            try {
                balance = calBalanceByAddress(address);
            } catch (NulsException e) {
                Log.info("getbalance of address[" + AddressTool.getStringAddressByBytes(address)
+ "] error");
            }
        } else {
            balance = entity.getBalance();
        }
        return Result.getSuccess().setData(balance);
    } finally {
        lock.unlock();
    }
}

```

```

    }
}

/**
 *
 */
public void refreshBalance(byte[] address) {
    if (address != null) {
        balanceMap.remove(AddressTool.getStringAddressByBytes(address));
    }
}

public void refreshBalance() {
    lock.lock();
    try {
        balanceMap.clear();
    } finally {
        lock.unlock();
    }
}

/**
 *
 */
public Balance calBalanceByAddress(byte[] address) throws NulsException {
    lock.lock();
    try {
        if (accountService.getAccount(address).isFailed()) {
            return null;
        }
        List<Coin> coinList = getCoinListByAddress(address);
        Collections.sort(coinList, CoinComparator.getInstance());

        BalanceCacheEntity balanceCacheEntity = new BalanceCacheEntity();

        Na usable = Na.ZERO;
        Na locked = Na.ZERO;
        for (Coin coin : coinList) {
            if (coin.usable()) {
                usable = usable.add(coin.getNa());
            } else {
                locked = locked.add(coin.getNa());
            }
        }
    }
}

```

```

        long lockTime = coin.getLockTime();
        // the consensus lock type
        if (lockTime <= 0L) {
            continue;
        }
        // the height lock type
        if (balanceCacheEntity.getLowestLockHeigh() == 0L || (lockTime <
NulsConstant.BLOCKHEIGHT_TIME_DIVIDE && lockTime <
balanceCacheEntity.getLowestLockHeigh())) {
            balanceCacheEntity.setLowestLockHeigh(lockTime);
            continue;
        }
        // the time lock type
        if (balanceCacheEntity.getEarlistLockTime() == 0L || (lockTime >
NulsConstant.BLOCKHEIGHT_TIME_DIVIDE && lockTime <
balanceCacheEntity.getEarlistLockTime())) {
            balanceCacheEntity.setEarlistLockTime(lockTime);
            continue;
        }
    }
}

```

```

Balance balance = new Balance();
balance.setUsable(usable);
balance.setLocked(locked);
balance.setBalance(usable.add(locked));
balanceCacheEntity.setBalance(balance);

```

```

        balanceMap.put(AddressTool.getStringAddressByBytes(address), balanceCacheEntity);
        return balance;
    } finally {
        lock.unlock();
    }
}

```

```

public List<Coin> getCoinListByAddress(byte[] address) {
    List<Coin> coinList = new ArrayList<>();
    Collection<Entry<byte[], byte[]>> rawList = localUtxoStorageService.loadAllCoinList();
    for (Entry<byte[], byte[]> coinEntry : rawList) {
        Coin coin = new Coin();
        try {

```

```

        coin.parse(coinEntry.getValue(), 0);
    } catch (NulsException e) {
        Log.info("parse coin form db error");
        continue;
    }
    if (Arrays.equals(coin.getAddress(), address)) {
        coin.setTempOwner(coin.getOwner());
        coin.setOwner(coinEntry.getKey());
        coinList.add(coin);
    }
}
return coinList;
}

public void refreshBalanceIfNecessary() {
    long bestHeight = NulsContext.getInstance().getBestHeight();
    Set<String> set = new HashSet<>(balanceMap.keySet());
    for (String address : set) {
        BalanceCacheEntity entity = balanceMap.get(address);
        if (entity == null) {
            balanceMap.remove(address);
            continue;
        }
        if (entity.getEarlistLockTime() == 0L && entity.getLowestLockHeigh() == 0L) {
            continue;
        }
        if (entity.getLowestLockHeigh() > 0L && entity.getLowestLockHeigh() <= bestHeight) {
            balanceMap.remove(address);
            continue;
        }
        if (entity.getEarlistLockTime() > 0L && entity.getEarlistLockTime() <=
TimeService.currentTimeMillis()) {
            balanceMap.remove(address);
            continue;
        }
    }
}
}
}

```

9:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\main\java\io\nuls\account\ledger\base\module\impl\AccountLedgerModuleBootstrap.java  
package io.nuls.account.ledger.base.module.impl;

```

import io.nuls.account.constant.AccountConstant;
import io.nuls.account.ledger.base.manager.BalanceManager;
import io.nuls.account.ledger.base.task.CheckUnConfirmTxThread;
import io.nuls.account.ledger.constant.AccountLedgerConstant;
import io.nuls.account.ledger.module.AbstractAccountLedgerModule;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.protocol.constant.ProtocolConstant;

import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

/**
 * @desription:
 * @author: PierreLuo
 */
public class AccountLedgerModuleBootstrap extends AbstractAccountLedgerModule {

    @Override
    public void init() {
    }

    @Override
    public void start() {
        this.waitForDependencyRunning(AccountConstant.MODULE_ID_ACCOUNT,
ProtocolConstant.MODULE_ID_PROTOCOL);
        BalanceManager balanceManager = NulsContext.getServiceBean(BalanceManager.class);
        balanceManager.initAccountBalance();
        ScheduledThreadPoolExecutor executor = TaskManager.createScheduledThreadPool(1,
new NulsThreadFactory(AccountLedgerConstant.MODULE_ID_ACCOUNTLEDGER,
"CheckUnConfirmTxThread"));
        executor.scheduleAtFixedRate(NulsContext.getServiceBean(CheckUnConfirmTxThread.class), 10
, 10, TimeUnit.MINUTES);
    }

    @Override
    public void shutdown() {
        TaskManager.shutdownByModuleId(this.getModuleId());
    }

```

```

    }

    @Override
    public void destroy() {

    }

    @Override
    public String getInfo() {
        return null;
    }
}

```

10:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\main\java\io\nuls\account\ledger\base\service\impl\AccountLedgerServiceImpl.java  
\*/

```

package io.nuls.account.ledger.base.service.impl;

import io.nuls.account.constant.AccountConstant;
import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.base.manager.BalanceManager;
import io.nuls.account.ledger.base.service.LocalUtxoService;
import io.nuls.account.ledger.base.service.TransactionInfoService;
import io.nuls.account.ledger.base.util.AccountLegerUtils;
import io.nuls.account.ledger.base.util.CoinComparator;
import io.nuls.account.ledger.base.util.CoinComparatorDesc;
import io.nuls.account.ledger.constant.AccountLedgerErrorCode;
import io.nuls.account.ledger.model.CoinDataResult;
import io.nuls.account.ledger.model.MultipleAddressTransferModel;
import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.ledger.storage.po.TransactionInfoPo;
import io.nuls.account.ledger.storage.service.UnconfirmedTransactionStorageService;
import io.nuls.account.model.Account;
import io.nuls.account.model.Balance;
import io.nuls.account.model.MultiSigAccount;
import io.nuls.account.service.AccountService;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.service.ContractService;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;

```



```
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.*;
import io.nuls.kernel.script.*;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.TransactionFeeCalculator;
import io.nuls.kernel.utils.TransactionManager;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.constant.LedgerErrorCode;
import io.nuls.ledger.service.LedgerService;
import io.nuls.ledger.util.LedgerUtil;
import io.nuls.protocol.model.tx.DataTransaction;
import io.nuls.protocol.model.tx.LogicData;
import io.nuls.protocol.model.tx.TransferTransaction;
import io.nuls.protocol.model.validator.TxMaxSizeValidator;
import io.nuls.protocol.model.validator.TxRemarkValidator;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.TransactionService;
```

```
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.*;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.stream.Collectors;
```

```
/**
```

```
 * @author Facjas
```

```
 */
```

```
@Service
```

```

public class AccountLedgerServiceImpl implements AccountLedgerService, InitializingBean {

    @Autowired
    private LocalUtxoService localUtxoService;

    @Autowired
    private UnconfirmedTransactionStorageService unconfirmedTransactionStorageService;

    @Autowired
    private AccountService accountService;

    @Autowired
    private BalanceManager balanceManager;

    @Autowired
    private TransactionService transactionService;

    @Autowired
    private LedgerService ledgerService;

    @Autowired
    private BlockService blockService;

    @Autowired
    private TransactionInfoService transactionInfoService;

    @Autowired
    private ContractService contractService;

    private Lock lock = new ReentrantLock();
    private Lock saveLock = new ReentrantLock();
    private Lock changeWholeLock = new ReentrantLock();

    //todo Save locally used transactions
    private Set<String> usedTxSets;

    @Override
    public void afterPropertiesSet() throws NulsException {
    }

    @Override
    public Result<Integer> saveConfirmedTransactionList(List<Transaction> txs) {

```

```

if (txs == null || txs.size() == 0) {
    return Result.getSuccess().setData(0);
}

```

```

List<byte[]> localAddresses = AccountLegerUtils.getLocalAddresses();

```

```

List<Transaction> savedTxList = new ArrayList<>();

```

```

Result result;

```

```

for (int i = 0; i < txs.size(); i++) {

```

```

    Transaction tx = txs.get(i);

```

```

    List<byte[]> addresses = AccountLegerUtils.getRelatedAddresses(tx, localAddresses);

```

```

    if (addresses == null || addresses.size() == 0) {
        continue;
    }

```

```

    result = saveConfirmedTransaction(tx, addresses);

```

```

    if (result.isSuccess()) {

```

```

        if (result.getData() != null && (int) result.getData() == 1) {
            savedTxList.add(tx);
        }

```

```

    } else {

```

```

        rollbackTransactions(savedTxList, false);
        return result;
    }

```

```

}

```

```

balanceManager.refreshBalanceIfNecessary();

```

```

return Result.getSuccess().setData(savedTxList.size());

```

```

}

```

```

@Override

```

```

public Result<Integer> saveConfirmedTransaction(Transaction tx) {

```

```

    if (tx == null) {

```

```

        return Result.getSuccess().setData(0);
    }

```

```

    List<byte[]> addresses = AccountLegerUtils.getRelatedAddresses(tx);

```

```

    if (addresses == null || addresses.size() == 0) {
        return Result.getSuccess().setData(0);
    }

```

```

    }
    return saveConfirmedTransaction(tx, addresses);
}

```

```

private Result<Integer> saveConfirmedTransaction(Transaction tx, List<byte[]> addresses) {
    if (tx == null) {
        return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
    }

    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    txInfoPo.setStatus(TransactionInfo.CONFIRMED);

    Result result = transactionInfoService.saveTransactionInfo(txInfoPo, addresses);
    if (result.isFailed()) {
        return result;
    }

    // coin
    Transaction unconfirmedTx =
    unconfirmedTransactionStorageService.getUnconfirmedTx(tx.getHash()).getData();

    if (unconfirmedTx == null) {
        result = localUtxoService.saveUtxoForLocalAccount(tx);
        if (result.isFailed()) {
            transactionInfoService.deleteTransactionInfo(txInfoPo);
            return result;
        }
    } else {
        unconfirmedTransactionStorageService.deleteUnconfirmedTx(tx.getHash());
    }

    for (int i = 0; i < addresses.size(); i++) {
        balanceManager.refreshBalance(addresses.get(i));
    }
    result.setData(new Integer(1));
    return result;
}

```

@Override

```

public Result<Integer> verifyAndSaveUnconfirmedTransaction(Transaction tx) {
    saveLock.lock();

```

```

try {
    ValidateResult result = tx.verify();
    if (result.isFailed()) {
        return result;
    }
    if (!tx.isSystemTx()) {
        Map<String, Coin> toCoinMap = addToCoinMap(tx);
        if (usedTxSets == null) {
            initUsedTxSets();
        }
        result = this.ledgerService.verifyCoinData(tx, toCoinMap, usedTxSets);
        if (result.isFailed()) {
            Log.info("verifyCoinData failed : " + result.getMsg());
            return result;
        }
    }
    Result<Integer> res = saveUnconfirmedTransaction(tx);
    return res;
} finally {
    saveLock.unlock();
}
}

```

```

public void resetUsedTxSets() {
    usedTxSets = null;
}

```

```

public void initUsedTxSets() {
    usedTxSets = new HashSet<>();
    List<Transaction> allUnconfirmedTxns =
unconfirmedTransactionStorageService.loadAllUnconfirmedList().getData();
    for (Transaction tx : allUnconfirmedTxns) {
        CoinData coinData = tx.getCoinData();
        if (coinData == null) {
            continue;
        }
        List<Coin> froms = tx.getCoinData().getFrom();
        for (Coin from : froms) {
            usedTxSets.add(LedgerUtil.asString(from.getOwner()));
        }
    }
}

```

```

    }

    private Map<String, Coin> addToCoinMap(Transaction transaction) {
        Map<String, Coin> toMap = new HashMap<>();

        CoinData coinData = transaction.getCoinData();
        if (coinData == null) {
            return toMap;
        }

        List<Coin> froms = coinData.getFrom();

        if (froms == null || froms.size() == 0) {
            return toMap;
        }

        for (Coin coin : froms) {
            byte[] keyBytes = coin.getOwner();
            try {
                Transaction unconfirmedTx =
getUnconfirmedTransaction(NulsDigestData.fromDigestHex(LedgerUtil.getTxHash(keyBytes))).get
Data();
                if (unconfirmedTx != null) {
                    int index = LedgerUtil.getIndex(keyBytes);
                    Coin toCoin = unconfirmedTx.getCoinData().getTo().get(index);
                    toMap.put(LedgerUtil.asString(keyBytes), toCoin);
                }
            } catch (NulsException e) {
                Log.error(e);
            }
        }
        return toMap;
    }

    protected Result saveUnconfirmedTransaction(Transaction tx) {
        if (tx == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        //
        List<byte[]> localAccountList = AccountLegerUtils.getLocalAddresses();
        List<byte[]> addresses = AccountLegerUtils.getRelatedAddresses(tx, localAccountList);
        if (addresses == null || addresses.size() == 0) {

```

```

        return Result.getSuccess().setData(new Integer(0));
    }

    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    txInfoPo.setStatus(TransactionInfo.UNCONFIRMED);

    Result result = transactionInfoService.saveTransactionInfo(txInfoPo, addresses);
    if (result.isFailed()) {
        return result;
    }

    result = localUtxoService.saveUtxoForAccount(tx, addresses);
    if (result.isFailed()) {
        transactionInfoService.deleteTransactionInfo(txInfoPo);
        return result;
    }

    result = unconfirmedTransactionStorageService.saveUnconfirmedTx(tx.getHash(), tx);

    for (int i = 0; i < addresses.size(); i++) {
        balanceManager.refreshBalance(addresses.get(i));
    }
    return result;
}

```

```

@Override
public Result<List<Transaction>> getAllUnconfirmedTransaction() {
    List<Transaction> localTxList =
unconfirmedTransactionStorageService.loadAllUnconfirmedList().getData();
    return Result.getSuccess().setData(localTxList);
}

```

```

@Override
public Result<Integer> rollbackTransactions(List<Transaction> txs) {
    Result result = rollbackTransactions(txs, true);
    if (result.isSuccess()) {
        balanceManager.refreshBalanceIfNecessary();
    }
    return result;
}

```

```

private Result<Integer> rollbackTransactions(List<Transaction> txs, boolean isCheckMine) {
    List<Transaction> txListToRollback;
    if (isCheckMine) {
        txListToRollback = filterLocalTransaction(txs);
    } else {
        txListToRollback = txs;
    }
    for (int i = txListToRollback.size() - 1; i >= 0; i--) {
        rollbackTransaction(txListToRollback.get(i));
    }
    return Result.getSuccess().setData(new Integer(txListToRollback.size()));
}

```

```

private Result<Integer> rollbackTransaction(Transaction tx) {
    if (!AccountLegerUtils.isLocalTransaction(tx)) {
        return Result.getSuccess().setData(new Integer(0));
    }

    List<byte[]> addresses = AccountLegerUtils.getRelatedAddresses(tx);
    if (addresses == null || addresses.size() == 0) {
        return Result.getSuccess().setData(new Integer(0));
    }
    if (tx.isSystemTx()) {
        return deleteTransaction(tx);
    }
    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    Result result = transactionInfoService.saveTransactionInfo(txInfoPo, addresses);
    if (result.isFailed()) {
        return result;
    }
    result = unconfirmedTransactionStorageService.saveUnconfirmedTx(tx.getHash(), tx);
    return result;
}

```

@Override

```

public Result<Integer> deleteTransaction(Transaction tx) {
    if (!AccountLegerUtils.isLocalTransaction(tx)) {
        return Result.getSuccess().setData(new Integer(0));
    }

    List<byte[]> addresses = AccountLegerUtils.getRelatedAddresses(tx);
    if (addresses == null || addresses.size() == 0) {

```



```

        return Result.getSuccess().setData(new Integer(0));
    }

    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    Result result = transactionInfoService.deleteTransactionInfo(txInfoPo);

    if (result.isFailed()) {
        return result;
    }
    result = localUtxoService.deleteUtxoOfTransaction(tx);

    if (result.isFailed()) {
        return result;
    }
    result = unconfirmedTransactionStorageService.deleteUnconfirmedTx(tx.getHash());

    for (int i = 0; i < addresses.size(); i++) {
        balanceManager.refreshBalance(addresses.get(i));
    }

    if (usedTxSets != null) {
        CoinData coinData = tx.getCoinData();
        if (coinData != null) {
            List<Coin> froms = tx.getCoinData().getFrom();
            for (Coin from : froms) {
                usedTxSets.remove(LedgerUtil.asString(from.getOwner()));
            }
        }
    }

    return result;
}

```

@Override

```

public Result<Balance> getBalance(byte[] address) {
    if (address == null || address.length != Address.ADDRESS_LENGTH) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }

    if (!AccountLegerUtils.isLocalAccount(address)) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
}

```

```
Balance balance = balanceManager.getBalance(address).getData();
```

```
if (balance == null) {  
    return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);  
}
```

```
return Result.getSuccess().setData(balance);  
}
```

@Override

```
public CoinDataResult getCoinData(byte[] address, Na amount, int size, Na price) throws  
NulsException {
```

```
    if (null == price) {  
        throw new NulsRuntimeException(KernelErrorCode.PARAMETER_ERROR);  
    }
```

```
    lock.lock();
```

```
    try {  
        CoinDataResult coinDataResult = new CoinDataResult();  
        coinDataResult.setEnough(false);
```

```
        List<Coin> coinList = balanceManager.getCoinListByAddress(address);
```

```
        coinList = coinList.stream()  
            .filter(coin1 -> coin1.usable() && !Na.ZERO.equals(coin1.getNa()))  
            .sorted(CoinComparator.getInstance())  
            .collect(Collectors.toList());
```

```
        if (coinList.isEmpty()) {  
            return coinDataResult;  
        }
```

```
        List<Coin> coins = new ArrayList<>();
```

```
        Na values = Na.ZERO;
```

```
//
```

```
for (int i = 0; i < coinList.size(); i++) {
```

```
    Coin coin = coinList.get(i);
```

```
    coins.add(coin);
```

```
    size += coin.size();
```

```
    if (i == 127) {
```

```
        size += 1;
```

```
    }
```

```

//
Na fee = TransactionFeeCalculator.getFee(size, price);
values = values.add(coin.getNa());

/**
 * UTXO
 * */
int signType = coinDataResult.getSignType();
if (signType != 3) {
    if ((signType & 0x01) == 0 && coin.getTempOwner().length == 23) {
        coinDataResult.setSignType((byte) (signType | 0x01));
        size += P2PHKSignature.SERIALIZE_LENGTH;
    } else if ((signType & 0x02) == 0 && coin.getTempOwner().length != 23) {
        coinDataResult.setSignType((byte) (signType | 0x02));
        size += P2PHKSignature.SERIALIZE_LENGTH;
    }
}

//
if (values.isGreaterThan(amount.add(fee))) {
    Na change = values.subtract(amount.add(fee));
    Coin changeCoin = new Coin();
    if (address[2] == NulsContext.P2SH_ADDRESS_TYPE) {
        changeCoin.setOwner(SignatureUtil.createOutputScript(address).getProgram());
    } else {
        changeCoin.setOwner(address);
    }
    changeCoin.setNa(change);
    fee = TransactionFeeCalculator.getFee(size + changeCoin.size(), price);
    if (values.isLessThan(amount.add(fee))) {
        continue;
    }
    changeCoin.setNa(values.subtract(amount.add(fee)));
    if (!changeCoin.getNa().equals(Na.ZERO)) {
        coinDataResult.setChange(changeCoin);
    }
}
coinDataResult.setFee(fee);
if (values.isGreaterOrEquals(amount.add(fee))) {
    coinDataResult.setEnough(true);
    coinDataResult.setCoinList(coins);
    break;
}

```

```

        }
    }
    return coinDataResult;
} finally {
    lock.unlock();
}
}

```

```
/**
```

```
 * ()
```

```
 */
```

```
@Override
```

```

public Result<Na> getMaxAmountOfOnce(byte[] address, Transaction tx, Na price) {
    lock.lock();
    try {
        tx.getCoinData().setFrom(null);
        int txSize = tx.size();
        //txsize
        for (Coin coin : tx.getCoinData().getTo()) {
            txSize += coin.size();
        }
        //
        if (null == tx.getTransactionSignature()) {
            txSize = txSize + P2PHKSignature.SERIALIZE_LENGTH;
        }
        //sizecoindatafrom
        int targetSize = TxMaxSizeValidator.MAX_TX_SIZE - txSize;
        List<Coin> coinList = balanceManager.getCoinListByAddress(address);
        if (coinList.isEmpty()) {
            return Result.getSuccess().setData(Na.ZERO);
        }
        Collections.sort(coinList, CoinComparator.getInstance());
        Na max = Na.ZERO;
        int size = 0;
        //
        for (int i = 0; i < coinList.size(); i++) {
            Coin coin = coinList.get(i);
            if (!coin.usable()) {
                continue;
            }
            if (coin.getNa().equals(Na.ZERO)) {

```

```

        continue;
    }
    size += coin.size();
    if (i == 127) {
        size += 1;
    }
    if (size > targetSize) {
        break;
    }
    max = max.add(coin.getNa());
}
Na fee = TransactionFeeCalculator.getFee(size, price);
if (max.isLessThan(fee)) {
    //0
    return Result.getSuccess().setData(Na.ZERO);
}
max = max.subtract(fee);
return Result.getSuccess().setData(max);
} catch (Exception e) {
    Log.error(e.fillInStackTrace());
    return Result.getFailed(TransactionErrorCode.DATA_ERROR);
} finally {
    lock.unlock();
}
}

```

@Override

```

public Na getTxFee(byte[] address, Na amount, int size, Na price) {
    List<Coin> coinList = balanceManager.getCoinListByAddress(address);
    if (coinList.isEmpty()) {
        return Na.ZERO;
    }
    Collections.sort(coinList, CoinComparator.getInstance());
    if (null == price) {
        price = TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES;
    }
    Na values = Na.ZERO;
    Na fee = null;
    for (int i = 0; i < coinList.size(); i++) {
        Coin coin = coinList.get(i);
        if (!coin.usable()) {
            continue;
        }
    }
}

```

```

    }
    size += coin.size();
    if (i == 127) {
        size += 1;
    }
    fee = TransactionFeeCalculator.getFee(size, price);
    values = values.add(coin.getNa());

    if (values.isGreaterOrEquals(amount.add(fee))) {
        Na change = values.subtract(amount.add(fee));
        if (change.isGreaterThan(Na.ZERO)) {
            Coin changeCoin = new Coin();
            changeCoin.setOwner(address);
            changeCoin.setNa(change);

            fee = TransactionFeeCalculator.getFee(size + changeCoin.size(), price);
            if (values.isLessThan(amount.add(fee))) {
                continue;
            } else {
                break;
            }
        } else {
            break;
        }
    }
}
return fee;
}

```

@Override

```

public Result estimateFee(byte[] address, Na price) {
    if (null == price) {
        throw new NulsRuntimeException(KernelErrorCode.PARAMETER_ERROR);
    }
    Transaction tx = new TransferTransaction();
    tx.setTime(TimeService.currentTimeMillis());
    lock.lock();
    try {
        //coin
        List<Coin> coinList = balanceManager.getCoinListByAddress(address);
        if (coinList.isEmpty()) {
            Result.getFailed(TransactionErrorCode.DATA_ERROR);
        }
    }
}

```

```

}
tx.setCoinData(null);
//coindatato38 ++
int txSize = tx.size() + 38 + TxRemarkValidator.MAX_REMARK_LEN;
int targetSize = TxMaxSizeValidator.MAX_TX_SIZE - txSize;
Collections.sort(coinList, CoinComparatorDesc.getInstance());
int size = tx.size() + 38;
//
byte signType = 0;
int txNum = 1;
for (int i = 0; i < coinList.size(); i++) {
    Coin coin = coinList.get(i);
    if (!coin.usable()) {
        continue;
    }
    if (coin.getNa().equals(Na.ZERO)) {
        continue;
    }
    size += coin.size();
    if (i == 127) {
        size += 1;
    }
    /**
     * UTXO
     */
    if (signType != 3) {
        if ((signType & 0x01) == 0 && coin.getTempOwner().length == 23) {
            signType = (byte) (signType | 0x01);
            size += P2PHKSignature.SERIALIZE_LENGTH;
        } else if ((signType & 0x02) == 0 && coin.getTempOwner().length != 23) {
            signType = (byte) (signType | 0x02);
            size += P2PHKSignature.SERIALIZE_LENGTH;
        }
    }
}
if (size > targetSize * txNum) { //txsize tx
    size += txSize;
    txNum++;
    signType = 0;
}
}
Na fee = TransactionFeeCalculator.getFee(size, price);
return Result.getSuccess().setData(fee);

```

```

    } catch (Exception e) {
        return Result.getFailed(TransactionErrorCode.DATA_ERROR);
    } finally {
        lock.unlock();
    }
}

```

@Override

```

public Result getAvailableTotalUTXO(byte[] address) {
    Map<String, Object> map = new HashMap<>();
    List<Coin> coinList = balanceManager.getCoinListByAddress(address);
    Na max = Na.ZERO;
    List<Coin> coins = new ArrayList<>();
    for (int i = 0; i < coinList.size(); i++) {
        Coin coin = coinList.get(i);
        if (!coin.usable()) {
            continue;
        }
        if (coin.getNa().equals(Na.ZERO)) {
            continue;
        }
        max = max.add(coin.getNa());
        coins.add(coin);
    }
    map.put("size", coins.size());
    map.put("max", max.getValue());
    return Result.getSuccess().setData(map);
}

```

@Override

```

public Result transfer(byte[] from, byte[] to, Na values, String password, byte[] remark, Na price)
{
    try {
        Result<Account> accountResult = accountService.getAccount(from);
        if (accountResult.isFailed()) {
            return accountResult;
        }
        Account account = accountResult.getData();
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(password, "the password can not be empty");
            if (!account.validatePassword(password)) {

```



```

        return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
    }
}

// to
if (contractService.isContractAddress(to)) {
    return
Result.getFailed(ContractErrorCode.NON_CONTRACTUAL_TRANSACTION_NO_TRANSFER);
}

TransferTransaction tx = new TransferTransaction();
tx.setRemark(remark);
tx.setTime(TimeService.currentTimeMillis());
CoinData coinData = new CoinData();
//
Coin toCoin;
if (to[2] == NulsContext.P2SH_ADDRESS_TYPE) {
    Script scriptPubkey = SignatureUtil.createOutputScript(to);
    toCoin = new Coin(scriptPubkey.getProgram(), values);
} else {
    toCoin = new Coin(to, values);
}
coinData.setTo().add(toCoin);
if (price == null) {
    price = TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES;
}
CoinDataResult coinDataResult = getCoinData(from, values, tx.size() + coinData.size(),
price);
if (!coinDataResult.isEnough()) {
    return Result.getFailed(AccountLedgerErrorCode.INSUFFICIENT_BALANCE);
}
coinData.setFrom(coinDataResult.getCoinList());
if (coinDataResult.getChange() != null) {
    coinData.setTo().add(coinDataResult.getChange());
}
tx.setCoinData(coinData);
tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
//
List<ECKey> signEckey = new ArrayList<>();
List<ECKey> scriptEckey = new ArrayList<>();
ECKey eckey = account.getEcKey(password);
//1

```

```

        if ((coinDataResult.getSignType() & 0x01) == 0x01) {
            signEckey.add(eckey);
        }
        //1
        if ((coinDataResult.getSignType() & 0x02) == 0x02) {
            scriptEckey.add(eckey);
        }
        SignatureUtil.createTransactionSignature(tx, scriptEckey, signEckey);

        //
        Result saveResult = verifyAndSaveUnconfirmedTransaction(tx);
        if (saveResult.isFailed()) {
            if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                //()
                Result rs = getMaxAmountOfOnce(from, tx, price);
                if (rs.isSuccess()) {
                    Na maxAmount = (Na) rs.getData();
                    rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
                    rs.setMsg(rs.getMsg() + maxAmount.toDouble());
                }
                return rs;
            }
            return saveResult;
        }
        // transactionService.newTx(tx);
        Result sendResult = transactionService.broadcastTx(tx);
        if (sendResult.isFailed()) {
            this.deleteTransaction(tx);
            return sendResult;
        }
        return Result.getSuccess().setData(tx.getHash().getDigestHex());
    } catch (IOException e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode());
    }
}

```

@Override

```
public Result dapp(byte[] from, String password, byte[] data, byte[] remark) {
    Result<Account> accountResult = accountService.getAccount(from);
    if (accountResult.isFailed()) {
        return accountResult;
    }
    Account account = accountResult.getData();
    if (account.isEncrypted() && account.isLocked()) {
        AssertUtil.canNotEmpty(password, "the password can not be empty");
        if (!account.validatePassword(password)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    }
}

DataTransaction tx = new DataTransaction();
tx.setRemark(remark);
tx.setTime(TimeService.currentTimeMillis());
LogicData logicData = new LogicData(data);
tx.setTxData(logicData);
CoinData coinData = new CoinData();
try {
    CoinDataResult coinDataResult = getCoinData(from, Na.ZERO, tx.size() + coinData.size(),
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
    if (!coinDataResult.isEnough()) {
        return Result.getFailed(AccountLedgerErrorCode.INSUFFICIENT_BALANCE);
    }

    coinData.setFrom(coinDataResult.getCoinList());
    if (coinDataResult.getChange() != null) {
        coinData.getTo().add(coinDataResult.getChange());
    }
    tx.setCoinData(coinData);
    tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));

    //
    List<ECKey> signEckey = new ArrayList<>();
    List<ECKey> scriptEckey = new ArrayList<>();
    ECKey eckey = account.getEcKey(password);
    //1
    if ((coinDataResult.getSignType() & 0x01) == 0x01) {
        signEckey.add(eckey);
    }
}
```

```

//1
if ((coinDataResult.getSignType() & 0x02) == 0x02) {
    scriptEckey.add(eckey);
}
SignatureUtil.createTransactionSignature(tx, scriptEckey, signEckey);
//
Result saveResult = verifyAndSaveUnconfirmedTransaction(tx);
if (saveResult.isFailed()) {
    if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
        //()
        Result rs = getMaxAmountOfOnce(from, tx,
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
        if (rs.isSuccess()) {
            Na maxAmount = (Na) rs.getData();
            rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
            rs.setMsg(rs.getMsg() + maxAmount.toDouble());
        }
        return rs;
    }
    return saveResult;
}
Result sendResult = transactionService.broadcastTx(tx);
if (sendResult.isFailed()) {
    this.deleteTransaction(tx);
    return sendResult;
}
return Result.getSuccess().setData(tx.getHash().getDigestHex());
} catch (NulsException e) {
    Log.error(e);
    return Result.getFailed(e.getErrorCode());

} catch (IOException e) {
    Log.error(e);
    return Result.getFailed(KernelErrorCode.IO_ERROR);
}
}

```

@Override

```

public Result multipleAddressTransfer(List<MultipleAddressTransferModel> fromList,
List<MultipleAddressTransferModel> toList, String password, Na amount, String remark, Na price)

```

```

{
    try {
        for (MultipleAddressTransferModel from : fromList) {
            Result<Account> accountResult = accountService.getAccount(from.getAddress());
            if (accountResult.isFailed()) {
                return accountResult;
            }
            Account account = accountResult.getData();
            if (account.isEncrypted() && account.isLocked()) {
                AssertUtil.canNotEmpty(password, "the password can not be empty");
                if (!account.validatePassword(password)) {
                    return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
                }
            }
        }
        for (MultipleAddressTransferModel to : toList) {
            // to
            if (contractService.isContractAddress(to.getAddress())) {
                return
Result.getFailed(ContractErrorCode.NON_CONTRACTUAL_TRANSACTION_NO_TRANSFER);
            }
        }
        TransferTransaction tx = new TransferTransaction();
        if (StringUtils.isNotBlank(remark)) {
            try {
                tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
            } catch (UnsupportedEncodingException e) {
                Log.error(e);
            }
        }
        tx.setTime(TimeService.currentTimeMillis());
        CoinData coinData = new CoinData();
        for (MultipleAddressTransferModel to : toList) {
            //
            Coin toCoin = null;
            if (to.getAddress()[2] == 3) {
                Script scriptPubkey = SignatureUtil.createOutputScript(to.getAddress());
                toCoin = new Coin(scriptPubkey.getProgram(), Na.valueOf(to.getAmount()));
            } else {
                toCoin = new Coin(to.getAddress(), Na.valueOf(to.getAmount()));
            }
            coinData.getTo().add(toCoin);
        }
    }
}

```

```

}
if (price == null) {
    price = TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES;
}

```

```

CoinDataResult coinDataResult = getCoinDataMultipleAdresses(fromList, amount, tx.size()
+ coinData.size(), price);

```

```

List<Coin> fromCoinList = new ArrayList<>(); // from
List<Coin> changeCoinList = new ArrayList<>();
if (!coinDataResult.isEnough()) { // utxo
    return Result.getFailed(AccountLedgerErrorCode.INSUFFICIENT_BALANCE);
}
fromCoinList.addAll(coinDataResult.getCoinList()); // list
if (coinDataResult.getChange() != null) {
    changeCoinList.add(coinDataResult.getChange());
}
coinData.setFrom(fromCoinList); // from utxo list
coinData.getTo().addAll(changeCoinList); //
tx.setCoinData(coinData);
tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
//
List<EKey> signEkeys = new ArrayList<>();
List<EKey> scriptEkeys = new ArrayList<>();
for (int index = 0; index < fromList.size(); index++) {
    Result<Account> accountResult =
accountService.getAccount(fromList.get(index).getAddress());
    Account account = accountResult.getData();
    // EKey
    EKey ecKey = account.getEcKey(password);
    // CoinDataResult coinDataResult = coinDataResult;
    // 1
    if ((coinDataResult.getSignType() & 0x01) == 0x01) {
        signEkeys.add(ecKey);
    }
    // 1
    if ((coinDataResult.getSignType() & 0x02) == 0x02) {
        scriptEkeys.add(ecKey);
    }
}
SignatureUtil.createTransactionSignature(tx, scriptEkeys, signEkeys);
//

```

```

        Result saveResult = verifyAndSaveUnconfirmedTransaction(tx);
        if (saveResult.isFailed()) {
            for (MultipleAddressTransferModel from : fromList) {
                if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                    //()
                    Na maxAmount = getMaxAmountOfOnce(from.getAddress(), tx, price).getData();
                    Result rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
                    rs.setMsg(rs.getMsg() + maxAmount.toDouble());
                    return rs;
                }
            }
            return saveResult;
        }
        Result sendResult = transactionService.broadcastTx(tx);
        if (sendResult.isFailed()) {
            this.deleteTransaction(tx);
            return sendResult;
        }
        return Result.getSuccess().setData(tx.getHash().getDigestHex());
    } catch (IOException e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode());
    }
}

```

```

private CoinDataResult getCoinDataMultipleAddresses(List<MultipleAddressTransferModel>
fromList, Na amount, int size, Na price) {

```

```

//utxo
List<Coin> coinListUTXO = new ArrayList<>();
for (int j = 0; j < fromList.size(); j++) {
    byte[] address = fromList.get(j).getAddress();
    List<Coin> coinList = balanceManager.getCoinListByAddress(address);
    coinListUTXO.addAll(coinList);
}
if (null == price) {
    throw new NulsRuntimeException(KernelErrorCode.PARAMETER_ERROR);
}

```

```

}
lock.lock();
try {
    CoinDataResult coinDataResult = new CoinDataResult();
    coinDataResult.setEnough(false);

    coinListUTXO = coinListUTXO.stream()
        .filter(coin1 -> coin1.usable() && !Na.ZERO.equals(coin1.getNa()))
        .sorted(CoinComparator.getInstance())
        .collect(Collectors.toList());

    if (coinListUTXO.isEmpty()) {
        return coinDataResult;
    }
    List<Coin> coins = new ArrayList<>();
    Na values = Na.ZERO;
    //
    //utxo
    byte[] changeAddress = null;
    for (int i = 0; i < coinListUTXO.size(); i++) {
        Coin coin = coinListUTXO.get(i);

        coins.add(coin);
        size += coin.size();
        if (i == 127) {
            size += 1;
        }
    }
    //
    Na fee = TransactionFeeCalculator.getFee(size, price);
    values = values.add(coin.getNa());

    /**
     * UTXO
     */
    int signType = coinDataResult.getSignType();
    if (signType != 3) {
        if ((signType & 0x01) == 0 && coin.getTempOwner().length == 23) {
            coinDataResult.setSignType((byte) (signType | 0x01));
            size += P2PHKSignature.SERIALIZE_LENGTH;
        } else if ((signType & 0x02) == 0 && coin.getTempOwner().length != 23) {
            coinDataResult.setSignType((byte) (signType | 0x02));
            size += P2PHKSignature.SERIALIZE_LENGTH;
        }
    }
}

```



```

    }
}

//
if (values.isGreaterThan(amount.add(fee))) {
    changeAddress = coin.getTempOwner();
    Na change = values.subtract(amount.add(fee));
    Coin changeCoin = new Coin();

    if (changeAddress[2] == NulsContext.P2SH_ADDRESS_TYPE) {
changeCoin.setOwner(SignatureUtil.createOutputScript(changeAddress).getProgram());
    } else {
        changeCoin.setOwner(changeAddress);
    }
    changeCoin.setNa(change);
    fee = TransactionFeeCalculator.getFee(size + changeCoin.size(), price);
    if (values.isLessThan(amount.add(fee))) {
        continue;
    }
    changeCoin.setNa(values.subtract(amount.add(fee)));
    if (!changeCoin.getNa().equals(Na.ZERO)) {
        coinDataResult.setChange(changeCoin);
    }
}
coinDataResult.setFee(fee);
if (values.isGreaterOrEquals(amount.add(fee))) {
    coinDataResult.setEnough(true);
    coinDataResult.setCoinList(coins);
    break;
}
}
return coinDataResult;
} finally {
    lock.unlock();
}

}

```

@Override

```

public Result changeWhole(byte[] address, String password, Na price) {
    try {
        Result<Account> accountResult = accountService.getAccount(address);
    }
}

```

```

    if (accountResult.isFailed()) {
        return accountResult;
    }
    Account account = accountResult.getData();
    if (account.isEncrypted() && account.isLocked()) {
        AssertUtil.canNotEmpty(password, "the password can not be empty");
        if (!account.validatePassword(password)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    }
    // to
    if (contractService.isContractAddress(address)) {
        return
Result.getFailed(ContractErrorCode.NON_CONTRACTUAL_TRANSACTION_NO_TRANSFER);
    }
    TransferTransaction tx = null;
    boolean flag = true;
    while (flag) {
        tx = getChangeWholeTxInfoList(address, account, password, price);
        //
        if (null != tx) {
            Result saveResult = verifyAndSaveUnconfirmedTransaction(tx);
            if (saveResult.isFailed()) {
                if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                    //()
                    Result rs = getMaxAmountOfOnce(address, tx, price);
                    if (rs.isSuccess()) {
                        Na maxAmount = (Na) rs.getData();
                        rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
                        rs.setMsg(rs.getMsg() + maxAmount.toDouble());
                    }
                    return rs;
                }
            }
            return saveResult;
        }
    }
    Result sendResult = transactionService.broadcastTx(tx);
    if (sendResult.isFailed()) {
        this.deleteTransaction(tx);
        return sendResult;
    }
}

```

```

        Result available = getAvailableTotalUTXO(address);//
        Map<String, Object> map = (Map<String, Object>) available.getData();
        int size = (int) map.get("size");
        if (size < AccountConstant.MIM_COUNT) { //20
            flag = false;
        }
        } else {
            flag = false;
        }
    }
    return Result.getSuccess().setData(tx.getHash().getDigestHex());
} catch (Exception e) {
    Log.error(e);
    Log.error("");
    return Result.getFailed();
}
}

```

```

private TransferTransaction getChangeWholeTxInfoList(byte[] address, Account account, String
password, Na price) {
    List<Coin> coinList = balanceManager.getCoinListByAddress(address);//
    TransferTransaction tx = new TransferTransaction();
    changeWholeLock.lock();
    try {
        tx.setTime(TimeService.currentTimeMillis());
        //coindatato38to coin
        int size = tx.size() + 38;
        //sizecoindatafrom
        int targetSize = TxMaxSizeValidator.MAX_TX_SIZE - size;
        if (coinList.isEmpty()) {
            return null;
        }
        //
        Collections.sort(coinList, CoinComparatorDesc.getInstance());
        Na max = Na.ZERO;
        List<Coin> coins = new ArrayList<>();
        byte signType = 0;
        for (int i = 0; i < coinList.size(); i++) {
            Coin coin = coinList.get(i);
            if (!coin.usable()) {
                continue;
            }

```

```

        if (coin.getNa().equals(Na.ZERO)) {
            continue;
        }
        size += coin.size();
        if (i == 127) {
            size += 1;
        }
        if (size > targetSize) {
            break;
        }
        coins.add(coin);
    /**
     * UTXO
     * */
    if (signType != 3) {
        if ((signType & 0x01) == 0 && coin.getTempOwner().length == 23) {
            signType = (byte) (signType | 0x01);
            size += P2PHKSignature.SERIALIZE_LENGTH;
        } else if ((signType & 0x02) == 0 && coin.getTempOwner().length != 23) {
            signType = (byte) (signType | 0x02);
            size += P2PHKSignature.SERIALIZE_LENGTH;
        }
    }
    max = max.add(coin.getNa());
}
Na fee = TransactionFeeCalculator.getFee(size, price);
max = max.subtract(fee);
CoinData coinData = new CoinData();
Coin toCoin = new Coin(address, max);
coinData.setTo().add(toCoin);
coinData.setFrom(coins);
tx.setCoinData(coinData);
tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash())); //setHash

//
List<ECKey> signEckey = new ArrayList<>();
List<ECKey> scriptEckey = new ArrayList<>();
ECKey eckey = account.getEcKey(password);
//1
if ((signType & 0x01) == 0x01) {
    signEckey.add(eckey);
}

```

```

//1
if ((signType & 0x02) == 0x02) {
    scriptEckey.add(eckey);
}
SignatureUtil.createTransactionSignature(tx, scriptEckey, signEckey);
return tx;
} catch (Exception e) {
    Log.error(e);
    return null;
} finally {
    changeWholeLock.unlock();
}
}

```

@Override

```

public Result transferFee(byte[] from, byte[] to, Na values, String remark, Na price) {
    Result<Account> accountResult = accountService.getAccount(from);
    if (accountResult.isFailed()) {
        return accountResult;
    }
    TransferTransaction tx = new TransferTransaction();
    try {
        tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
    } catch (UnsupportedEncodingException e) {
        return Result.getFailed(LedgerErrorCode.PARAMETER_ERROR);
    }
    tx.setTime(TimeService.currentTimeMillis());
    CoinData coinData = new CoinData();
    Script scriptPubkey = SignatureUtil.createOutputScript(to);
    Coin toCoin = new Coin(scriptPubkey.getProgram(), values);
    coinData.getTo().add(toCoin);
    tx.setCoinData(coinData);
    Na fee = getTxFee(from, values, tx.size() + P2PHKSignature.SERIALIZE_LENGTH, price);
    Result result = Result.getSuccess().setData(fee);
    return result;
}

```

@Override

```

public Result createTransaction(List<Coin> inputs, List<Coin> outputs, byte[] remark) {
    TransferTransaction tx = new TransferTransaction();
    CoinData coinData = new CoinData();

```

```

coinData.setTo(outputs);
coinData.setFrom(inputs);
tx.setRemark(remark);

tx.setCoinData(coinData);
tx.setTime(TimeService.currentTimeMillis());
//
int size = tx.size() + P2PHKSignature.SERIALIZE_LENGTH;
Na minFee = TransactionFeeCalculator.getTransferFee(size);
//inputsoutputs
Na fee = Na.ZERO;
for (Coin coin : tx.getCoinData().getFrom()) {
    fee = fee.add(coin.getNa());
}
for (Coin coin : tx.getCoinData().getTo()) {
    fee = fee.subtract(coin.getNa());
}
if (fee.isLessThan(minFee)) {
    return Result.getFailed(LedgerErrorCode.FEE_NOT_RIGHT);
}

try {
    String txHex = Hex.encode(tx.serialize());
    return Result.getSuccess().setData(txHex);
} catch (IOException e) {
    Log.error(e);
    return Result.getFailed(KernelErrorCode.IO_ERROR);
}
}

@Override
public Transaction signTransaction(Transaction tx, ECKey ecKey) throws IOException {
    List<ECKey> pubEckeyes = new ArrayList<>();
    pubEckeyes.add(ecKey);
    SignatureUtil.createTransactionSignature(tx, null, pubEckeyes);
    return tx;
}

@Override
public Result broadcast(Transaction tx) {
    return transactionService.broadcastTx(tx);
}

```

```

/**
 *
 */
@Override
public Result importLedgerByAddress(String address) {
    if (address == null || !AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }

    // NRC20
    contractService.initAllTokensByAccount(address);

    byte[] addressBytes = null;
    try {
        addressBytes = AddressTool.getAddress(address);
    } catch (Exception e) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }

    long start = 0;
    long end = NulsContext.getInstance().getBestHeight();
    while (start <= end) {
        for (long i = start; i <= end; i++) {
            List<Transaction> txs = blockService.getBlock(i, true).getData().getTxs();
            for (Transaction tx : txs) {
                importConfirmedTransaction(tx, addressBytes);
            }
        }
        start = end;
        end = NulsContext.getInstance().getBestHeight();
        if (start == end) {
            break;
        }
    }
    try {
        balanceManager.refreshBalance(addressBytes);
    } catch (Exception e) {
        Log.info(address);
    }
    return Result.getSuccess();
}

```

```
}
```

```
@Override
```

```
public Result<List<TransactionInfo>> getTxInfoList(byte[] address) {  
    return transactionInfoService.getTxInfoList(address);  
}
```

```
@Override
```

```
public Result<List<Coin>> getLockedUtxo(byte[] address) {  
    Result<List<Coin>> result = new Result<>();  
    result.setSuccess(true);  
    List<Coin> coinList = balanceManager.getCoinListByAddress(address);  
    List<Coin> lockCoinList = new ArrayList<>();  
    for (Coin coin : coinList) {  
        if (coin != null && !coin.usable()) {  
            lockCoinList.add(coin);  
        }  
    }  
    result.setData(lockCoinList);  
    return result;  
}
```

```
@Override
```

```
public Result<Integer> deleteUnconfirmedTx(byte[] address) {  
    Result result = getAllUnconfirmedTransaction();  
    if (result.getData() == null) {  
        return Result.getSuccess().setData(new Integer(0));  
    }  
    List<Transaction> txs = (List<Transaction>) result.getData();  
    int i = 0;  
    try {  
        for (Transaction tx : txs) {  
            if (SignatureUtil.containsAddress(tx, address)) {  
                unconfirmedTransactionStorageService.deleteUnconfirmedTx(tx.getHash());  
                localUtxoService.deleteUtxoOfTransaction(tx);  
                i++;  
            }  
        }  
    } catch (NulsException e) {  
        Log.error(e);  
        return Result.getFailed(e.getErrorCode());  
    }  
}
```



```

        return Result.getSuccess().setData(new Integer(i));
    }

protected Result<Integer> importConfirmedTransaction(Transaction tx, byte[] address) {

    if (!AccountLegerUtils.isTxRelatedToAddress(tx, address)) {
        return Result.getSuccess().setData(new Integer(0));
    }

    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    txInfoPo.setStatus(TransactionInfo.CONFIRMED);

    List<byte[]> addresses = new ArrayList<>();
    addresses.add(address);
    Result result = transactionInfoService.saveTransactionInfo(txInfoPo, addresses);

    if (result.isFailed()) {
        return result;
    }
    result = localUtxoService.saveUtxoForAccount(tx, address);
    if (result.isFailed()) {
        transactionInfoService.deleteTransactionInfo(txInfoPo);
    }
    return result;
}

protected List<Transaction> filterLocalTransaction(List<Transaction> txs) {
    List<Transaction> resultTxs = new ArrayList<>();
    if (txs == null || txs.size() == 0) {
        return resultTxs;
    }
    Collection<Account> localAccountList = accountService.getAccountList().getData();
    if (localAccountList == null || localAccountList.size() == 0) {
        return resultTxs;
    }
    Transaction tmpTx;
    for (int i = 0; i < txs.size(); i++) {
        tmpTx = txs.get(i);
        if (AccountLegerUtils.isLocalTransaction(tmpTx)) {
            resultTxs.add(tmpTx);
        }
    }
}

```

```

    return resultTx;
}

```

@Override

```

public Result unlockCoinData(Transaction tx, long newLockTime) {
    List<byte[]> addresses = AccountLegerUtils.getRelatedAddresses(tx);
    if (addresses == null || addresses.size() == 0) {
        return Result.getSuccess();
    }
    byte status = TransactionInfo.CONFIRMED;
    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    txInfoPo.setStatus(status);

    List<byte[]> addresses1 = localUtxoService.unlockCoinData(tx, newLockTime).getData();
    for (byte[] address : addresses1) {
        balanceManager.refreshBalance(address);
    }
    return Result.getSuccess();
}

```

@Override

```

public Result rollbackUnlockTxCoinData(Transaction tx) {
    List<byte[]> addresses = AccountLegerUtils.getRelatedAddresses(tx);
    if (addresses == null || addresses.size() == 0) {
        return Result.getSuccess();
    }
    byte status = TransactionInfo.CONFIRMED;
    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    txInfoPo.setStatus(status);

    List<byte[]> addresses1 = localUtxoService.rollbackUnlockTxCoinData(tx).getData();
    for (byte[] address : addresses1) {
        balanceManager.refreshBalance(address);
    }

    CoinData coinData = tx.getCoinData();
    if (coinData != null) {
        List<Coin> froms = tx.getCoinData().getFrom();
        for (Coin from : froms) {
            usedTxSets.remove(LedgerUtil.asString(from.getOwner()));
        }
    }
}

```

```
    return Result.getSuccess();
}
```

@Override

```
public Result<Transaction> getUnconfirmedTransaction(NulsDigestData hash) {
    return unconfirmedTransactionStorageService.getUnconfirmedTx(hash);
}
```

/\*\*

```
 * A transfers NULS to B
 *
 * @param fromAddr address of A
 * @param signAddr address of B
 * @param values    NULS amount
 * @param password password of A
 * @param remark    remarks of transaction
 * @param price     Unit price of fee
 * @param pubkeys
 * @param m
 * @return Result
 */
```

@Override

```
public Result transferP2SH(byte[] fromAddr, byte[] signAddr,
    List<MultipleAddressTransferModel> outputs,
        Na values, String password, String remark, Na price,
        List<String> pubkeys, int m, String txdata) {
    try {
        Result<Account> accountResult = accountService.getAccount(signAddr);
        if (accountResult.isFailed()) {
            return accountResult;
        }
        Account account = accountResult.getData();
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(password, "the password can not be empty");
            if (!account.validatePassword(password)) {
                return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
            }
        }
        TransferTransaction tx = new TransferTransaction();
        TransactionSignature transactionSignature = new TransactionSignature();
        List<P2PHKSignature> p2PHKSignatures = new ArrayList<>();
        List<Script> scripts = new ArrayList<>();
```

```

//txdata
if (txdata == null || txdata.trim().length() == 0) {
    if (StringUtils.isNotBlank(remark)) {
        try {
            tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
        } catch (UnsupportedEncodingException e) {
            Log.error(e);
        }
    }
}
Script redeemScript = ScriptBuilder.createNulsRedeemScript(m, pubkeys);
tx.setTime(TimeService.currentTimeMillis());
CoinData coinData = new CoinData();
for (MultipleAddressTransferModel to : outputs) {
    //
    Coin toCoin = null;
    if (to.getAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
        Script scriptPubkey = SignatureUtil.createOutputScript(to.getAddress());
        toCoin = new Coin(scriptPubkey.getProgram(), Na.valueOf(to.getAmount()));
    } else {
        toCoin = new Coin(to.getAddress(), Na.valueOf(to.getAmount()));
    }
    coinData.getTo().add(toCoin);
}
if (price == null) {
    price = TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES;
}
//m*+
int scriptSignLenth = redeemScript.getProgram().length + m * 72;
CoinDataResult coinDataResult = getMutilCoinData(fromAddr, values, tx.size() +
coinData.size() + scriptSignLenth, price);
if (!coinDataResult.isEnough()) {
    return Result.getFailed(AccountLedgerErrorCode.INSUFFICIENT_BALANCE);
}
coinData.setFrom(coinDataResult.getCoinList());
if (coinDataResult.getChange() != null) {
    coinData.getTo().add(coinDataResult.getChange());
}
tx.setCoinData(coinData);
tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
//
scripts.add(redeemScript);
transactionSignature.setScripts(scripts);

```

```

    }
    //txdata
    else {
        byte[] txByte = Hex.decode(txdata);
        tx.parse(new NulsByteBuffer(txByte));
        transactionSignature.parse(new NulsByteBuffer(tx.getTransactionSignature()));
        p2PHKSignatures = transactionSignature.getP2PHKSignatures();
        scripts = transactionSignature.getScripts();
    }
    //
    P2PHKSignature p2PHKSignature = new P2PHKSignature();
    ECKey eckey = account.getEcKey(password);
    p2PHKSignature.setPublicKey(eckey.getPubKey());
    //hash
    p2PHKSignature.setSignData(accountService.signDigest(tx.getHash().getDigestBytes(),
eckey));
    p2PHKSignatures.add(p2PHKSignature);

    //M
    if (p2PHKSignatures.size() == SignatureUtil.getM(scripts.get(0))) {
        //P2PHKSignatures
        Collections.sort(p2PHKSignatures, P2PHKSignature.PUBKEY_COMPARATOR);
        //P2PHKSignatures
        List<byte[]> signatures = new ArrayList<>();
        for (P2PHKSignature p2PHKSignatureTemp : p2PHKSignatures) {
            signatures.add(p2PHKSignatureTemp.getSignData().getSignBytes());
        }
        transactionSignature.setP2PHKSignatures(null);
        Script scriptSign = ScriptBuilder.createNulsP2SHMultiSigInputScript(signatures,
scripts.get(0));
        transactionSignature.getScripts().clear();
        transactionSignature.getScripts().add(scriptSign);
        tx.setTransactionSignature(transactionSignature.serialize());
        //
        Result saveResult = verifyAndSaveUnconfirmedTransaction(tx);
        if (saveResult.isFailed()) {
            if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                //()
                Result rs = getMaxAmountOfOnce(fromAddr, tx, price);
                if (rs.isSuccess()) {

```

```

        Na maxAmount = (Na) rs.getData();
        rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
        rs.setMsg(rs.getMsg() + maxAmount.toDouble());
    }
    return rs;
}
return saveResult;
}
Result sendResult = transactionService.broadcastTx(tx);
if (sendResult.isFailed()) {
    this.deleteTransaction(tx);
    return sendResult;
}
return Result.getSuccess().setData(tx.getHash().getDigestHex());
}
//
else {
    transactionSignature.setP2PHKSignatures(p2PHKSignatures);
    tx.setTransactionSignature(transactionSignature.serialize());
    return Result.getSuccess().setData(Hex.encode(tx.serialize()));
}
} catch (IOException e) {
    Log.error(e);
    return Result.getFailed(KernelErrorCode.IO_ERROR);
} catch (NulsException e) {
    Log.error(e);
    return Result.getFailed(e.getErrorCode());
}
}
}

```

/\*\*

\* A transfers NULS to B

\*

\* @param fromAddr

\* @param signAddr

\* @param outputs

\* @param password password of A

\* @param remark remarks of transaction

\* @return Result

\*/

@Override

```

public Result createP2shTransfer(String fromAddr, String signAddr,
List<MultipleAddressTransferModel> outputs, String password, String remark) {
    try {
        Result<Account> accountResult = accountService.getAccount(signAddr);
        if (accountResult.isFailed()) {
            return accountResult;
        }
        Account account = accountResult.getData();
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(password, "the password can not be empty");
            if (!account.validatePassword(password)) {
                return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
            }
        }
        TransferTransaction tx = new TransferTransaction();
        TransactionSignature transactionSignature = new TransactionSignature();
        List<Script> scripts = new ArrayList<>();
        Result<MultiSigAccount> result = accountService.getMultiSigAccount(fromAddr);
        MultiSigAccount multiSigAccount = result.getData();
        Script redeemScript = getRedeemScript(multiSigAccount);
        if (redeemScript == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
        }
        tx.setTime(TimeService.currentTimeMillis());
        if (StringUtils.isNotBlank(remark)) {
            try {
                tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
            } catch (UnsupportedEncodingException e) {
                Log.error(e);
            }
        }
        CoinData coinData = new CoinData();
        Na values = Na.ZERO;
        for (MultipleAddressTransferModel to : outputs) {
            //
            Coin toCoin = null;
            values = values.add(Na.valueOf(to.getAmount()));
            if (to.getAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
                Script scriptPubkey = SignatureUtil.createOutputScript(to.getAddress());
                toCoin = new Coin(scriptPubkey.getProgram(), Na.valueOf(to.getAmount()));
            } else {
                toCoin = new Coin(to.getAddress(), Na.valueOf(to.getAmount()));
            }
        }
    }
}

```

```

        }
        coinData.getTo().add(toCoin);
    }
    //m*+
    int scriptSignLenth = redeemScript.getProgram().length + ((int) multiSigAccount.getM()) *
72;

    CoinDataResult coinDataResult = getMutilCoinData(AddressTool.getAddress(fromAddr),
values, tx.size() + coinData.size() + scriptSignLenth,
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
    if (!coinDataResult.isEnough()) {
        return Result.getFailed(AccountLedgerErrorCode.INSUFFICIENT_BALANCE);
    }
    coinData.setFrom(coinDataResult.getCoinList());
    if (coinDataResult.getChange() != null) {
        coinData.getTo().add(coinDataResult.getChange());
    }
    tx.setCoinData(coinData);
    tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
    //
    scripts.add(redeemScript);
    transactionSignature.setScripts(scripts);
    return txMultiProcess(tx, transactionSignature, account, password);
} catch (IOException e) {
    Log.error(e);
    return Result.getFailed(KernelErrorCode.IO_ERROR);
} catch (NulsException e) {
    Log.error(e);
    return Result.getFailed(e.getErrorCode());
} catch (Exception e) {
    Log.error(e);
    return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
}
}
}

```

@Override

```

public CoinDataResult getMutilCoinData(byte[] address, Na amount, int size, Na price) {
    if (null == price) {
        throw new NulsRuntimeException(KernelErrorCode.PARAMETER_ERROR);
    }
    lock.lock();
    try {
        CoinDataResult coinDataResult = new CoinDataResult();
    }
}

```



```

coinDataResult.setEnough(false);
List<Coin> coinList = ledgerService.getAllUtxo(address);

coinList = coinList.stream()
    .filter(coin1 -> coin1.usable() && !Na.ZERO.equals(coin1.getNa()))
    .sorted(CoinComparator.getInstance())
    .collect(Collectors.toList());

if (coinList.isEmpty()) {
    return coinDataResult;
}
List<Coin> coins = new ArrayList<>();
Na values = Na.ZERO;
//
for (int i = 0; i < coinList.size(); i++) {
    Coin coin = coinList.get(i);
    coins.add(coin);
    size += coin.size();
    if (i == 127) {
        size += 1;
    }
}
//
Na fee = TransactionFeeCalculator.getFee(size, price);
values = values.add(coin.getNa());

//
if (values.isGreaterThan(amount.add(fee))) {
    Na change = values.subtract(amount.add(fee));
    Coin changeCoin = new Coin();
    changeCoin.setOwner(SignatureUtil.createOutputScript(address).getProgram());
    //changeCoin.setOwner(address);
    changeCoin.setNa(change);
    fee = TransactionFeeCalculator.getFee(size + changeCoin.size(), price);
    if (values.isLessThan(amount.add(fee))) {
        continue;
    }
    changeCoin.setNa(values.subtract(amount.add(fee)));
    if (!changeCoin.getNa().equals(Na.ZERO)) {
        coinDataResult.setChange(changeCoin);
    }
}
}
coinDataResult.setFee(fee);

```

```

        if (values.isGreaterOrEquals(amount.add(fee))) {
            coinDataResult.setEnough(true);
            coinDataResult.setCoinList(coins);
            break;
        }
    }
    return coinDataResult;
} finally {
    lock.unlock();
}
}

```

@Override

```

public Result<Na> getMultiMaxAmountOfOnce(byte[] address, Transaction tx, Na price, int
signSize) {
    lock.lock();
    try {
        tx.getCoinData().setFrom(null);
        int txSize = tx.size();
        //tosize
        for (Coin coin : tx.getCoinData().getTo()) {
            txSize += coin.size();
        }
        //sizecoindatafrom
        if (tx.getTransactionSignature() == null || tx.getTransactionSignature().length == 0) {
            txSize += signSize;
        }
        int targetSize = TxMaxSizeValidator.MAX_TX_SIZE - txSize;
        List<Coin> coinList = ledgerService.getAllUtxo(address);
        ;
        if (coinList.isEmpty()) {
            return Result.getSuccess().setData(Na.ZERO);
        }
        Collections.sort(coinList, CoinComparator.getInstance());
        Na max = Na.ZERO;
        int size = 0;
        //
        for (int i = 0; i < coinList.size(); i++) {
            Coin coin = coinList.get(i);
            if (!coin.usable()) {
                continue;
            }

```

```

        if (coin.getNa().equals(Na.ZERO)) {
            continue;
        }
        size += coin.size();
        if (i == 127) {
            size += 1;
        }
        if (size > targetSize) {
            break;
        }
        max = max.add(coin.getNa());
    }
    Na fee = TransactionFeeCalculator.getFee(size, price);
    max = max.subtract(fee);
    return Result.getSuccess().setData(max);
} catch (Exception e) {
    return Result.getFailed(TransactionErrorCode.DATA_ERROR);
} finally {
    lock.unlock();
}
}

```

@Override

```

public Result txMultiProcess(Transaction tx, TransactionSignature transactionSignature,
Account account, String password) {
    try {
        List<P2PHKSignature> p2PHKSignatures = new ArrayList<>();
        if (transactionSignature.getP2PHKSignatures() != null &&
transactionSignature.getP2PHKSignatures().size() > 0) {
            p2PHKSignatures = transactionSignature.getP2PHKSignatures();
        }
        List<Script> scripts = transactionSignature.getScripts();
        //
        P2PHKSignature p2PHKSignature = new P2PHKSignature();
        ECKey eckey = account.getEcKey(password);
        p2PHKSignature.setPublicKey(eckey.getPubKey());
        //hash
        p2PHKSignature.setSignData(accountService.signDigest(tx.getHash().getDigestBytes(),
eckey));
        p2PHKSignatures.add(p2PHKSignature);
        //M
        if (p2PHKSignatures.size() == SignatureUtil.getM(scripts.get(0))) {

```

```

//P2PHKSignatures
p2PHKSignatures.sort(P2PHKSignature.PUBKEY_COMPARATOR);
//P2PHKSignatures
List<byte[]> signatures = new ArrayList<>();
for (P2PHKSignature p2PHKSignatureTemp : p2PHKSignatures) {
    signatures.add(p2PHKSignatureTemp.getSignData().getSignBytes());
}
transactionSignature.setP2PHKSignatures(null);
Script scriptSign = ScriptBuilder.createNulsP2SHMultiSigInputScript(signatures,
scripts.get(0));
transactionSignature.getScripts().clear();
transactionSignature.getScripts().add(scriptSign);
tx.setTransactionSignature(transactionSignature.serialize());
//
Result saveResult = verifyAndSaveUnconfirmedTransaction(tx);
if (saveResult.isFailed()) {
    return saveResult;
}
transactionService.newTx(tx);
Result sendResult = transactionService.broadcastTx(tx);
if (sendResult.isFailed()) {
    this.deleteTransaction(tx);
    return sendResult;
}
return Result.getSuccess().setData(tx.getHash().getDigestHex());
}
//
else {
    transactionSignature.setP2PHKSignatures(p2PHKSignatures);
    tx.setTransactionSignature(transactionSignature.serialize());
    return Result.getSuccess().setData(Hex.encode(tx.serialize()));
}
} catch (IOException e) {
    Log.error(e);
    return Result.getFailed(KernelErrorCode.IO_ERROR);
} catch (NulsException e) {
    Log.error(e);
    return Result.getFailed(e.getErrorCode());
}
}

```

@Override

```

public Script getRedeemScript(MultiSigAccount multiSigAccount) {
    try {
        List<String> pubkeys = new ArrayList<>();
        if (multiSigAccount.getPubKeyList() != null && multiSigAccount.getM() > 0
            && multiSigAccount.getPubKeyList().size() >= multiSigAccount.getM()) {
            for (byte[] pubkeyByte : multiSigAccount.getPubKeyList()) {
                pubkeys.add(Hex.encode(pubkeyByte));
            }
            return ScriptBuilder.createNulsRedeemScript((int) multiSigAccount.getM(), pubkeys);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

```

```
/**
```

```
* A transfers NULS to B
```

```
*
```

```
* @param signAddr
```

```
* @param password password of A
```

```
* @param txdata
```

```
* @return Result
```

```
*/
```

```
@Override
```

```

public Result signMultiTransaction(String signAddr, String password, String txdata) {
    try {
        Result<Account> accountResult = accountService.getAccount(signAddr);
        if (accountResult.isFailed()) {
            return accountResult;
        }
        Account account = accountResult.getData();
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(password, "the password can not be empty");
            if (!account.validatePassword(password)) {
                return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
            }
        }
        byte[] txByte = Hex.decode(txdata);
        Transaction tx = TransactionManager.getInstance(new NulsByteBuffer(txByte));
        TransactionSignature transactionSignature = new TransactionSignature();
        transactionSignature.parse(new NulsByteBuffer(tx.getTransactionSignature()));
    }
}

```

```

        return txMultiProcess(tx, transactionSignature, account, password);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode());
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
}

```

@Override

```

public Result getSignatureType(List<String> utxoList) {
    //utxotx_hashindex,tx_hashindexutxo
    byte signType = 0;
    for (String utxo : utxoList) {
        if ((signType & 0x01) == 0x01 && (signType & 0x02) == 0x02) {
            break;
        }
        byte[] owner = Hex.decode(utxo);
        Coin coin = ledgerService.getUtxo(owner);
        if (coin == null) {
            continue;
        }
        if (signType != 3) {
            if ((signType & 0x01) == 0 && coin.getOwner().length == 23) {
                signType = (byte) (signType | 0x01);
            } else if ((signType & 0x02) == 0 && coin.getTempOwner().length != 23) {
                signType = (byte) (signType | 0x02);
            }
        }
    }
    return Result.getSuccess().setData(signType);
}
}

```

11:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\main\java\io\nuls\account\ledger\base\service\impl\LocalUtxoServiceImpl.java  
\*/

```
package io.nuls.account.ledger.base.service.impl;
```

```
import io.nuls.account.ledger.base.service.LocalUtxoService;
import io.nuls.account.ledger.base.util.AccountLegerUtils;
```

```

import io.nuls.account.ledger.storage.service.LocalUtxoStorageService;
import io.nuls.account.ledger.storage.service.UnconfirmedTransactionStorageService;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.db.model.Entry;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.VarInt;
import io.nuls.ledger.service.LedgerService;

```

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

```

```

/**

```

```

 * author Facjas
 * date 2018/5/27.
 */

```

```

@Component

```

```

public class LocalUtxoServiceImpl implements LocalUtxoService {

```

```

    @Autowired

```

```

    private LedgerService ledgerService;

```

```

    @Autowired

```

```

    private LocalUtxoStorageService localUtxoStorageService;

```

```

    @Autowired

```

```

    UnconfirmedTransactionStorageService unconfirmedTransactionStorageService;

```

```

    @Override

```

```

    public Result saveUtxoForLocalAccount(Transaction tx) {

```

```

        List<byte[]> localAddressList = AccountLegerUtils.getLocalAddresses();

```

```

        return saveUtxoForAccount(tx, localAddressList);

```

```

    }

```

@Override

```
public Result saveUtxoForAccount(Transaction tx, byte[] addresses) {  
    List<byte[]> addressList = new ArrayList<>();  
    addressList.add(addresses);  
    return saveUtxoForAccount(tx, addressList);  
}
```

@Override

```
public Result saveUtxoForAccount(Transaction tx, List<byte[]> addressesList) {  
  
    if (tx == null || addressesList == null || addressesList.size() == 0) {  
        return Result.getFailed(KernelErrorCode.NULL_PARAMETER);  
    }  
  
    CoinData coinData = tx.getCoinData();  
  
    if (coinData != null) {  
        // delete - from  
        List<Coin> froms = coinData.getFrom();  
        List<byte[]> fromsList = new ArrayList<>();  
        byte[] fromSource;  
        byte[] utxoFromSource;  
        byte[] fromIndex;  
        Transaction sourceTx;  
        Coin fromOfFromCoin;  
        for (Coin from : froms) {  
            fromSource = from.getOwner();  
  
            fromOfFromCoin = from.getFrom();  
            if (fromOfFromCoin == null) {  
                utxoFromSource = new byte[tx.getHash().size()];  
                fromIndex = new byte[fromSource.length - utxoFromSource.length];  
  
                System.arraycopy(fromSource, 0, utxoFromSource, 0, tx.getHash().size());  
                System.arraycopy(fromSource, tx.getHash().size(), fromIndex, 0, fromIndex.length);  
  
                try {  
                    sourceTx =  
ledgerService.getTx(NulsDigestData.fromDigestHex(Hex.encode(utxoFromSource)));  
                } catch (Exception e) {  
                    continue;  
                }  
            }  
        }  
    }  
}
```



```

        if (sourceTx == null) {
            return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
        }
        int index = (int) new VarInt(fromIndex, 0).value;
        fromOfFromCoin = sourceTx.getCoinData().getTo().get(index);

        from.setFrom(fromOfFromCoin);
    }

    if(fromOfFromCoin == null) {
        Log.warn("from coin not found!");
        continue;
    }

    byte[] toAddress = fromOfFromCoin.getAddress();

    boolean addressIsMatch = false;
    for(byte[] addresses : addressesList) {
        if(Arrays.equals(toAddress, addresses)) {
            addressIsMatch = true;
            break;
        }
    }
    if(!addressIsMatch) {
        continue;
    }

    fromsList.add(fromSource);
}

// save utxo - to
List<Coin> tos = coinData.getTo();
List<Entry<byte[], byte[]>> toList = new ArrayList<>();
byte[] txHashBytes = null;
try {
    txHashBytes = tx.getHash().serialize();
} catch (IOException e) {
    throw new NulsRuntimeException(e);
}
byte[] outKey;
Coin to;
byte[] toAddress;

```

```

for (int i = 0, length = tos.size(); i < length; i++) {
    to = tos.get(i);
    toAddress = to.getAddress();

    boolean addressIsMatch = false;
    for(byte[] addresses : addressesList) {
        if(Arrays.equals(toAddress, addresses)) {
            addressIsMatch = true;
            break;
        }
    }
    if(!addressIsMatch) {
        continue;
    }

    try {
        outKey = ArraysTool.concatenate(txHashBytes, new VarInt(i).encode());
        toList.add(new Entry<>(outKey, to.serialize()));
    } catch (IOException e) {
        Log.error(e);
    }
}
Result result = localUtxoStorageService.batchSaveAndDeleteUTXO(toList, fromsList);
if (result.isFailed() || result.getData() == null || (int) result.getData() != toList.size() +
fromsList.size()) {
    return Result.getFailed();
}
return Result.getSuccess();
}

```

@Override

```

public Result deleteUtxoOfTransaction(Transaction tx) {
    if (tx == null) {
        return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
    }
}

```

```

CoinData coinData = tx.getCoinData();
if (coinData != null) {
    // delete utxo - to
    List<Coin> tos = coinData.getTo();
    List<byte[]> toList = new ArrayList<>();
}

```

```

byte[] outKey;
for (int i = 0, length = tos.size(); i < length; i++) {
    try {
        //if(!AccountLegerUtils.isLocalAccount(tos.get(i).getOwner()))
        if(!AccountLegerUtils.isLocalAccount(tos.get(i).getAddress())) {
            continue;
        }
        outKey = ArraysTool.concatenate(tx.getHash().serialize(), new VarInt(i).encode());
        toList.add(outKey);
    } catch (IOException e) {
        throw new NulsRuntimeException(e);
    }
}

```

```

// save - from
List<Coin> froms = coinData.getFrom();
List<Entry<byte[], byte[]>> fromList = new ArrayList<>();

```

```

byte[] fromSource;
Coin fromOfFromCoin;
byte[] utxoFromSource;
byte[] fromIndex;
Transaction sourceTx;
byte[] address;
for (Coin from : froms) {
    fromSource = from.getOwner();

    fromOfFromCoin = from.getFrom();
    if(fromOfFromCoin == null) {
        utxoFromSource = new byte[tx.getHash().size()];
        fromIndex = new byte[fromSource.length - utxoFromSource.length];
        System.arraycopy(fromSource, 0, utxoFromSource, 0, tx.getHash().size());
        System.arraycopy(fromSource, tx.getHash().size(), fromIndex, 0, fromIndex.length);
    }

```

```

        try {
            sourceTx =
ledgerService.getTx(NulsDigestData.fromDigestHex(Hex.encode(utxoFromSource)));
        } catch (Exception e) {
            continue;
        }
        if (sourceTx == null) {
            return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
        }
    }
}

```

```

    }
    fromOfFromCoin = sourceTx.getCoinData().getTo().get((int) new VarInt(fromIndex,
0).value);

    from.setFrom(fromOfFromCoin);
}

if(fromOfFromCoin == null) {
    Log.warn("from coin not found!");
    continue;
}

address = fromOfFromCoin.getAddress();
if(!AccountLegerUtils.isLocalAccount(address)) {
    continue;
}
try {
    fromList.add(new Entry<>(fromSource, fromOfFromCoin.serialize()));
} catch (IOException e) {
    throw new NulsRuntimeException(e);
}
}

Result result = localUtxoStorageService.batchSaveAndDeleteUTXO(fromList, toList);
if (result.isFailed() || result.getData() == null || (int) result.getData() != fromList.size() +
toList.size()) {
    return Result.getFailed();
}

return Result.getSuccess();
}

```

@Override

```

public Result<List<byte[]>> unlockCoinData(Transaction tx, long newLockTime) {
    List<byte[]> addresses = new ArrayList<>();
    CoinData coinData = tx.getCoinData();
    if (coinData != null) {
        List<Coin> tos = coinData.getTo();
        Coin to;
        for (int i = 0, length = tos.size(); i < length; i++) {
            to = tos.get(i);

```

```

        if (to.getLockTime() == -1L) {
            Coin needUnlockUtxoNew = new Coin(to.getOwner(), to.getNa(), newLockTime);
            needUnlockUtxoNew.setFrom(to.getFrom());
            try {
                byte[] outKey = ArraysTool.concatenate(tx.getHash().serialize(), new
VarInt(i).encode());
                saveUTXO(outKey, needUnlockUtxoNew.serialize());
                addresses.add(to.getAddress());
            } catch (IOException e) {
                throw new NulsRuntimeException(e);
            }
            //todo , think about weather to add a transaction history
            break;
        }
    }
}
return Result.getSuccess().setData(addresses);
}

```

@Override

```

public Result<List<byte[]>> rollbackUnlockTxCoinData(Transaction tx) {
    List<byte[]> addresses = new ArrayList<>();
    CoinData coinData = tx.getCoinData();
    if (coinData != null) {
        // lock utxo - to
        List<Coin> tos = coinData.getTo();
        for (int i = 0, length = tos.size(); i < length; i++) {
            Coin to = tos.get(i);
            if (to.getLockTime() == -1L) {
                try {
                    byte[] outKey = ArraysTool.concatenate(tx.getHash().serialize(), new
VarInt(i).encode());
                    saveUTXO(outKey, to.serialize());
                    addresses.add(to.getAddress());
                } catch (IOException e) {
                    throw new NulsRuntimeException(e);
                }
                break;
            }
        }
    }
}
return Result.getSuccess().setData(addresses);
}

```

```

    }

    protected void saveUTXO(byte[] outKey, byte[] serialize) {
        localUtxoStorageService.saveUTXO(outKey, serialize);
    }

    @Override
    public Result getUtxo(byte[] owner) {
        if (owner == null) {
            return null;
        }
        Coin coin = localUtxoStorageService.getUtxo(owner);
        if (coin == null) {
            return Result.getFailed();
        }
        return Result.getSuccess().setData(coin);
    }
}

12:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
base\src\main\java\io\nuls\account\ledger\base\service\impl\TransactionInfoServiceImpl.java
*/
package io.nuls.account.ledger.base.service.impl;

import io.nuls.account.ledger.base.service.TransactionInfoService;
import io.nuls.account.ledger.base.util.TxInfoComparator;
import io.nuls.account.ledger.constant.AccountLedgerErrorCode;
import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.account.ledger.storage.po.TransactionInfoPo;
import io.nuls.account.ledger.storage.service.TransactionInfoStorageService;
import io.nuls.account.ledger.storage.service.impl.TransactionInfoStorageServiceImpl;
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Result;

```

```

import io.nuls.kernel.utils.AddressTool;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * author Facjas
 * date 2018/5/27.
 */
@Component
public class TransactionInfoServiceImpl implements TransactionInfoService {
    @Autowired
    TransactionInfoStorageService transactionInfoStorageService;

    @Override
    public Result<List<TransactionInfo>> getTxInfoList(byte[] address) {
        try {
            List<TransactionInfoPo> infoPoList =
transactionInfoStorageService.getTransactionInfoListByAddress(address);
            List<TransactionInfo> infoList = new ArrayList<>();
            for (TransactionInfoPo po : infoPoList) {
                if (po.getTxType() == ConsensusConstant.TX_TYPE_RED_PUNISH || po.getTxType()
== ConsensusConstant.TX_TYPE_YELLOW_PUNISH) {
                    continue;
                }
                infoList.add(po.toTransactionInfo());
            }

            Collections.sort(infoList, TxInfoComparator.getInstance());
            return Result.getSuccess().setData(infoList);
        } catch (NulsException e) {
            Log.error(e);
            return Result.getFailed(e.getErrorCode());
        }
    }

    @Override
    public Result<Integer> saveTransactionInfo(TransactionInfoPo infoPo, List<byte[]> addresses) {
        if (infoPo == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
    }

```

```

    }

    if (addresses == null || addresses.size() == 0) {
        return Result.getSuccess().setData(new Integer(0));
    }

    List<byte[]> savedKeyList = new ArrayList<>();

    try {
        for (int i = 0; i < addresses.size(); i++) {
            byte[] infoKey = new byte[Address.ADDRESS_LENGTH + infoPo.getTxHash().size()];
            System.arraycopy(addresses.get(i), 0, infoKey, 0, Address.ADDRESS_LENGTH);
            System.arraycopy(infoPo.getTxHash().serialize(), 0, infoKey,
Address.ADDRESS_LENGTH, infoPo.getTxHash().size());
            transactionInfoStorageService.saveTransactionInfo(infoKey, infoPo);
            savedKeyList.add(infoKey);
        }
    } catch (IOException e) {
        for (int i = 0; i < savedKeyList.size(); i++) {
            transactionInfoStorageService.deleteTransactionInfo(savedKeyList.get(i));
        }
        return Result.getFailed(AccountLedgerErrorCode.IO_ERROR);
    }
    return Result.getSuccess().setData(new Integer(addresses.size()));
}

```

@Override

```

public Result deleteTransactionInfo(TransactionInfoPo infoPo) {
    byte[] infoBytes = null;
    if (infoPo == null) {
        return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
    }

    try {
        infoBytes = infoPo.serialize();
    } catch (IOException e) {
        return Result.getFailed(AccountLedgerErrorCode.IO_ERROR);
    }

    if (ArraysTool.isEmptyOrNull(infoBytes)) {
        return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
    }
}

```



```

byte[] addresses = infoPo.getAddresses();
if (addresses.length % Address.ADDRESS_LENGTH != 0) {
    return Result.getFailed(KernelErrorCode.PARAMETER_ERROR);
}

int addressCount = addresses.length / Address.ADDRESS_LENGTH;

for (int i = 0; i < addressCount; i++) {

    byte[] infoKey = new byte[Address.ADDRESS_LENGTH + infoPo.getTxHash().size()];
    System.arraycopy(addresses, i * Address.ADDRESS_LENGTH, infoKey, 0,
Address.ADDRESS_LENGTH);
    try {
        System.arraycopy(infoPo.getTxHash().serialize(), 0, infoKey,
Address.ADDRESS_LENGTH, infoPo.getTxHash().size());
    } catch (IOException e) {
        Log.info(e.getMessage());
    }
    transactionInfoStorageService.deleteTransactionInfo(infoKey);
}

return Result.getSuccess().setData(new Integer(addressCount));
}
}

```

13:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\main\java\io\nuls\account\ledger\base\service\LocalUtxoService.java  
 \*/

```

package io.nuls.account.ledger.base.service;

```

```

import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;

```

```

import java.util.List;

```

```

/**
 * author Facjas
 * date 2018/5/27.
 */

```

```

public interface LocalUtxoService {

```

```

    Result saveUtxoForLocalAccount(Transaction tx);

    Result saveUtxoForAccount(Transaction tx, byte[] addresses);

    Result saveUtxoForAccount(Transaction tx, List<byte[]> addressesList);

    Result deleteUtxoOfTransaction(Transaction tx);

    Result<List<byte[]>> unlockCoinData(Transaction tx, long newLockTime);

    Result<List<byte[]>> rollbackUnlockTxCoinData(Transaction tx);

    Result getUtxo(byte[] owner);
}

14:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
base\src\main\java\io\nuls\account\ledger\base\service\TransactionInfoService.java
*/

package io.nuls.account.ledger.base.service;

import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.account.ledger.storage.po.TransactionInfoPo;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 * author Facjas
 * date 2018/5/27.
 */

public interface TransactionInfoService {

    Result<List<TransactionInfo>> getTxInfoList(byte[] address);

    Result<Integer> saveTransactionInfo(TransactionInfoPo infoPo, List<byte[]> addresses);

    Result deleteTransactionInfo(TransactionInfoPo infoPo);
}

```

```
15:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-  
base\src\main\java\io\nuls\account\ledger\base\task\CheckUnConfirmTxThread.java  
package io.nuls.account.ledger.base.task;
```

```
import io.nuls.account.ledger.base.manager.BalanceManager;  
import io.nuls.account.ledger.base.service.TransactionInfoService;  
import io.nuls.account.ledger.base.service.impl.AccountLedgerServiceImpl;  
import io.nuls.account.ledger.base.util.AccountLegerUtils;  
import io.nuls.account.ledger.storage.po.TransactionInfoPo;  
import io.nuls.account.ledger.storage.service.LocalUtxoStorageService;  
import io.nuls.account.ledger.storage.service.UnconfirmedTransactionStorageService;  
import io.nuls.core.tools.crypto.Hex;  
import io.nuls.core.tools.log.Log;  
import io.nuls.db.model.Entry;  
import io.nuls.kernel.exception.NulsRuntimeException;  
import io.nuls.kernel.func.TimeService;  
import io.nuls.kernel.lite.annotation.Autowired;  
import io.nuls.kernel.lite.annotation.Component;  
import io.nuls.kernel.model.*;  
import io.nuls.kernel.utils.AddressTool;  
import io.nuls.kernel.utils.VarInt;  
import io.nuls.ledger.service.LedgerService;  
import io.nuls.protocol.service.TransactionService;  
import io.nuls.protocol.utils.TransactionTimeComparator;  
  
import java.io.IOException;  
import java.util.*;
```

```
@Component
```

```
public class CheckUnConfirmTxThread implements Runnable {
```

```
    @Autowired
```

```
    private AccountLedgerServiceImpl accountLedgerService;
```

```
    @Autowired
```

```
    private TransactionService transactionService;
```

```
    @Autowired
```

```
    private LedgerService ledgerService;
```

@Autowired

private UnconfirmedTransactionStorageService unconfirmedTransactionStorageService;

@Autowired

private LocalUtxoStorageService localUtxoStorageService;

@Autowired

private BalanceManager balanceManager;

@Autowired

private TransactionInfoService transactionInfoService;

private TransactionTimeComparator comparator = TransactionTimeComparator.getInstance();

@Override

public void run() {

try {

doTask();

} catch (Exception e) {

Log.error(e);

}

}

private void doTask() throws IOException {

List<Transaction> list = accountLedgerService.getAllUnconfirmedTransaction().getData();

if (list == null || list.size() == 0) {

return;

}

Map<String, Coin> toMaps = new HashMap<>();

Set<String> fromSet = new HashSet<>();

Collections.sort(list, this.comparator);

for (Transaction tx : list) {

Result result = verifyTransaction(tx, toMaps, fromSet);

boolean hashRight =

NulsDigestData.calcDigestData(tx.serializeForHash()).equals(tx.getHash());

if (result.isSuccess() && hashRight) {

if (TimeService.currentTimeMillis() - tx.getTime() < 300000L) {

return;

}

result = reBroadcastTransaction(tx);

if (result.isFailed()) {

```

        Log.info("reBroadcastTransaction tx error");
    }
} else {
    deleteUnconfirmedTransaction(tx);
    List<byte[]> addresses = tx.getAllRelativeAddress();
    Set<String> set = new HashSet<>();
    for (byte[] address : addresses) {
        if (AccountLegerUtils.isLocalAccount(address) &&
set.add(AddressTool.getStringAddressByBytes(address))) {
            balanceManager.refreshBalance(address);
        }
    }
}
}
}
}

```

```

private void deleteUnconfirmedTransaction(Transaction tx) {
    accountLedgerService.resetUsedTxSets();
    unconfirmedTransactionStorageService.deleteUnconfirmedTx(tx.getHash());
    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    transactionInfoService.deleteTransactionInfo(txInfoPo);
    rollbackUtxo(tx);
}

```

```

private void rollbackUtxo(Transaction tx) {
    if (tx == null) {
        return;
    }
}

```

```

CoinData coinData = tx.getCoinData();
if (coinData != null) {
    // save - from
    List<Coin> froms = coinData.getFrom();
    List<Entry<byte[], byte[]>> fromList = new ArrayList<>();
    byte[] fromSource;
    byte[] utxoFromSource;
    byte[] fromIndex;
    Transaction sourceTx;
    Coin fromCoin;
    for (Coin from : froms) {
        fromSource = from.getOwner();
    }
}

```

```

        utxoFromSource = new byte[tx.getHash().size()];
        fromIndex = new byte[fromSource.length - utxoFromSource.length];
        System.arraycopy(fromSource, 0, utxoFromSource, 0, tx.getHash().size());
        System.arraycopy(fromSource, tx.getHash().size(), fromIndex, 0, fromIndex.length);
        try {
            sourceTx =
ledgerService.getTx(NulsDigestData.fromDigestHex(Hex.encode(utxoFromSource)));
        } catch (Exception e) {
            continue;
        }
        if (sourceTx == null) {
            continue;
        }
        try {
            fromCoin = sourceTx.getCoinData().getTo().get((int) new VarInt(fromIndex, 0).value);

            //if (!AccountLegerUtils.isLocalAccount(fromCoin.getOwner()))
            if (!AccountLegerUtils.isLocalAccount(fromCoin.getAddress())) {
                continue;
            }
            Coin fromCoinFromLedger = ledgerService.getUtxo(fromSource);
            if (fromCoinFromLedger == null || !fromCoinFromLedger.usable()) {
                continue;
            }
            fromList.add(new Entry<>(from.getOwner(), fromCoin.serialize()));
        } catch (IOException e) {
            throw new NulsRuntimeException(e);
        }
    }

    // delete utxo - to
    List<Coin> tos = coinData.getTo();
    List<byte[]> toList = new ArrayList<>();
    Coin toCoin;
    byte[] outKey;
    for (int i = 0, length = tos.size(); i < length; i++) {
        try {
            toCoin = tos.get(i);
            /*if (!AccountLegerUtils.isLocalAccount(toCoin.getOwner())) {
                continue;
            }*/
            if (!AccountLegerUtils.isLocalAccount(toCoin.getAddress())) {

```

```

        continue;
    }
    outKey = org.spongycastle.util.Arrays.concatenate(tx.getHash().serialize(), new
VarInt(i).encode());
    toList.add(outKey);
} catch (IOException e) {
    Log.info("delete unconfirmed output error");
    throw new NulsRuntimeException(e);
}
}
localUtxoStorageService.batchSaveAndDeleteUTXO(fromList, toList);
}
}

```

```

private Result reBroadcastTransaction(Transaction tx) {
    Result sendResult = transactionService.broadcastTx(tx);
    if (sendResult.isFailed()) {
        return sendResult;
    }
    return Result.getSuccess();
}

```

```

private Result verifyTransaction(Transaction tx, Map<String, Coin> toMaps, Set<String>
fromSet) {
    Result result = tx.verify();
    if (result.isFailed()) {
        return result;
    }
    result = ledgerService.verifyCoinData(tx, toMaps, fromSet);

    if (result.isFailed()) {
        return result;
    }
    return Result.getSuccess();
}
}

```

```

16:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
base\src\main\java\io\nuls\account\ledger\base\util\AccountLegerUtils.java
*/
package io.nuls.account.ledger.base.util;

```

```

import io.nuls.account.model.Account;
import io.nuls.account.service.AccountService;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.VarInt;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;

/**
 * author Facjas
 * date 2018/5/27.
 */
@Component
public class AccountLegerUtils {

    @Autowired
    private static AccountService accountService;

    private final static int TX_HASH_LENGTH = NulsDigestData.HASH_LENGTH;

    public static boolean isLocalAccount(byte[] address) {
        Collection<Account> localAccountList = accountService.getAccountList().getData();
        if (localAccountList == null || localAccountList.size() == 0) {
            return false;
        }

        for (Account account : localAccountList) {
            if (Arrays.equals(account.getAddress().getAddressBytes(), address)) {
                return true;
            }
        }
        return false;
    }

    public static List<byte[]> getLocalAddresses() {

```



```

List<byte[]> result = new ArrayList<>();
Collection<Account> localAccountList = accountService.getAccountList().getData();
if (localAccountList == null || localAccountList.size() == 0) {
    return result;
}
List<byte[]> destAddresses = new ArrayList<>();
for (Account account : localAccountList) {
    destAddresses.add(account.getAddress().getAddressBytes());
}

return destAddresses;
}

```

```

public static List<byte[]> getRelatedAddresses(Transaction tx) {
    List<byte[]> result = new ArrayList<>();
    if (tx == null) {
        return result;
    }
    Collection<Account> localAccountList = accountService.getAccountList().getData();
    if (localAccountList == null || localAccountList.size() == 0) {
        return result;
    }
    List<byte[]> destAddresses = new ArrayList<>();
    for (Account account : localAccountList) {
        destAddresses.add(account.getAddress().getAddressBytes());
    }

    return getRelatedAddresses(tx, destAddresses);
}

```

```

public static List<byte[]> getRelatedAddresses(Transaction tx, List<byte[]> addresses) {
    List<byte[]> result = new ArrayList<>();
    if (tx == null) {
        return result;
    }
    if (addresses == null || addresses.size() == 0) {
        return result;
    }
    //
    List<byte[]> sourceAddresses = tx.getAllRelativeAddress();
    if (sourceAddresses == null || sourceAddresses.size() == 0) {
        return result;
    }
}

```

```

    }

    for (byte[] tempSourceAddress : sourceAddresses) {
        for (byte[] tempDestAddress : addresses) {
            if (Arrays.equals(tempDestAddress, tempSourceAddress)) {
                result.add(tempSourceAddress);
                continue;
            }
        }
    }
    return result;
}

public static boolean isLocalTransaction(Transaction tx) {

    if (tx == null) {
        return false;
    }
    Collection<Account> localAccountList = accountService.getAccountList().getData();
    if (localAccountList == null || localAccountList.size() == 0) {
        return false;
    }
    List<byte[]> addresses = tx.getAllRelativeAddress();
    for (int j = 0; j < addresses.size(); j++) {
        if (AccountLegerUtils.isLocalAccount(addresses.get(j))) {
            return true;
        }
    }
    return false;
}

public static boolean isTxRelatedToAddress(Transaction tx, byte[] address){
    List<byte[]> sourceAddresses = tx.getAllRelativeAddress();
    for (byte[] tmpAddress : sourceAddresses){
        if(Arrays.equals(tmpAddress, address)){
            return true;
        }
    }
    return false;
}

public static byte[] getTxHashBytes(byte[] fromBytes) {

```

```

        if(fromBytes == null || fromBytes.length < TX_HASH_LENGTH) {
            return null;
        }
        byte[] txBytes = new byte[TX_HASH_LENGTH];
        System.arraycopy(fromBytes, 0, txBytes, 0, TX_HASH_LENGTH);
        return txBytes;
    }

    public static String getTxHash(byte[] fromBytes) {
        byte[] txBytes = getTxHashBytes(fromBytes);
        if(txBytes != null) {
            return Hex.encode(txBytes);
        }
        return null;
    }

    public static byte[] getIndexBytes(byte[] fromBytes) {
        if(fromBytes == null || fromBytes.length < TX_HASH_LENGTH) {
            return null;
        }
        int length = fromBytes.length - TX_HASH_LENGTH;
        byte[] indexBytes = new byte[length];
        System.arraycopy(fromBytes, TX_HASH_LENGTH, indexBytes, 0, length);
        return indexBytes;
    }

    public static Integer getIndex(byte[] fromBytes) {
        byte[] indexBytes = getIndexBytes(fromBytes);
        if(indexBytes != null) {
            VarInt varInt = new VarInt(indexBytes, 0);
            return Math.toIntExact(varInt.value);
        }
        return null;
    }
}

```

17:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\main\java\io\nuls\account\ledger\base\util\CoinComparator.java  
 \*/

package io.nuls.account.ledger.base.util;

```

import io.nuls.kernel.model.Coin;

import java.util.Comparator;

public class CoinComparator implements Comparator<Coin> {

    private static CoinComparator instance = new CoinComparator();

    private CoinComparator() {

    }

    public static CoinComparator getInstance() {
        return instance;
    }

    @Override
    public int compare(Coin o1, Coin o2) {
        if(o1 == null) {
            return 1;
        }
        if(o2 == null) {
            return -1;
        }
        return o1.getNa().compareTo(o2.getNa());
    }
}

```

```

18:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
base\src\main\java\io\nuls\account\ledger\base\util\CoinComparatorDesc.java
*/

```

```

package io.nuls.account.ledger.base.util;

import io.nuls.kernel.model.Coin;

import java.util.Comparator;

public class CoinComparatorDesc implements Comparator<Coin> {
    private static CoinComparatorDesc instance = new CoinComparatorDesc();

    private CoinComparatorDesc() {

```

```

    }

    public static CoinComparatorDesc getInstance() {
        return instance;
    }

    @Override
    public int compare(Coin o1, Coin o2) {
        if(o1 == null) {
            return -1;
        }
        if(o2 == null) {
            return 1;
        }
        return o2.getNa().compareTo(o1.getNa());
    }
}

19:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
base\src\main\java\io\nuls\account\ledger\base\util\TxInfoComparator.java
*/

```

```

package io.nuls.account.ledger.base.util;

import io.nuls.account.ledger.model.TransactionInfo;

import java.util.Comparator;

public class TxInfoComparator implements Comparator<TransactionInfo> {

    private TxInfoComparator() {

    }

    private static TxInfoComparator instance = new TxInfoComparator();

    public static TxInfoComparator getInstance() {
        return instance;
    }

    @Override
    public int compare(TransactionInfo o1, TransactionInfo o2) {

```

```

        if (o1.getTime() < o2.getTime()) {
            return 1;
        } else if (o1.getTime() > o2.getTime()) {
            return -1;
        }
        return 0;
    }
}

```

20:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\test\java\BaseTest.java

\*/

```

import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;

```

```

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;

```

```

public class BaseTest {

```

```

    public static String post(String url, final String param, String encoding) {
        StringBuffer sb = new StringBuffer();
        OutputStream os = null;
        InputStream is = null;
        InputStreamReader isr = null;
        BufferedReader br = null;
        // UTF-8
        if (StringUtils.isNull(encoding)) {
            encoding = "UTF-8";
        }
        try {
            URL u = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) u.openConnection();
            connection.setRequestProperty("Content-Type", "application/json");
            connection.setDoOutput(true);
            connection.setDoInput(true);
            connection.setRequestMethod("POST");

            connection.connect();

```

```

os = connection.getOutputStream();
os.write(param.getBytes(encoding));
os.flush();
is = connection.getInputStream();
isr = new InputStreamReader(is, encoding);
br = new BufferedReader(isr);
String line;
while ((line = br.readLine()) != null) {
    sb.append(line);
    sb.append("\n");
}
} catch (Exception ex) {
    System.err.println(ex);
} finally {
    if (is != null) {
        try {
            is.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (os != null) {
        try {
            os.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (isr != null) {
        try {
            isr.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (br != null) {
        try {
            br.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
}

```

```

    }
    return sb.toString();
}
}

```

21:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-base\src\test\java\ScriptTransactionTestTool.java

```

import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.CoinData;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.script.Script;
import io.nuls.kernel.script.ScriptBuilder;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.protocol.model.tx.TransferTransaction;

```

```

import java.io.IOException;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;

```

```

/**
 * @author: Niels Wang
 * @date: 2018/10/5
 */

```

```

public class ScriptTransactionTestTool extends BaseTest {

```

```

    // @Test

```

```

    public void test() throws Exception {
        NulsContext.MAIN_NET_VERSION = 2;

```

```

        TransferTransaction tx = new TransferTransaction();
        tx.setRemark("test script".getBytes());

```

```

        CoinData data = new CoinData();
        Coin coin = new Coin();

```

```

        coin.setOwner(ArraysTool.concatenate(Hex.decode("0020dab71b3cd376e2ccf2f290e384d2917cc

```



```

0929f8de582f63a01fc15144fe38371"), new byte[]{0}));
    coin.setNa(Na.parseNuls(9997));
    coin.setLockTime(0);
    List<Coin> from = new ArrayList<>();
    from.add(coin);
    data.setFrom(from);
    Coin toCoin = new Coin();
    toCoin.setLockTime(0);
    Script script =
ScriptBuilder.createOutputScript(AddressTool.getAddress("NsdvuzHyQJEJkz4LEKweDeCs97845
xN9"),1);
    toCoin.setOwner(script.getProgram());
    toCoin.setNa(Na.parseNuls(9994));

    List<Coin> to = new ArrayList<>();
    to.add(toCoin);
    data.setTo(to);
    tx.setCoinData(data);

//    ECKey ecKey = ECKey.fromPrivate(new
BigInteger(1,Hex.decode("00b491621168dff80c4684f7445ef378ba4d381b2fe2a7b1fbf905864ed
8fbeb9")));
    ECKey ecKey = ECKey.fromPrivate(new
BigInteger(1,Hex.decode("4b19caef601a45531b7068430a5b0e380a004001f14bfec025ddf16d5d8
7fa8e")));
    List<ECKey> signEckey = new ArrayList<>();
    signEckey.add(ecKey);
    List<ECKey> scriptEckey = new ArrayList<>();
    SignatureUtil.createTransactionSignature(tx, scriptEckey, signEckey);

    String param = "{\"txHex\": \"" + Hex.encode(tx.serialize()) + "\"}";
    String res = post("http://127.0.0.1:7001/api/accountledger/transaction/validTransaction",
param, "utf-8");
    System.out.println(res);
    res = post("http://127.0.0.1:7001/api/accountledger/transaction/broadcast", param, "utf-8");
    System.out.println(res);
}
//@Test
public void test1() throws IOException {

    NulsContext.MAIN_NET_VERSION = 2;

```

```

TransferTransaction tx = new TransferTransaction();
tx.setRemark("test script".getBytes());

CoinData data = new CoinData();
Coin coin = new Coin();
coin.setOwner(ArraysTool.concatenate(Hex.decode("0020dab71b3cd376e2ccf2f290e384d2917cc
0929f8de582f63a01fc15144fe38371"), new byte[]{0}));
coin.setNa(Na.parseNuls(9997));
coin.setLockTime(0);
List<Coin> from = new ArrayList<>();
from.add(coin);
data.setFrom(from);
Coin toCoin = new Coin();
toCoin.setLockTime(0);
Script script =
ScriptBuilder.createOutputScript(AddressTool.getAddress("NsdvuzHyQJEJkz4LEKweDeCs97845
xN9"),1);
toCoin.setOwner(script.getProgram());
toCoin.setNa(Na.parseNuls(9994));

List<Coin> to = new ArrayList<>();
to.add(toCoin);
data.setTo(to);
tx.setCoinData(data);

//      ECKey ecKey = ECKey.fromPrivate(new
BigInteger(1,Hex.decode("00b491621168dff80c4684f7445ef378ba4d381b2fe2a7b1fbf905864ed
8fbeb9")));
      ECKey ecKey = ECKey.fromPrivate(new
BigInteger(1,Hex.decode("4b19caef601a45531b7068430a5b0e380a004001f14bfec025ddf16d5d8
7fa8e")));
      List<ECKey> signEckey = new ArrayList<>();
      signEckey.add(ecKey);
      List<ECKey> scriptEckey = new ArrayList<>();
      SignatureUtil.createTransactionSignature(tx, scriptEckey, signEckey);

String param = "{\"txHex\": \"" + Hex.encode(tx.serialize()) + "\"}";
String res = post("http://127.0.0.1:7001/api/accountledger/transaction/validTransaction",
param, "utf-8");
System.out.println(res);
res = post("http://127.0.0.1:7001/api/accountledger/transaction/broadcast", param, "utf-8");

```

```
        System.out.println(res);
    }
}
```

22:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\AccountLedgerResource.java  
\*/

```
/**
 * @author: Facjas
 */
```

```
package io.nuls.accout.ledger.rpc;
```

```
import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.base.service.LocalUtxoService;
import io.nuls.account.ledger.base.util.AccountLegerUtils;
import io.nuls.account.ledger.constant.AccountLedgerErrorCode;
import io.nuls.account.ledger.model.MultipleAddressTransferModel;
import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.Balance;
import io.nuls.account.service.AccountService;
import io.nuls.account.util.AccountTool;
import io.nuls.accout.ledger.rpc.dto.*;
import io.nuls.accout.ledger.rpc.form.*;
import io.nuls.accout.ledger.rpc.util.UtxoDtoComparator;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.service.ContractService;
import io.nuls.core.tools.crypto.AESEncrypt;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Exception.CryptoException;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.map.MapUtil;
import io.nuls.core.tools.page.Page;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.constant.TxStatusEnum;
import io.nuls.kernel.context.NulsContext;
```

```
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.*;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.constant.LedgerErrorCode;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.tx.TransferTransaction;
import io.nuls.protocol.model.validator.TxMaxSizeValidator;
import io.swagger.annotations.*;
import org.spongycastle.util.Arrays;
```

```
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.io.UnsupportedEncodingException;
import java.math.BigInteger;
import java.util.*;
import java.util.stream.Collectors;
```

```
/**
 * author Facjas
 * date 2018/5/14.
 */
```

```
@Path("/accountledger")
@Api(value = "/accountledger", description = "accountledger")
@Component
public class AccountLedgerResource {
```

```
    @Autowired
    private AccountService accountService;
```

```
    @Autowired
    private AccountLedgerService accountLedgerService;
```

```
    @Autowired
    private LedgerService ledgerService;
```

```
@Autowired
private LocalUtxoService localUtxoService;
```

```
@Autowired
private ContractService contractService;
```

```
@GET
```

```
@Path("/balance/{address}")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
@ApiOperation(value = "", notes = "result.data: balanceJson ")
```

```
@ApiResponses(value = {
```

```
    @ApiResponse(code = 200, message = "success", response = Balance.class)
```

```
})
```

```
public RpcClientResult getBalance(@ApiParam(name = "address", value = "", required = true)
```

```
    @PathParam("address") String address) {
```

```
    byte[] addressBytes = null;
```

```
    try {
```

```
        addressBytes = AddressTool.getAddress(address);
```

```
    } catch (Exception e) {
```

```
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
```

```
    }
```

```
    if (addressBytes.length != Address.ADDRESS_LENGTH) {
```

```
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
```

```
    }
```

```
    Result result = accountLedgerService.getBalance(addressBytes);
```

```
    return result.toRpcClientResult();
```

```
}
```

```
@POST
```

```
@Path("/transfer")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
@ApiOperation(value = "", notes = "result.data: resultJson ")
```

```
@ApiResponses(value = {
```

```
    @ApiResponse(code = 200, message = "success")
```

```
})
```

```
public RpcClientResult transfer(@ApiParam(name = "form", value = "", required = true)
```

```
TransferForm form) {
```

```
    if (form == null) {
```

```
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
```

```
    }
```

```
    if (!AddressTool.validAddress(form.getAddress())) {
```

```
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
```

```

    }
    if (!AddressTool.validAddress(form.getToAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (form.getAmount() <= 0) {
        return
Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }

    byte[] remarkBytes = new byte[0];

    if (form.getRemark() != null && form.getRemark().length() > 0) {
        if (!validTxRemark(form.getRemark())) {
            return
Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

        try {
            remarkBytes = form.getRemark().getBytes(NulsConfig.DEFAULT_ENCODING);
        } catch (UnsupportedEncodingException e) {
            return
Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
    }

    Na value = Na.valueOf(form.getAmount());

    Result result = accountLedgerService.transfer(AddressTool.getAddress(form.getAddress()),
        AddressTool.getAddress(form.getToAddress()),
        value, form.getPassword(), remarkBytes,
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("value", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

@POST

@Path("/changeWhole")

@Produces(MediaType.APPLICATION\_JSON)

```

@ApiOperation(value = "", notes = "result.data: resultJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success")
})
public RpcClientResult changeWhole(@ApiParam(name = "form", value = "", required = true)
ChangeToWholeTransactionForm form) {
    if (form == null) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Result result =
accountLedgerService.changeWhole(AddressTool.getAddress(form.getAddress()),
form.getPassword(), TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("value", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

```

@POST
@Path("/dapp")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "DAPP", notes = "result.data: resultJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success")
})
public RpcClientResult dapp(@ApiParam(name = "form", value = "DAPP", required = true)
DataTransactionForm form) {
    if (form == null) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(form.getData())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
}

```

```

byte[] data;
try {
    data = form.getData().getBytes("UTF-8");
} catch (UnsupportedEncodingException e) {
    return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
}
if (!validTxRemark(form.getRemark())) {
    return
Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
}
byte[] remarkBytes;
try {
    remarkBytes = form.getRemark().getBytes(NulsConfig.DEFAULT_ENCODING);
} catch (UnsupportedEncodingException e) {
    return
Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
}
Result result = accountLedgerService.dapp(AddressTool.getAddress(form.getAddress()),
form.getPassword(), data, remarkBytes);
return result.toRpcClientResult();
}

```

@POST

@Path("/multipleAddressTransfer")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "from", notes = "result.data: resultJson ")

@ApiResponses(value = {

    @ApiResponse(code = 200, message = "success")

})

```

public RpcClientResult multipleAddressTransfer(@ApiParam(name = "form", value = "",
required = true)

```

```

        MultipleTransactionForm form) {

```

```

    if (NulsContext.MAIN_NET_VERSION <= 1) {

```

```

        return Result.getFailed(KernelErrorCode.VERSION_TOO_LOW).toRpcClientResult();
    }

```

```

    List<MultipleAddressTransferModel> fromModelList = new ArrayList<>();

```

```

    List<MultipleAddressTransferModel> toModelList = new ArrayList<>();

```

```

    if (form.getInputs() == null || form.getOutputs() == null) {

```

```

        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }

```

```

    for (MulipleTxFromDto from : form.getInputs()) {

```



```

        MultipleAddressTransferModel model = new MultipleAddressTransferModel();
        if (!AddressTool.validAddress(from.getAddress())) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        model.setAddress(AddressTool.getAddress(from.getAddress()));
        fromModelList.add(model);
    }
    if (!validTxRemark(form.getRemark())) {
        return
        Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Long toTotal = 0L;
    for (MultipleTxToDto to : form.getOutputs()) {
        MultipleAddressTransferModel model = new MultipleAddressTransferModel();
        if (!AddressTool.validAddress(to.getToAddress())) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        model.setAddress(AddressTool.getAddress(to.getToAddress()));
        model.setAmount(to.getAmount());
        toModelList.add(model);
        toTotal += to.getAmount();
    }
    if (toTotal < 0) {
        return
        Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Result result = accountLedgerService.multipleAddressTransfer(fromModelList, toModelList,
    form.getPassword(), Na.valueOf(toTotal), form.getRemark(),
    TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("value", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

@GET

@Path("/estimateFee/{address}")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "—", notes = "result.data: resultJson ")

@ApiResponse(value = {

```

        @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult estimateFee(@ApiParam(name = "address", value = "—", required =
true)
        @PathParam("address") String address) {
        if (address == null) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        Result result = accountLedgerService.estimateFee(AddressTool.getAddress(address),
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
        Long fee = null;
        if (result.isSuccess()) {
            fee = ((Na) result.getData()).getValue();
        }
        Map<String, Long> map = new HashMap<>();
        map.put("fee", fee);
        result.setData(map);
        return result.toRpcClientResult();
    }

```

```

@GET
@Path("/getTotalUTXO/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "utxo—", notes = "result.data: resultJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success")
})
public RpcClientResult getTotalUTXO(@ApiParam(name = "address", value = "utxo—",
required = true)
    @PathParam("address") String address) {
    if (address == null) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Result result =
accountLedgerService.getAvailableTotalUTXO(AddressTool.getAddress(address));
    if (!result.isSuccess()) {

```

```

        return Result.getFailed(AccountErrorCode.FAILED).toRpcClientResult();
    }
    return result.toRpcClientResult();
}

@GET
@Path("/transfer/fee")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: resultJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success")
})
public RpcClientResult transferFee(@BeanParam() TransferFeeForm form) {
    if (form == null) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(form.getToAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (form.getAmount() <= 0) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!validTxRemark(form.getRemark())) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }

    Na value = Na.valueOf(form.getAmount());
    Result result =
accountLedgerService.transferFee(AddressTool.getAddress(form.getAddress()),
    AddressTool.getAddress(form.getToAddress()), value, form.getRemark(),
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
    Long fee = null;
    Long maxAmount = null;
    Map<String, Long> map = new HashMap<>();
    if (result.isSuccess()) {
        fee = ((Na) result.getData()).getValue();
        //
        long feeMax =
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES.multiply(TxMaxSizeValidator.MAX_T

```

```

X_BYTES).getValue();
    if(fee > feeMax){
        Transaction tx = new TransferTransaction();
        try {
            tx.setRemark(form.getRemark().getBytes(NulsConfig.DEFAULT_ENCODING));
        } catch (UnsupportedEncodingException e) {
            Log.error(e);
        }
        tx.setTime(TimeService.currentTimeMillis());
        CoinData coinData = new CoinData();
        Coin toCoin = new Coin(AddressTool.getAddress(form.getToAddress()), value);
        coinData.getTo().add(toCoin);
        tx.setCoinData(coinData);
        Result rs =
accountLedgerService.getMaxAmountOfOnce(AddressTool.getAddress(form.getAddress()), tx,
        TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
        if (rs.isSuccess()) {
            maxAmount = ((Na) rs.getData()).getValue();
        }
    }
    map.put("fee", fee);
    map.put("maxAmount", maxAmount);
    result.setData(map);
}
return result.toRpcClientResult();
}

```

```

@POST
@Path("/transaction")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: resultJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success")
})
public RpcClientResult createTransaction(@ApiParam(name = "form", value = "", required =
true)
        TransactionForm form) {
    if (form.getInputs() == null || form.getInputs().isEmpty()) {
        return RpcClientResult.getFailed("inputs error");
    }
    if (form.getOutputs() == null || form.getOutputs().isEmpty()) {
        return RpcClientResult.getFailed("outputs error");
    }
}

```

```
}
```

```
byte[] remark = null;
```

```
if (!StringUtils.isBlank(form.getRemark())) {
```

```
    try {
```

```
        remark = form.getRemark().getBytes(NulsConfig.DEFAULT_ENCODING);
```

```
    } catch (UnsupportedEncodingException e) {
```

```
        return RpcClientResult.getFailed("remark error");
```

```
    }
```

```
}
```

```
List<Coin> outputs = new ArrayList<>();
```

```
for (int i = 0; i < form.getOutputs().size(); i++) {
```

```
    OutputDto outputDto = form.getOutputs().get(i);
```

```
    Coin to = new Coin();
```

```
    try {
```

```
        to.setOwner(AddressTool.getAddress(outputDto.getAddress()));
```

```
    } catch (Exception e) {
```

```
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
```

```
    }
```

```
    try {
```

```
        to.setNa(Na.valueOf(outputDto.getValue()));
```

```
    } catch (Exception e) {
```

```
        return Result.getFailed(LedgerErrorCode.DATA_PARSE_ERROR).toRpcClientResult();
```

```
    }
```

```
if (outputDto.getLockTime() < 0) {
```

```
    return RpcClientResult.getFailed("lockTime error");
```

```
}
```

```
to.setLockTime(outputDto.getLockTime());
```

```
outputs.add(to);
```

```
}
```

```
List<Coin> inputs = new ArrayList<>();
```

```
for (int i = 0; i < form.getInputs().size(); i++) {
```

```
    InputDto inputDto = form.getInputs().get(i);
```

```
    byte[] key = Arrays.concatenate(Hex.decode(inputDto.getFromHash()), new  
VarInt(inputDto.getFromIndex()).encode());
```

```
    Coin coin = new Coin();
```

```
    coin.setOwner(key);
```

```
    coin.setLockTime(inputDto.getLockTime());
```

```

        coin.setNa(Na.valueOf(inputDto.getValue()));
        inputs.add(coin);
    }
    Result result = accountLedgerService.createTransaction(inputs, outputs, remark);
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("value", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

@POST

@Path("/transaction/sign")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "", notes = "result.data: resultJson ")

@ApiResponses(value = {

@ApiResponse(code = 200, message = "success")

```

}))
public RpcClientResult signTransaction(@ApiParam(name = "form", value = "", required = true)
                                     TransactionHexForm form) {
    if (StringUtils.isBlank(form.getPriKey())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(form.getTxHex())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }

    String priKey = form.getPriKey();
    if (StringUtils.isNotBlank(form.getPassword())) {
        if (StringUtils.validPassword(form.getPassword())) {
            //decrypt
            byte[] privateKeyBytes = null;
            try {
                privateKeyBytes = AESEncrypt.decrypt(Hex.decode(priKey), form.getPassword());
            } catch (CryptoException e) {
                return
            }
            Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
    }
}

```

```

        priKey = Hex.encode(privateKeyBytes);
    } else {
        return
Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
}

if (!ECKey.isValidPrivteHex(priKey)) {
    return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
}
//is private key matches address
ECKey key = ECKey.fromPrivate(new BigInteger(1, Hex.decode(priKey)));
try {
    String newAddress = AccountTool.newAddress(key).getBase58();
    if (!newAddress.equals(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
} catch (NulsException e) {
    return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
}

try {
    byte[] data = Hex.decode(form.getTxHex());
    Transaction tx = TransactionManager.getInstance(new NulsByteBuffer(data));
    tx = accountLedgerService.signTransaction(tx, key);

//    Result validateResult = tx.verify();
//    if (validateResult.isFailed()) {
//        return Result.getFailed(validateResult.getErrorCode()).toRpcClientResult();
//    }

//    for (Coin coin : tx.getCoinData().getFrom()) {
//        Coin utxo = ledgerService.getUtxo(coin.());
//        if (utxo == null) {
//            return
Result.getFailed(LedgerErrorCode.UTXO_NOT_FOUND).toRpcClientResult();
//        }
//
//        if (!form.getAddress().equals(AddressTool.getStringAddressByBytes(utxo.()))) {
//            return Result.getFailed(LedgerErrorCode.INVALID_INPUT).toRpcClientResult();
//        }
//
//

```

```

//      }

    Map<String, String> map = new HashMap<>();
    map.put("value", Hex.encode(tx.serialize()));
    return Result.getSuccess().setData(map).toRpcClientResult();
} catch (Exception e) {
    Log.error(e);
    return Result.getFailed(LedgerErrorCode.DATA_PARSE_ERROR).toRpcClientResult();
}
}

@POST
@Path("/transaction/broadcast")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: resultJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success")
})
public RpcClientResult broadcast(@ApiParam(name = "form", value = "", required = true)
BroadHexTxForm form) {
    if (StringUtils.isBlank(form.getTxHex())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    try {
        byte[] data = Hex.decode(form.getTxHex());
        Transaction tx = TransactionManager.getInstance(new NulsByteBuffer(data));
        Result result = accountLedgerService.broadcast(tx);
        if (result.isSuccess()) {
            Map<String, Object> map = new HashMap<>();
            map.put("value", tx.getHash().getDigestHex());
            result.setData(map);
        }
        return result.toRpcClientResult();
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed(LedgerErrorCode.DATA_PARSE_ERROR).toRpcClientResult();
    }
}

@POST
@Path("/transaction/valiTransaction")

```



```

@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: resultJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success")
})
public RpcClientResult valiTransaction(@ApiParam(name = "form", value = "", required = true)
BroadHexTxForm form) {
    if (StringUtils.isBlank(form.getTxHex())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    try {
        byte[] data = Hex.decode(form.getTxHex());
        Transaction tx = TransactionManager.getInstance(new NulsByteBuffer(data));
        ValidateResult validateResult = tx.verify();
        if (validateResult.isFailed()) {
            return Result.getFailed(validateResult.getErrorCode()).toRpcClientResult();
        }
        validateResult = this.ledgerService.verifyCoinData(tx, new HashMap<>(), new
HashSet<>());
        if (validateResult.isFailed() &&
!validateResult.getErrorCode().equals(TransactionErrorCode.ORPHAN_TX)) {
            return Result.getFailed(validateResult.getErrorCode()).toRpcClientResult();
        }
        Result result = Result.getSuccess();
        return result.toRpcClientResult();
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed(LedgerErrorCode.DATA_PARSE_ERROR).toRpcClientResult();
    }
}

@GET
@Path("/tx/list/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: balanceJson ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = Page.class)
})
public RpcClientResult getTxInfoList(@ApiParam(name = "address", value = "", required = true)
@PathParam("address") String address,
@ApiParam(name = "assetType", value = "")

```

```

        @QueryParam("assetType") String assetType,
        @ApiParam(name = "type", value = "")
        @QueryParam("type") Integer type,
        @ApiParam(name = "pageNumber", value = "")
        @QueryParam("pageNumber") Integer pageNumber,
        @ApiParam(name = "pageSize", value = "")
        @QueryParam("pageSize") Integer pageSize) {
    if (null == pageNumber || pageNumber == 0) {
        pageNumber = 1;
    }
    if (null == pageSize || pageSize == 0) {
        pageSize = 10;
    }
    if (pageNumber < 0 || pageSize < 0 || pageSize > 100) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (type == null || type <= 0) {
        type = -1;
    }

    byte[] addressBytes = null;
    Result dtoResult = Result.getSuccess();

    try {
        addressBytes = AddressTool.getAddress(address.trim());
    } catch (Exception e) {
        return
Result.getFailed(AccountLedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }

    List<TransactionInfo> result = new ArrayList<TransactionInfo>();

    boolean isEmptyAssetType = StringUtils.isBlank(assetType);
    boolean isNeedQueryToken = StringUtils.isBlank(assetType) ||
AddressTool.validAddress(assetType);
    boolean isNeedQueryNuls = StringUtils.isBlank(assetType) || "NULS".equals(assetType);

    Set<String> hashCheckSet = MapUtil.createHashSet(8);
    // token
    if (isNeedQueryToken) {
        Result<List<ContractTokenTransferInfoPo>> listResult =
contractService.getTokenTransferInfoList(address);

```

```

List<ContractTokenTransferInfoPo> list = listResult.getData();
if (list != null && list.size() > 0) {
    List<ContractTokenTransferInfoPo> tokenInfoList = null;
    if (!isEmptyAssetType) {
        String contractAddress = assetType;
        if (AddressTool.validAddress(contractAddress)) {
            tokenInfoList = list.stream().filter(po ->
contractAddress.equals(po.getContractAddress())).collect(Collectors.toList());
        }
    } else {
        tokenInfoList = list;
    }
    TransactionInfo info = null;
    for (ContractTokenTransferInfoPo po : tokenInfoList) {
        info = new TransactionInfo();
        // type -
        info.setTxType(1000);
        NulsDigestData hashData = new NulsDigestData();
        try {
            hashData.parse(po.getTxHash(), 0);
        } catch (NulsException e) {
            Log.error(e);
            //skip it
        }
        info.setTxHash(hashData);
        info.setContractAddress(AddressTool.getAddress(po.getContractAddress()));
        info.setTime(po.getTime());
        info.setBlockHeight(po.getBlockHeight());
        info.setStatus(po.getStatus());
        info.setInfo(po.getInfo(addressBytes));
        info.setSymbol(po.getSymbol());
        result.add(info);
        hashCheckSet.add(hashData.getDigestHex());
    }
}

//
if (isNeedQueryNuls) {
    Result<List<TransactionInfo>> rawResult =
accountLedgerService.getTxInfoList(addressBytes);

```

```

if (rawResult.isFailed()) {
    dtoResult.setSuccess(false);
    dtoResult.setErrorCode(rawResult.getErrorCode());
    return dtoResult.toRpcClientResult();
}

```

```

List<TransactionInfo> infoList = rawResult.getData();
if (infoList != null && infoList.size() > 0) {
    //
    List<TransactionInfo> baseList = infoList.stream().filter(info ->
hashCheckSet.add(info.getTxHash().getDigestHex())).collect(Collectors.toList());

```

```

        if (type == -1) {
            result.addAll(baseList);
        } else {
            for (TransactionInfo txInfo : baseList) {
                if (txInfo.getTxType() == type) {
                    result.add(txInfo);
                }
            }
        }
    }
}
}

```

```

result.sort(new Comparator<TransactionInfo>() {
    @Override
    public int compare(TransactionInfo o1, TransactionInfo o2) {
        return o1.compareTo(o2.getTime());
    }
});

```

```

Page<TransactionInfoDto> page = new Page<>(pageNumber, pageSize, result.size());
int start = pageNumber * pageSize - pageSize;
if (start >= page.getTotal()) {
    dtoResult.setData(page);
    return dtoResult.toRpcClientResult();
}

```

```

int end = start + pageSize;
if (end > page.getTotal()) {
    end = (int) page.getTotal();
}

```

```

List<TransactionInfoDto> infoDtoList = new ArrayList<>();
for (int i = start; i < end; i++) {
    TransactionInfo info = result.get(i);
    Transaction tx = ledgerService.getTx(info.getTxHash());
    if (tx == null) {
        tx = accountLedgerService.getUnconfirmedTransaction(info.getTxHash()).getData();
    }
    if (tx == null) {
        continue;
    }
    info.setBlockHeight(tx.getBlockHeight());
    // Token
    if (info.getTxType() != 1000) {
        info.setInfo(tx.getInfo(addressBytes));
    }
    infoDtoList.add(new TransactionInfoDto(info));
}
page.setList(infoDtoList);

dtoResult.setSuccess(true);
dtoResult.setData(page);
return dtoResult.toRpcClientResult();
}

```

```

@GET
@Path("/utxo/lock/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: balanceJson UTXO")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = Page.class)
})
public RpcClientResult getLockUtxo(@ApiParam(name = "address", value = "")
    @PathParam("address") String address,
    @ApiParam(name = "pageNumber", value = "")
    @QueryParam("pageNumber") Integer pageNumber,
    @ApiParam(name = "pageSize", value = "")
    @QueryParam("pageSize") Integer pageSize) {
    if (null == pageNumber || pageNumber == 0) {
        pageNumber = 1;
    }
}

```

```

if (null == pageSize || pageSize == 0) {
    pageSize = 10;
}
if (pageNumber < 0 || pageSize < 0 || pageSize > 100) {
    return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
}

byte[] addressBytes = null;
Result dtoResult = new Result<>();

try {
    addressBytes = AddressTool.getAddress(address);
} catch (Exception e) {
    return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
}
//utxo
Result<List<Coin>> result = accountLedgerService.getLockedUtxo(addressBytes);
if (result.isFailed()) {
    dtoResult.setSuccess(false);
    dtoResult.setErrorCode(result.getErrorCode());
    return dtoResult.toRpcClientResult();
}

List<Coin> coinList = result.getData();
Page<UtxoDto> page = new Page<>(pageNumber, pageSize, result.getData().size());
int start = pageNumber * pageSize - pageSize;
if (start >= coinList.size()) {
    dtoResult.setSuccess(true);
    dtoResult.setData(page);
    return dtoResult.toRpcClientResult();
}

List<UtxoDto> utxoDtoList = new ArrayList<>();
byte[] txHash = new byte[NulsDigestData.HASH_LENGTH];
for (Coin coin : coinList) {
    //uxto
    System.arraycopy(coin.getOwner(), 0, txHash, 0, NulsDigestData.HASH_LENGTH);
    Transaction tx = ledgerService.getTx(txHash);
    if (tx == null) {
        NulsDigestData hash = new NulsDigestData();
        try {
            hash.parse(txHash, 0);

```

```

        tx = accountLedgerService.getUnconfirmedTransaction(hash).getData();
    } catch (NulsException e) {
        Log.error(e);
        return
    }
    Result.getFailed(KernelErrorCode.DATA_PARSE_ERROR).toRpcClientResult();
}

//
if (tx == null) {
    continue;
}
utxoDtoList.add(new UtxoDto(coin, tx));
}

//page
page = new Page<>(pageNumber, pageSize, utxoDtoList.size());
if (start >= page.getTotal()) {
    dtoResult.setData(page);
    return dtoResult.toRpcClientResult();
}

Collections.sort(utxoDtoList, UtxoDtoComparator.getInstance());
int end = start + pageSize;
if (end > utxoDtoList.size()) {
    end = utxoDtoList.size();
}

page.setList(utxoDtoList.subList(start, end));
dtoResult.setSuccess(true);
dtoResult.setData(page);
return dtoResult.toRpcClientResult();
}

private boolean validTxRemark(String remark) {
    if (StringUtils.isBlank(remark)) {
        return true;
    }
    try {
        byte[] bytes = remark.getBytes(NulsConfig.DEFAULT_ENCODING);
        if (bytes.length > 100) {
            return false;
        }
    }
}

```

```

        return true;
    } catch (UnsupportedEncodingException e) {
        return false;
    }
}

@GET
@Path("/tx/{hash}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "hash")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = TransactionDto.class)
})
public RpcClientResult getTxByHash(@ApiParam(name = "hash", value = "hash", required =
true)
                                @PathParam("hash") String hash) {
    if (StringUtils.isBlank(hash)) {
        return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
    }
    if (!NulsDigestData.validHash(hash)) {
        return Result.getFailed(LedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Result result = getUnconfirmedTx(hash);
    if (result.isSuccess()) {
        return result.toRpcClientResult();
    }
    return getConfirmedTx(hash).toRpcClientResult();
}

/**
 *
 */
private Result getUnconfirmedTx(String hash) {
    Result result = null;
    try {
        Result<Transaction> txResult =
accountLedgerService.getUnconfirmedTransaction(NulsDigestData.fromDigestHex(hash));
        if (txResult.isFailed() || null == txResult.getData()) {
            result = Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
        } else {
            Transaction tx = txResult.getData();
            tx.setStatus(TxStatusEnum.UNCONFIRM);

```



```

TransactionDto txDto = null;
CoinData coinData = tx.getCoinData();
if (coinData != null) {
    // from
    List<Coin> froms = coinData.getFrom();
    if (froms != null && froms.size() > 0) {
        byte[] fromHash, owner;
        int fromIndex;
        NulsDigestData fromHashObj;
        Transaction fromTx;
        Coin fromUtxo;
        for (Coin from : froms) {
            owner = from.getOwner();
            // ownertxHashindex
            fromHash = AccountLegerUtils.getTxHashBytes(owner);
            fromIndex = AccountLegerUtils.getIndex(owner);
            // from UTXO
            fromHashObj = new NulsDigestData();
            fromHashObj.parse(fromHash, 0);
            //to,,
            fromTx =
accountLederService.getUnconfirmedTransaction(fromHashObj).getData();
            if (null == fromTx) {
                fromTx = ledgerService.getTx(fromHashObj);
            }
            fromUtxo = fromTx.getCoinData().getTo().get(fromIndex);
            from.setFrom(fromUtxo);
        }
    }
    txDto = new TransactionDto(tx);
    List<OutputDto> outputDtoList = new ArrayList<>();
    // to
    List<Coin> tos = coinData.getTo();
    if (tos != null && tos.size() > 0) {
        String txHash = hash;
        OutputDto outputDto = null;
        Coin to;
        for (int i = 0, length = tos.size(); i < length; i++) {
            to = tos.get(i);
            outputDto = new OutputDto(to);
            outputDto.setTxHash(txHash);
            outputDto.setIndex(i);
        }
    }
}

```

```

        outputDto.setStatus(0);
        outputDtoList.add(outputDto);
    }
}
txDto.setOutputs(outputDtoList);
//
    calTransactionValue(txDto);
}
result = Result.getSuccess();
result.setData(txDto);
}
} catch (NulsRuntimeException re) {
    Log.error(re);
    result = Result.getFailed(re.getErrorCode());
} catch (Exception e) {
    Log.error(e);
    result = Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION);
}
return result;
}

/**
 *
 */
private Result getConfirmedTx(String hash) {
    Result result = null;
    try {
        Transaction tx = ledgerService.getTx(NulsDigestData.fromDigestHex(hash));
        if (tx == null) {
            result = Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
        } else {
            tx.setStatus(TxStatusEnum.CONFIRMED);
            TransactionDto txDto = null;
            CoinData coinData = tx.getCoinData();
            if (coinData != null) {
                // from
                List<Coin> froms = coinData.getFrom();
                if (froms != null && froms.size() > 0) {
                    byte[] fromHash, owner;
                    int fromIndex;
                    NulsDigestData fromHashObj;
                    Transaction fromTx;

```

```

Coin fromUtxo;
for (Coin from : froms) {
    owner = from.getOwner();
    // ownertxHashindex
    fromHash = AccountLegerUtils.getTxHashBytes(owner);
    fromIndex = AccountLegerUtils.getIndex(owner);
    // from UTXO
    fromHashObj = new NulsDigestData();
    fromHashObj.parse(fromHash, 0);
    fromTx = ledgerService.getTx(fromHashObj);
    fromUtxo = fromTx.getCoinData().getTo().get(fromIndex);
    from.setFrom(fromUtxo);
}
}
txDto = new TransactionDto(tx);
List<OutputDto> outputDtoList = new ArrayList<>();
// to
List<Coin> tos = coinData.getTo();
if (tos != null && tos.size() > 0) {
    byte[] txHashBytes = tx.getHash().serialize();
    String txHash = hash;
    OutputDto outputDto = null;
    Coin to, temp;
    long bestHeight = NulsContext.getInstance().getBestHeight();
    long currentTime = TimeService.currentTimeMillis();
    long lockTime;
    for (int i = 0, length = tos.size(); i < length; i++) {
        to = tos.get(i);
        outputDto = new OutputDto(to);
        outputDto.setTxHash(txHash);
        outputDto.setIndex(i);
        temp = ledgerService.getUtxo(Arrays.concatenate(txHashBytes, new
VarInt(i).encode()));
        if (temp == null) {
            //
            outputDto.setStatus(3);
        } else {
            lockTime = temp.getLockTime();
            if (lockTime < 0) {
                //
                outputDto.setStatus(2);
            } else if (lockTime == 0) {

```

```

        //
        outputDto.setStatus(0);
    } else if (lockTime > NulsConstant.BLOCKHEIGHT_TIME_DIVIDE) {
        //
        if (lockTime > currentTime) {
            //
            outputDto.setStatus(1);
        } else {
            //
            outputDto.setStatus(0);
        }
    } else {
        //
        if (lockTime > bestHeight) {
            //
            outputDto.setStatus(1);
        } else {
            //
            outputDto.setStatus(0);
        }
    }
}
outputDtoList.add(outputDto);
}
}
txDto.setOutputs(outputDtoList);
//
calTransactionValue(txDto);
}
result = Result.getSuccess();
result.setData(txDto);
}
} catch (NulsRuntimeException re) {
    Log.error(re);
    result = Result.getFailed(re.getErrorCode());
} catch (Exception e) {
    Log.error(e);
    result = Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION);
}
return result;
}

```

```

/**
 * ()
 * Calculate the actual amount of the transaction.
 */
private void calTransactionValue(TransactionDto txDto) {
    if (txDto == null) {
        return;
    }
    List<InputDto> inputDtoList = txDto.getInputs();
    Set<String> inputAdressSet = new HashSet<>(inputDtoList.size());
    for (InputDto inputDto : inputDtoList) {
        inputAdressSet.add(inputDto.getAddress());
    }
    Na value = Na.ZERO;
    List<OutputDto> outputDtoList = txDto.getOutputs();
    for (OutputDto outputDto : outputDtoList) {
        if (inputAdressSet.contains(outputDto.getAddress())) {
            continue;
        }
        value = value.add(Na.valueOf(outputDto.getValue()));
    }
    txDto.setValue(value.getValue());
}

```

@POST

@Path("/multiAccount/createMultiTransfer")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "", notes = "result.data: resultJson ")

@ApiResponses(value = {@ApiResponse(code = 200, message = "success")

})

```

public RpcClientResult createTransfer(@ApiParam(name = "form", value = "", required = true)
CreateP2shTransactionForm form) {

```

```

    if (NulsContext.MAIN_NET_VERSION <= 1) {
        return Result.getFailed(KernelErrorCode.VERSION_TOO_LOW).toRpcClientResult();
    }

```

```

    List<MultipleAddressTransferModel> toModelList = new ArrayList<>();

```

```

    if (form == null) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }

```

```

    if (!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }

```

```

    }
    if (!AddressTool.validAddress(form.getSignAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (form.getOutputs() == null || form.getOutputs().size() == 0) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    for (MultipleTxToDto to : form.getOutputs()) {
        if
(Na.valueOf(to.getAmount()).isLessThan(ProtocolConstant.MINIMUM_TRANSFER_AMOUNT)) {
            return
Result.getFailed(TransactionErrorCode.TOO_SMALL_AMOUNT).toRpcClientResult();
        }
        MultipleAddressTransferModel model = new MultipleAddressTransferModel();
        if (!AddressTool.validAddress(to.getToAddress())) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        model.setAddress(AddressTool.getAddress(to.getToAddress()));
        model.setAmount(to.getAmount());
        toModelList.add(model);
    }
    Result result = accountLedgerService.createP2shTransfer(form.getAddress(),
form.getSignAddress(), toModelList, form.getPassword(), form.getRemark());
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("txData", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

@POST

@Path("/multiAccount/signMultiTransaction")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "", notes = "result.data: resultJson ")

@ApiResponse(value = {@ApiResponse(code = 200, message = "success")

})

```

public RpcClientResult signMultiTransaction(@ApiParam(name = "form", value = "", required =
true) SignMultiTransactionForm form) {
    if (NulsContext.MAIN_NET_VERSION <= 1) {
        return Result.getFailed(KernelErrorCode.VERSION_TOO_LOW).toRpcClientResult();
    }
}

```

```

    if (form == null) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(form.getSignAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (form.getTxdata() == null || form.getTxdata().length() == 0) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Result result = accountLedgerService.signMultiTransaction(form.getSignAddress(),
form.getPassword(), form.getTxdata());
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("txData", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

@POST
@Path("/multiAccount/getSignType")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: resultJson ")
@ApiResponses(value = {@ApiResponse(code = 200, message = "success")
})
public RpcClientResult getSignatureType(@ApiParam(name = "utxoList", value = "", required =
true)

        @QueryParam("utxoList") List<String> utxoList) {
    if (utxoList == null || utxoList.size() == 0) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Result result = accountLedgerService.getSignatureType(utxoList);
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("signType", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}
}

```





```

        return "createMultiTransfer <address> <signAddress>
<toAddress>,<toamount>;....;<toAddress><toamount> [remark] -createMultiTransfer-";
    }

```

@Override

```

public boolean argsValidate(String[] args) {
    int length = args.length;
    if(length != 4 && length != 5) {
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    if (StringUtils.isBlank(args[1]) || StringUtils.isBlank(args[2])) {
        return false;
    }
    if(!CreateP2shTransactionForm.validToData(args[3])){
        return false;
    }
    return true;
}

```

@Override

```

public CommandResult execute(String[] args) {
    RpcClientResult res = CommandHelper.getPassword(args[2], restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("address",args[1]);
    parameters.put("signAddress",args[2]);
    parameters.put("outputs",CreateP2shTransactionForm.getTodata(args[3]));
    if(args.length == 5){
        parameters.put("remark",args[4]);
    }
    parameters.put("password",password);
    RpcClientResult result = restFul.post("/accountledger/multiAccount/createMultiTransfer",
parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
}

```

```

        return CommandResult.getResult(CommandResult.dataMultiTransformValue(result));
    }
}

```

24:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\cmd\GetAccountTxListProcessor.java  
\*/

```
package io.nuls.accout.ledger.rpc.cmd;
```

```

import io.nuls.core.tools.date.DateUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

```

```

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

```

```

/**
 * @author: Charlie
 */

```

```
public class GetAccountTxListProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```

    @Override
    public String getCommand() {
        return "gettxlist";
    }

```

```

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
            .newLine("\t<address>    address -required")
    }

```

```

        .newLine("\t<pageNumber>    pageNumber -required")
        .newLine("\t<pageSize>    pageSize -required");
    return bulider.toString();
}

```

```

@Override
public String getCommandDescription() {
    return "gettxlist <address> <pageNumber> <pageSize> --get the transaction information list
by address";
}

```

```

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length < 4 || length > 5) {
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    if (!AddressTool.validAddress(args[1])) {
        return false;
    }
    if (!StringUtils.isNumeric(args[2]) || !StringUtils.isNumeric(args[3])) {
        return false;
    }
    if (args.length == 5) {
        if (!StringUtils.isNumeric(args[4])) {
            return false;
        }
    }
    return true;
}

```

```

@Override
public CommandResult execute(String[] args) {
    int type = 0;
    int pageNumber = 0;
    int pageSize = 0;
    if (args.length == 4) {
        pageNumber = Integer.parseInt(args[2]);
        pageSize = Integer.parseInt(args[3]);
    }
}

```

```

    } else {
        type = Integer.parseInt(args[2]);
        pageNumber = Integer.parseInt(args[3]);
        pageSize = Integer.parseInt(args[4]);
    }
    String address = args[1];
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("type", type);
    parameters.put("pageNumber", pageNumber);
    parameters.put("pageSize", pageSize);
    RpcClientResult result = restFul.get("/accountledger/tx/list/" + address, parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    List<Map<String, Object>> list = (List<Map<String,
Object>>)((Map)result.getData()).get("list");
    for(Map<String, Object> map : list){
        map.put("time", DateUtil.convertDate(new Date((Long)map.get("time"))));
        map.put("txType", CommandHelper.txTypeExplain((Integer)map.get("txType")));
    }
    result.setData(list);
    return CommandResult.getResult(result);
}
}

```

25:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\cmd\GetUTXOProcessor.java  
\*/

```

package io.nuls.accout.ledger.rpc.cmd;

import io.nuls.kernel.model.Address;
import io.nuls.core.tools.date.DateUtil;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.RestFulUtils;

```

```

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author: Charlie
 */
public class GetUTXOProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "getutxo";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address>    the account address - Required")
            .newLine("\t<pageNumber>  pageNumber -required")
            .newLine("\t<pageSize>   pageSize -required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getutxo <address> <pageNumber> <pageSize> -- get utxo list ";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 4) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNotNull(args)) {
            return false;
        }
        if (!AddressTool.validAddress(args[1])) {

```

```

        return false;
    }
    if (!StringUtils.isNumeric(args[2]) || !StringUtils.isNumeric(args[3])) {
        return false;
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    int pageNumber = Integer.parseInt(args[2]);
    int pageSize = Integer.parseInt(args[3]);
    String address = args[1];
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("pageNumber", pageNumber);
    parameters.put("pageSize", pageSize);
    RpcClientResult result = restFul.get("/accountledger/utxo/lock/" + address, parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    List<Map<String, Object>> list = (List<Map<String,
Object>>)((Map)result.getData()).get("list");
    for(Map<String, Object> map : list){
        map.put("value", CommandHelper.naToNuls(map.get("value")));
        map.put("createTime", DateUtil.convertDate(new Date((Long)map.get("createTime"))));
        map.put("txType", CommandHelper.txTypeExplain((Integer)map.get("txType")));
    }
    result.setData(list);
    return CommandResult.getResult(result);
}
}

```

26:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\cmd\SignMultiTransactionProcess.java

\*/

```
package io.nuls.accout.ledger.rpc.cmd;
```

```

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;

```

```

import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

import java.util.HashMap;
import java.util.Map;

/**
 * @author: tag
 */
public class SignMultiTransactionProcess implements CommandProcessor {
    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "signMultiTransaction";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> \t\tsource address - Required")
            .newLine("\t<txdata> \t\ttransaction data - Required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "signMultiTransfer <signAddress> <txdata> -sign a multiTransfer";
    }

    @Override
    public boolean argsValidate(String[] args) {
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        return StringUtils.validSign(args);
    }

    @Override
    public CommandResult execute(String[] args) {

```

```

    RpcClientResult res = CommandHelper.getPassword(args[1], restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("signAddress",args[1]);
    parameters.put("txdata",args[2]);
    parameters.put("password",password);
    RpcClientResult result = restFul.post("/accountledger/multiAccount/signMultiTransaction",
parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(CommandResult.dataMultiTransformValue(result));
}
}

```

27:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\cmd\TransferProcessor.java  
 \*/

```

package io.nuls.accout.ledger.rpc.cmd;

```

```

import io.nuls.accout.ledger.rpc.form.TransferForm;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

```

```

import java.util.HashMap;
import java.util.Map;

```

```

/**

```

```

 * @author: Charlie

```

```

 */

```

```

public class TransferProcessor implements CommandProcessor {

```



```
private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
private ThreadLocal<TransferForm> paramsData = new ThreadLocal<>();
```

```
@Override
```

```
public String getCommand() {  
    return "transfer";  
}
```

```
@Override
```

```
public String getHelp() {  
    CommandBuilder builder = new CommandBuilder();  
    builder.newLine(getCommandDescription())  
        .newLine("\t<address> \t\tsource address - Required")  
        .newLine("\t<toaddress> \treceiving address - Required")  
        .newLine("\t<amount> \t\tamount, you can have up to 8 valid digits after the decimal  
point - Required")  
        .newLine("\t[remark] \t\tremark - ");  
    return builder.toString();  
}
```

```
@Override
```

```
public String getCommandDescription() {  
    return "transfer <address> <toAddress> <amount> [remark] --transfer";  
}
```

```
@Override
```

```
public boolean argsValidate(String[] args) {  
    boolean result;  
    do {  
        int length = args.length;  
        if (length != 4 && length != 5) {  
            result = false;  
            break;  
        }  
        if (!CommandHelper.checkArgsIsNull(args)) {  
            result = false;  
            break;  
        }  
  
        if (!StringUtil.isNuls(args[3])) {
```

```

        result = false;
        break;
    }
    TransferForm form = getTransferForm(args);
    if(null == form){
        result = false;
        break;
    }
    paramsData.set(form);
    result = StringUtils.isNotBlank(form.getToAddress());
    if (!result) {
        break;
    }
    result = form.getAmount() > 0;
} while (false);
return result;
}

```

```

private TransferForm getTransferForm(String[] args) {
    TransferForm form = null;
    Long amount = null;
    try {
        Na na = Na.parseNuls(args[3]);
        if (na != null) {
            amount = na.getValue();
            form = new TransferForm();
        } else {
            return null;
        }
    } catch (Exception e) {
        return null;
    }
    switch (args.length) {
        case 4:
            form.setAddress(args[1]);
            form.setToAddress(args[2]);
            form.setAmount(amount);
            break;
        case 5:
            form.setAddress(args[1]);
            form.setToAddress(args[2]);
            form.setAmount(amount);

```

```

        form.setRemark(args[4]);
        break;
    }
    return form;
}

```

@Override

```

public CommandResult execute(String[] args) {
    TransferForm form = paramsData.get();
    if (null == form) {
        form = getTransferForm(args);
    }
    String address = form.getAddress();
    RpcClientResult res = CommandHelper.getPassword(address, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("address", form.getAddress());
    parameters.put("toAddress", form.getToAddress());
    parameters.put("password", password);
    parameters.put("amount", form.getAmount());
    parameters.put("remark", form.getRemark());
    RpcClientResult result = restFul.post("/accountledger/transfer", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(CommandResult.dataTransformValue(result));
}
}

```

28:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\ChangeToWholeFromDto.java

\*/

```
package io.nuls.accout.ledger.rpc.dto;
```

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

/\*\*

```

* @author Facjas
*/
@ApiModel(value = "from")
public class ChangeToWholeFromDto {

    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;

    @ApiModelProperty(name = "amount", value = "", required = true)
    private long amount;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public long getAmount() {
        return amount;
    }

    public void setAmount(long amount) {
        this.amount = amount;
    }

}

```

29:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\ChangeToWholeToDto.java

```

*/
package io.nuls.accout.ledger.rpc.dto;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author Facjas
 */
@ApiModel(value = "to")
public class ChangeToWholeToDto {

```

```

@ApiModelProperty(name = "toAddress", value = "", required = true)
private String toAddress;

@ApiModelProperty(name = "amount", value = "", required = true)
private long amount;

public String getToAddress() {
    return toAddress;
}

public void setToAddress(String toAddress) {
    this.toAddress = toAddress;
}

public long getAmount() {
    return amount;
}

public void setAmount(long amount) {
    this.amount = amount;
}

}

30:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\InputDto.java
*/

package io.nuls.accout.ledger.rpc.dto;

import io.nuls.account.ledger.base.util.AccountLegerUtils;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "inputJSON")
public class InputDto {

```

```
@ApiModelProperty(name = "fromHash", value = "outputtxHash")
```

```
private String fromHash;
```

```
@ApiModelProperty(name = "fromIndex", value = "outputoutIndex")
```

```
private Integer fromIndex;
```

```
@ApiModelProperty(name = "address", value = "")
```

```
private String address;
```

```
@ApiModelProperty(name = "value", value = "")
```

```
private Long value;
```

```
@ApiModelProperty(name = "lockTime", value = "")
```

```
private Long lockTime = 0L;
```

```
public InputDto() {
```

```
    this.lockTime = 0L;
```

```
}
```

```
public InputDto(Coin input) {
```

```
    this.fromHash = AccountLegerUtils.getTxHash(input.getOwner());
```

```
    this.fromIndex = AccountLegerUtils.getIndex(input.getOwner());
```

```
    //this.address = AddressTool.getStringAddressByBytes(input.getFrom().());
```

```
    this.address = AddressTool.getStringAddressByBytes(input.getFrom().getAddress());
```

```
    this.value = input.getFrom().getNa().getValue();
```

```
    this.lockTime = input.getFrom().getLockTime();
```

```
}
```

```
public String getAddress() {
```

```
    return address;
```

```
}
```

```
public void setAddress(String address) {
```

```
    this.address = address;
```

```
}
```

```
public Long getValue() {
```

```
    return value;
```

```
}
```

```
public void setValue(Long value) {
```

```
    this.value = value;
```

```

    }

    public String getFromHash() {
        return fromHash;
    }

    public void setFromHash(String fromHash) {
        this.fromHash = fromHash;
    }

    public Integer getFromIndex() {
        return fromIndex;
    }

    public void setFromIndex(Integer fromIndex) {
        this.fromIndex = fromIndex;
    }

    public Long getLockTime() {
        return lockTime;
    }

    public void setLockTime(Long lockTime) {
        this.lockTime = lockTime;
    }
}

```

31:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\MultipleTxFromDto.java

\*/

```
package io.nuls.accout.ledger.rpc.dto;
```

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
@ApiModel(value = "from")
```

```
public class MultipleTxFromDto {
```

```
    @ApiModelProperty(name = "address", value = "", required = true)
```

```
    private String address;
```

```
    public String getAddress() {
```

```

        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}

```

32:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\MultipleTxToDto.java

```

*/

```

```

package io.nuls.accout.ledger.rpc.dto;

```

```

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

```

```

/**

```

```

 * @author Facjas

```

```

 */

```

```

@ApiModel(value = "to")

```

```

public class MultipleTxToDto {

```

```

    @ApiModelProperty(name = "toAddress", value = "", required = true)

```

```

    private String toAddress;

```

```

    @ApiModelProperty(name = "amount", value = "", required = true)

```

```

    private long amount;

```

```

    public String getToAddress() {

```

```

        return toAddress;

```

```

    }

```

```

    public void setToAddress(String toAddress) {

```

```

        this.toAddress = toAddress;

```

```

    }

```

```

    public long getAmount() {

```

```

        return amount;

```

```

    }

```



```

    public void setAmount(long amount) {
        this.amount = amount;
    }
}

33:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\OutputDto.java
*/

package io.nuls.accout.ledger.rpc.dto;

import io.nuls.kernel.model.Coin;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "outputJSON")
public class OutputDto {

    @ApiModelProperty(name = "txHash", value = "hash")
    private String txHash;

    @ApiModelProperty(name = "index", value = "")
    private Integer index;

    @ApiModelProperty(name = "address", value = "")
    private String address;

    @ApiModelProperty(name = "script", value = "")
    private String script;

    @ApiModelProperty(name = "value", value = "")
    private Long value;

    @ApiModelProperty(name = "lockTime", value = "")
    private Long lockTime;

    @ApiModelProperty(name = "status",
        value = " 0:usable(), 1:timeLock(), 2:consensusLock(), 3:spent()")
    private Integer status;

```

```
public OutputDto() {
```

```
}
```

```
public OutputDto(Coin output) {
```

```
    this.address = AddressTool.getStringAddressByBytes(output.getAddress());
```

```
    this.script = AddressTool.getStringAddressByBytes(output.getOwner());
```

```
    this.value = output.getNa().getValue();
```

```
    this.lockTime = output.getLockTime();
```

```
}
```

```
public Integer getIndex() {
```

```
    return index;
```

```
}
```

```
public void setIndex(Integer index) {
```

```
    this.index = index;
```

```
}
```

```
public String getAddress() {
```

```
    return address;
```

```
}
```

```
public void setAddress(String address) {
```

```
    this.address = address;
```

```
}
```

```
public Long getValue() {
```

```
    return value;
```

```
}
```

```
public void setValue(Long value) {
```

```
    this.value = value;
```

```
}
```

```
public Long getLockTime() {
```

```
    return lockTime;
```

```
}
```

```
public void setLockTime(Long lockTime) {
```

```
    this.lockTime = lockTime;
```

```
}
```

```

    public Integer getStatus() {
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }

    public String getTxHash() {
        return txHash;
    }

    public void setTxHash(String txHash) {
        this.txHash = txHash;
    }
}

```

34:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\TransactionDto.java  
 \*/

```

package io.nuls.accout.ledger.rpc.dto;

import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.TxStatusEnum;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.CoinData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.model.TransactionLogicData;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

import static io.nuls.core.tools.str.StringUtils.EMPTY;

```

```
/**
 * @desription:
 * @author: PierreLuo
 */
@ApiModel(value = "transactionDtoJSON")
public class TransactionDto {

    @ApiModelProperty(name = "hash", value = "hash")
    private String hash;

    @ApiModelProperty(name = "type", value = " ")
    private Integer type;

    @ApiModelProperty(name = "time", value = "")
    private Long time;

    @ApiModelProperty(name = "blockHeight", value = "")
    private Long blockHeight;

    @ApiModelProperty(name = "fee", value = "")
    private Long fee;

    @ApiModelProperty(name = "value", value = "")
    private Long value;

    @ApiModelProperty(name = "remark", value = "")
    private String remark;

    @ApiModelProperty(name = "scriptSig", value = "")
    private String scriptSig;

    @ApiModelProperty(name = "status", value = " 0:unConfirm(), 1:confirm()")
    private Integer status;

    @ApiModelProperty(name = "confirmCount", value = "")
    private Long confirmCount;

    @ApiModelProperty(name = "size", value = "")
    private int size;

    @ApiModelProperty(name = "inputs", value = "")
```

```
private List<InputDto> inputs;
```

```
@ApiModelProperty(name = "outputs", value = "")
```

```
private List<OutputDto> outputs;
```

```
@ApiModelProperty(name = "txDataHexString", value = "txDataHex")
```

```
private String txDataHexString;
```

```
public TransactionDto(Transaction tx) {
```

```
    long bestBlockHeight = NulsContext.getInstance().getBestBlock().getHeader().getHeight();
```

```
    this.hash = tx.getHash().getDigestHex();
```

```
    this.type = tx.getType();
```

```
    this.time = tx.getTime();
```

```
    this.blockHeight = tx.getBlockHeight();
```

```
    this.fee = tx.getFee().getValue();
```

```
    this.size = tx.getSize();
```

```
    if (this.blockHeight > 0 || TxStatusEnum.CONFIRMED.equals(tx.getStatus())) {
```

```
        this.confirmCount = bestBlockHeight - this.blockHeight;
```

```
    } else {
```

```
        this.confirmCount = 0L;
```

```
    }
```

```
    if (TxStatusEnum.CONFIRMED.equals(tx.getStatus())) {
```

```
        this.status = 1;
```

```
    } else {
```

```
        this.status = 0;
```

```
    }
```

```
    if (tx.getRemark() != null) {
```

```
        try {
```

```
            this.setRemark(new String(tx.getRemark(), NulsConfig.DEFAULT_ENCODING));
```

```
        } catch (UnsupportedEncodingException e) {
```

```
            this.setRemark(Hex.encode(tx.getRemark()));
```

```
        }
```

```
    }
```

```
    if (tx.getTransactionSignature() != null) {
```

```
        this.setScriptSig(Hex.encode(tx.getTransactionSignature()));
```

```
    }
```

```
CoinData coinData = tx.getCoinData();
```

```
List<InputDto> inputs = new ArrayList<>();
```

```
if(coinData != null) {
```

```
    List<Coin> froms = coinData.getFrom();
```

```

        for(Coin from : froms) {
            inputs.add(new InputDto(from));
        }
    }
    this.inputs = inputs;

    this.txDataHexString = EMPTY;
    TransactionLogicData txData = tx.getTxData();
    if(txData != null) {
        try {
            byte[] serialize = txData.serialize();
            this.txDataHexString = Hex.encode(serialize);
        } catch (IOException e) {
            Log.error(e);
        }
    }
}

```

```

public String getHash() {
    return hash;
}

```

```

public void setHash(String hash) {
    this.hash = hash;
}

```

```

public Integer getType() {
    return type;
}

```

```

public void setType(Integer type) {
    this.type = type;
}

```

```

public Long getTime() {
    return time;
}

```

```

public void setTime(Long time) {
    this.time = time;
}

```

```
public Long getBlockHeight() {  
    return blockHeight;  
}
```

```
public void setBlockHeight(Long blockHeight) {  
    this.blockHeight = blockHeight;  
}
```

```
public Long getFee() {  
    return fee;  
}
```

```
public void setFee(Long fee) {  
    this.fee = fee;  
}
```

```
public Long getValue() {  
    return value;  
}
```

```
public void setValue(Long value) {  
    this.value = value;  
}
```

```
public List<InputDto> getInputs() {  
    return inputs;  
}
```

```
public void setInputs(List<InputDto> inputs) {  
    this.inputs = inputs;  
}
```

```
public List<OutputDto> getOutputs() {  
    return outputs;  
}
```

```
public void setOutputs(List<OutputDto> outputs) {  
    this.outputs = outputs;  
}
```

```
public String getRemark() {  
    return remark;  
}
```

```
}

public void setRemark(String remark) {
    this.remark = remark;
}

public String getScriptSig() {
    return scriptSig;
}

public void setScriptSig(String scriptSig) {
    this.scriptSig = scriptSig;
}

public Integer getStatus() {
    return status;
}

public void setStatus(Integer status) {
    this.status = status;
}

public Long getConfirmCount() {
    return confirmCount;
}

public void setConfirmCount(Long confirmCount) {
    this.confirmCount = confirmCount;
}

public int getSize() {
    return size;
}

public void setSize(int size) {
    this.size = size;
}

public String getTxDataHexString() {
    return txDataHexString;
}
```



```

    public void setTxDataHexString(String txDataHexString) {
        this.txDataHexString = txDataHexString;
    }
}

```

35:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\TransactionInfoDto.java  
\*/

```

package io.nuls.accout.ledger.rpc.dto;

```

```

import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.utils.AddressTool;

```

```

public class TransactionInfoDto {

```

```

    private String txHash;

```

```

    private long blockHeight;

```

```

    private long time;

```

```

    private int txType;

```

```

    private byte status;

```

```

    private String info;

```

```

    /**

```

```

     * contract address

```

```

     */

```

```

    private String contractAddress;

```

```

    /**

```

```

     * contract token symbol

```

```

     */

```

```

    private String symbol;

```

```

    public TransactionInfoDto() {

```

```

    }

```

```
public TransactionInfoDto(TransactionInfo info) {
    this.txHash = info.getTxHash().getDigestHex();
    this.blockHeight = info.getBlockHeight();
    this.time = info.getTime();
    this.status = info.getStatus();
    this.txType = info.getTxType();
    this.info = info.getInfo();
    if(info.getContractAddress() != null) {
        this.contractAddress = AddressTool.getStringAddressByBytes(info.getContractAddress());
    }
    this.symbol = info.getSymbol();
}

public String getTxHash() {
    return txHash;
}

public void setTxHash(String txHash) {
    this.txHash = txHash;
}

public long getBlockHeight() {
    return blockHeight;
}

public void setBlockHeight(long blockHeight) {
    this.blockHeight = blockHeight;
}

public long getTime() {
    return time;
}

public void setTime(long time) {
    this.time = time;
}

public int getTxType() {
    return txType;
}

public void setTxType(int txType) {
```

```

        this.txType = txType;
    }

    public byte getStatus() {
        return status;
    }

    public void setStatus(byte status) {
        this.status = status;
    }

    public String getInfo() {
        return info;
    }

    public void setInfo(String info) {
        this.info = info;
    }

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    public String getSymbol() {
        return symbol;
    }

    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }
}

```

```

36:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\dto\UtxoDto.java
*/

```

```

package io.nuls.accout.ledger.rpc.dto;

```

```
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Transaction;

public class UtxoDto {

    private String txHash;

    private long createTime;

    private int txType;

    private long lockTime;

    private long value;

    public UtxoDto() {

    }

    public UtxoDto(Coin coin, Transaction tx) {
        this.txHash = tx.getHash().getDigestHex();
        this.createTime = tx.getTime();
        this.txType = tx.getType();
        this.lockTime = coin.getLockTime();
        this.value = coin.getNa().getValue();
    }

    public String getTxHash() {
        return txHash;
    }

    public void setTxHash(String txHash) {
        this.txHash = txHash;
    }

    public long getCreateTime() {
        return createTime;
    }

    public void setCreateTime(long createTime) {
        this.createTime = createTime;
    }
}
```

```

    public int getTxType() {
        return txType;
    }

    public void setTxType(int txType) {
        this.txType = txType;
    }

    public long getLockTime() {
        return lockTime;
    }

    public void setLockTime(long lockTime) {
        this.lockTime = lockTime;
    }

    public long getValue() {
        return value;
    }

    public void setValue(long value) {
        this.value = value;
    }
}

37:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\form\BroadHexTxForm.java
*/
package io.nuls.accout.ledger.rpc.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import javax.ws.rs.QueryParam;

/**
 * @author Facjas
 */
@ApiModel(value = "")
public class BroadHexTxForm {

```

```

@ApiModelProperty(name = "txHex", value = "", required = true)
@QueryParam("txHex")
private String txHex;

public void setTxHex(String txHex) {
    this.txHex = txHex;
}

public String getTxHex() {
    return txHex;
}
}

38:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\form\ChangeToWholeTransactionForm.java
*/
package io.nuls.accout.ledger.rpc.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "")
public class ChangeToWholeTransactionForm {

    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;

    @ApiModelProperty(name = "password", value = "")
    private String password;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getAddress() {
        return address;
    }
}

```

```

    public void setAddress(String address) {
        this.address = address;
    }
}

```

39:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\form\CreateP2shTransactionForm.java  
\*/

```
package io.nuls.accout.ledger.rpc.form;
```

```

import io.nuls.accout.ledger.rpc.dto.MultipleTxToDto;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

```

```

import java.util.ArrayList;
import java.util.List;

```

```
/**
```

```
 * @author tag
```

```
*/
```

```
@ApiModel(value = "form")
```

```
public class CreateP2shTransactionForm {
```

```
    @ApiModelProperty(name = "address", value = "", required = true)
```

```
    private String address;
```

```
    @ApiModelProperty(name = "signAddress", value = "", required = true)
```

```
    private String signAddress;
```

```
    @ApiModelProperty(name = "outputs", value = "", required = true)
```

```
    private List<MultipleTxToDto> outputs;
```

```
    @ApiModelProperty(name = "password", value = "", required = false)
```

```
    private String password;
```

```
    @ApiModelProperty(name = "remark", value = "")
```

```
    private String remark;
```

```
    public String getAddress() {
```

```
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getSignAddress() {
        return signAddress;
    }

    public void setSignAddress(String signAddress) {
        this.signAddress = signAddress;
    }

    public List<MultipleTxToDto> getOutputs() {
        return outputs;
    }

    public void setOutputs(List<MultipleTxToDto> outputs) {
        this.outputs = outputs;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }

    public static boolean validToData(String todata){
        if(StringUtils.isBlank(todata)){
            return false;
        }
    }
```



```

    }
    //to
    String[] dataList = todata.split(";");
    if(dataList == null || dataList.length == 0){
        return false;
    }
    for (String data:dataList) {
        //to
        String[] separateData = data.split(",");
        if(separateData == null || separateData.length != 2){
            return false;
        }
        if (!AddressTool.validAddress(separateData[0]) || !StringUtil.isNuls(separateData[1])) {
            return false;
        }
    }
    return true;
}

```

```

public static List<MultipleTxToDto> getTodata(String todata){
    List<MultipleTxToDto> toDatas = new ArrayList<>();
    String[] dataList = todata.split(";");
    long toAmount;
    for (String data:dataList) {
        //to
        MultipleTxToDto toData = new MultipleTxToDto();
        String[] separateData = data.split(",");
        Na toNa = Na.parseNuls(separateData[1]);
        toAmount = toNa.getValue();
        if(toAmount <= 0) {
            return null;
        }
        toData.setAmount(toAmount);
        toData.setToAddress(separateData[0]);
        toDatas.add(toData);
    }
    return toDatas;
}
}

```

```
*/
```

```
package io.nuls.accout.ledger.rpc.form;
```

```
import io.swagger.annotations.ApiModel;
```

```
import io.swagger.annotations.ApiModelProperty;
```

```
@ApiModel(value = "DAPP")
```

```
public class DataTransactionForm {
```

```
    @ApiModelProperty(name = "address", value = "", required = true)
```

```
    private String address;
```

```
    @ApiModelProperty(name = "password", value = "", required = false)
```

```
    private String password;
```

```
    @ApiModelProperty(name = "data", value = "", required = true)
```

```
    private String data;
```

```
    @ApiModelProperty(name = "remark", value = "", required = false)
```

```
    private String remark;
```

```
    public String getPassword() {
```

```
        return password;
```

```
    }
```

```
    public void setPassword(String password) {
```

```
        this.password = password;
```

```
    }
```

```
    public String getAddress() {
```

```
        return address;
```

```
    }
```

```
    public void setAddress(String address) {
```

```
        this.address = address;
```

```
    }
```

```
    public String getData() {
```

```
        return data;
```

```
    }
```

```
    public void setData(String data) {
```

```

        this.data = data;
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }
}

41:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\form\MulitpleTransactionForm.java
*/
package io.nuls.accout.ledger.rpc.form;

import io.nuls.accout.ledger.rpc.dto.MulipleTxFromDto;
import io.nuls.accout.ledger.rpc.dto.MultipleTxToDto;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import javax.ws.rs.QueryParam;
import java.util.List;

@ApiModel(value = "form")
public class MulitpleTransactionForm {

    @ApiModelProperty(name = "inputs", value = "", required = true)
    private List<MulipleTxFromDto> inputs;

    @ApiModelProperty(name = "outputs", value = "", required = true)
    private List<MultipleTxToDto> outputs;

    @ApiModelProperty(name = "remark", value = "")
    private String remark;

    @ApiModelProperty(name = "password", value = "")
    private String password;

    public List<MulipleTxFromDto> getInputs() {

```

```

        return inputs;
    }

    public List<MultipleTxToDto> getOutputs() {
        return outputs;
    }

    public void setOutputs(List<MultipleTxToDto> outputs) {
        this.outputs = outputs;
    }

    public void setInputs(List<MultipleTxFromDto> inputs) {
        this.inputs = inputs;
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

```

```

42:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\form\SignMultiTransactionForm.java
*/
package io.nuls.accout.ledger.rpc.form;

```

```

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

```

```

/**

```

```

* @author tag
*/
@ApiModel(value = "form")
public class SignMultiTransactionForm {
    @ApiModelProperty(name = "signAddress", value = "", required = true)
    private String signAddress;

    @ApiModelProperty(name = "password", value = "", required = false)
    private String password;

    @ApiModelProperty(name = "txdata", value = "")
    private String txdata;

    public String getSignAddress() {
        return signAddress;
    }

    public void setSignAddress(String signAddress) {
        this.signAddress = signAddress;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getTxdata() {
        return txdata;
    }

    public void setTxdata(String txdata) {
        this.txdata = txdata;
    }
}

```

```

43:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\form\TransactionForm.java
*/
package io.nuls.accout.ledger.rpc.form;

```

```
import io.nuls.accout.ledger.rpc.dto.InputDto;
import io.nuls.accout.ledger.rpc.dto.OutputDto;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
import java.util.List;
```

```
/**
```

```
 * @author Facjas
```

```
 */
```

```
@ApiModel(value = "")
```

```
public class TransactionForm {
```

```
    @ApiModelProperty(name = "inputs", value = "", required = true)
```

```
    private List<InputDto> inputs;
```

```
    @ApiModelProperty(name = "outputs", value = "", required = true)
```

```
    private List<OutputDto> outputs;
```

```
    @ApiModelProperty(name = "remark", value = "")
```

```
    private String remark;
```

```
    public TransactionForm() {
```

```
    }
```

```
    public List<InputDto> getInputs() {
```

```
        return inputs;
```

```
    }
```

```
    public void setInputs(List<InputDto> inputs) {
```

```
        this.inputs = inputs;
```

```
    }
```

```
    public List<OutputDto> getOutputs() {
```

```
        return outputs;
```

```
    }
```

```
    public void setOutputs(List<OutputDto> outputs) {
```

```
        this.outputs = outputs;
```

```
    }
```

```

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }
}

44:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\form\TransactionHexForm.java
*/
package io.nuls.accout.ledger.rpc.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import javax.ws.rs.QueryParam;

/**
 * @author Facjas
 */
@ApiModel(value = "")
public class TransactionHexForm {

    @ApiModelProperty(name = "txHex", value = "", required = true)
    @QueryParam("txHex")
    private String txHex;

    @ApiModelProperty(name = "address", value = "")
    @QueryParam("address")
    private String address;

    @ApiModelProperty(name = "priKey", value = "")
    @QueryParam("priKey")
    private String priKey;

    @ApiModelProperty(name = "password", value = "")
    @QueryParam("password")
    private String password;

```

```

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public void setTxHex(String txHex) {
    this.txHex = txHex;
}

public String getTxHex() {
    return txHex;
}

public void setPriKey(String priKey) {
    this.priKey = priKey;
}

public String getPriKey() {
    return priKey;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}
}

```

45:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\main\java\io\nuls\accout\ledger\rpc\form\TransferFeeForm.java  
\*/

```
package io.nuls.accout.ledger.rpc.form;
```

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
import javax.ws.rs.QueryParam;
```



```
/**
 * @author Facjas
 */
@ApiModel(value = "")
public class TransferFeeForm {

    @ApiModelProperty(name = "address", value = "", required = true)
    @QueryParam("address")
    private String address;

    @ApiModelProperty(name = "toAddress", value = "", required = true)
    @QueryParam("toAddress")
    private String toAddress;

    @ApiModelProperty(name = "amount", value = "", required = true)
    @QueryParam("amount")
    private long amount;

    @ApiModelProperty(name = "remark", value = "", required = true)
    @QueryParam("remark")
    private String remark;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getToAddress() {
        return toAddress;
    }

    public void setToAddress(String toAddress) {
        this.toAddress = toAddress;
    }

    public long getAmount() {
        return amount;
    }
}
```

```

    public void setAmount(long amount) {
        this.amount = amount;
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }
}

46:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
rpc\src\main\java\io\nuls\accout\ledger\rpc\form\TransferForm.java
*/
package io.nuls.accout.ledger.rpc.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author Facjas
 */
@ApiModel(value = "")
public class TransferForm {

    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;

    @ApiModelProperty(name = "toAddress", value = "", required = true)
    private String toAddress;

    @ApiModelProperty(name = "password", value = "", required = false)
    private String password;

    @ApiModelProperty(name = "amount", value = "", required = true)
    private long amount;

    @ApiModelProperty(name = "remark", value = "", required = true)
    private String remark;

```

```
public String getAddress() {  
    return address;  
}  
  
public void setAddress(String address) {  
    this.address = address;  
}  
  
public String getToAddress() {  
    return toAddress;  
}  
  
public void setToAddress(String toAddress) {  
    this.toAddress = toAddress;  
}  
  
public String getPassword() {  
    return password;  
}  
  
public void setPassword(String password) {  
    this.password = password;  
}  
  
public long getAmount() {  
    return amount;  
}  
  
public void setAmount(long amount) {  
    this.amount = amount;  
}  
  
public String getRemark() {  
    return remark;  
}  
  
public void setRemark(String remark) {  
    this.remark = remark;  
}  
}
```

```
47:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-  
rpc\src\main\java\io\nuls\accout\ledger\rpc\util\UtxoDtoComparator.java  
*/
```

```
package io.nuls.accout.ledger.rpc.util;  
  
import io.nuls.accout.ledger.rpc.dto.UtxoDto;  
  
import java.util.Comparator;  
  
public class UtxoDtoComparator implements Comparator<UtxoDto> {  
  
    private static UtxoDtoComparator instance = new UtxoDtoComparator();  
  
    private UtxoDtoComparator() {  
  
    }  
  
    public static UtxoDtoComparator getInstance() {  
        return instance;  
    }  
  
    @Override  
    public int compare(UtxoDto o1, UtxoDto o2) {  
        if (o1.getCreateTime() < o2.getCreateTime()) {  
            return 1;  
        } else if (o1.getCreateTime() > o2.getCreateTime()) {  
            return -1;  
        }  
        return 0;  
    }  
}
```

```
48:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-  
rpc\src\test\java\io\nuls\account\ledger\BaseTest.java  
*/
```

```
package io.nuls.account.ledger;  
  
import io.nuls.core.tools.log.Log;  
import io.nuls.core.tools.str.StringUtils;  
  
import java.io.*;
```

```

import java.net.HttpURLConnection;
import java.net.URL;

public class BaseTest {

    public static String post(String url, final String param, String encoding) {
        StringBuffer sb = new StringBuffer();
        OutputStream os = null;
        InputStream is = null;
        InputStreamReader isr = null;
        BufferedReader br = null;
        // UTF-8
        if (StringUtils.isNull(encoding)) {
            encoding = "UTF-8";
        }
        try {
            URL u = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) u.openConnection();
            connection.setRequestProperty("Content-Type", "application/json");
            connection.setDoOutput(true);
            connection.setDoInput(true);
            connection.setRequestMethod("POST");

            connection.connect();

            os = connection.getOutputStream();
            os.write(param.getBytes(encoding));
            os.flush();
            is = connection.getInputStream();
            isr = new InputStreamReader(is, encoding);
            br = new BufferedReader(isr);
            String line;
            while ((line = br.readLine()) != null) {
                sb.append(line);
                sb.append("\n");
            }
        } catch (Exception ex) {
            System.err.println(ex);
        } finally {
            if (is != null) {
                try {
                    is.close();
                }
            }
        }
    }
}

```

```

        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (os != null) {
        try {
            os.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (isr != null) {
        try {
            isr.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (br != null) {
        try {
            br.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
}
return sb.toString();
}
}

```

49:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-rpc\src\test\java\io\nuls\account\ledger\rpc\MultiAddressTransferTest.java  
 \*/

```
package io.nuls.account.ledger.rpc;
```

```
import io.nuls.account.ledger.BaseTest;
import io.nuls.core.tools.json.JSONUtils;
```

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
```

```

public class MultiAddressTransferTest extends BaseTest {

    static long t,t1,t2,t3,t4;

    public static void main(String[] args) {
        MultiAddressTransferTest test = new MultiAddressTransferTest();
        List<Map> list = test.genOrLoadAddress();

        System.out.println(list.size());

        long time = System.currentTimeMillis();

        for(int i = 0 ; i < 10 ; i ++ ) {
            for (Map info : list) {
                Map map = test.sendOfflineTx(info, "Nse9Jxd1VdLWEoZxe3fWkXxus8TKgJyd");
                if (map == null || !(boolean) map.get("success")) {
                    System.err.println("3" + map);
                    continue;
                }
                info.put("value", info.get("balance"));
                info.put("txHash", ((Map) map.get("data")).get("value"));
                info.put("index", 1);
            }
        }

        System.out.println("");
        System.out.println("" + (System.currentTimeMillis() - time));
        System.out.println("t1 " + t1 / 1000000);
        System.out.println("t2 " + t2 / 1000000);
        System.out.println("t3 " + t3 / 1000000);

        try {
            ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("./address.txt"));
            oos.writeObject(list);
            oos.flush();
            oos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```
}
```

```
private Map sendOfflineTx(Map info, String toAddress) {
```

```
    t = System.nanoTime();
```

```
    Map map = createTx(info, toAddress);
```

```
    t1 += (System.nanoTime() - t);
```

```
    t = System.nanoTime();
```

```
    if(map == null || !(boolean) map.get("success")) {
```

```
        System.err.println("1" + map);
```

```
        return map;
```

```
    }
```

```
    map = singTx(info, map);
```

```
    t2 += (System.nanoTime() - t);
```

```
    t = System.nanoTime();
```

```
    if(map == null || !(boolean) map.get("success")) {
```

```
        System.err.println("2" + map);
```

```
        return map;
```

```
    }
```

```
    map = broadcastTx(map);
```

```
    t3 += (System.nanoTime() - t);
```

```
    return map;
```

```
}
```

```
private Map broadcastTx(Map map) {
```

```
    String hex = (String)((Map)map.get("data")).get("value");
```

```
    String param = "{\"txHex\":\"" + hex + "\"}";
```

```
    String url = "http://127.0.0.1:8001/api/accountledger/transaction/broadcast";
```

```
    String res = post(url, param, "utf-8");
```

```
    try {
```

```
        Map result = JSONUtils.json2map(res);
```

```
        return result;
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    return null;
```



```
}
```

```
private Map singTx(Map info, Map map) {
```

```
    String hex = (String)((Map)map.get("data")).get("value");
```

```
    String privateKey = (String) info.get("priKey");
```

```
    String param = "{\"txHex\":\"" + hex + "\", \"address\": \"" + info.get("address") + "\", \"priKey\": \"" + privateKey + "\", \"password\": \"\"}";
```

```
    String url = "http://127.0.0.1:8001/api/accountledger/transaction/sign";
```

```
    String res = post(url, param, "utf-8");
```

```
    try {
```

```
        Map result = JSONUtils.json2map(res);
```

```
        return result;
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    return null;
```

```
}
```

```
private Map createTx(Map info, String toAddress) {
```

```
    String address = (String) info.get("address");
```

```
    long value = (long) info.get("value");
```

```
    long balance = (value - 1100000L);
```

```
    String param = "{\"inputs\": [{\"fromHash\":\"" + info.get("txHash") + "\", \"fromIndex\": " + info.get("index") + ", \"address\": \"" + address + "\", \"value\": " + value + ", \"lockTime\": 0}], \"outputs\": [{\"address\": \"" + toAddress + "\", \"value\": 1000000, \"lockTime\": 0}, {\"address\": \"" + address + "\", \"value\": " + balance + ", \"lockTime\": 0}], \"remark\": \"\"}";
```

```
    String url = "http://127.0.0.1:8001/api/accountledger/transaction";
```

```
    String res = post(url, param, "utf-8");
```

```
    try {
```

```
        info.put("balance", balance);
```

```
        Map result = JSONUtils.json2map(res);
```

```
        return result;
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    return null;
```

```
}
```

```

private Map send(String fromAddress, String toAddress, long amount, String password, String
remark) {
    String param = "{\"address\": \"" + fromAddress + "\", \"toAddress\": \"" + toAddress + "\",
    \"password\": \"" + password + "\", \"amount\": \"" + amount + "\", \"remark\": \"" + remark + "\"}";
    String url = "http://127.0.0.1:8001/api/accountledger/transfer";
    String res = post(url, param, "utf-8");
    try {
        return JSONUtils.json2map(res);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

```

```

private List<Map> genOrLoadAddress() {
    List<Map> addressList = null;
    try {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream("./address.txt"));
        try {
            addressList = (List<Map>) ois.readObject();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        ois.close();
    } catch (FileNotFoundException fe) {

        addressList = new ArrayList<>();

        String param = "{\"count\": 100, \"password\": \"\"}";
        String url = "http://127.0.0.1:8001/api/account/offline";

        int count = 0;
        long amount = 10000000000L;
        for (int i = 0; i < 100; i++) {
            String res = post(url, param, "utf-8");

            try {
                Map map = JSONUtils.json2map(res);

                List<Map> list = ((List<Map>) ((Map) map.get("data")).get("list"));

                for(Map m : list) {

```

```

        count++;
        Map result = send("NsduWRoBQcdTw6vxBVmVWtLBxLSyaSVr", (String)
m.get("address"), amount, "", "");
        System.out.println(" " + count + " " + result);
        String txHash = (String)((Map)result.get("data")).get("value");
        m.put("txHash", txHash);
        m.put("index", 0);
        m.put("value", amount);
        Thread.sleep(10L);
    }
    addressList.addAll(list);
} catch (Exception e) {
    e.printStackTrace();
}
}
try {
    ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("./address.txt"));
    oos.writeObject(addressList);
    oos.flush();
    oos.close();
} catch (IOException e) {
    e.printStackTrace();
}
} catch (IOException e) {
    e.printStackTrace();
}
}

return addressList;
}
}

```

50:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-  
rpc\src\test\java\io\nuls\account\ledger\rpc\TransferTest.java  
\*/

```
package io.nuls.account.ledger.rpc;
```

```
import io.nuls.account.ledger.BaseTest;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.utils.AddressTool;
```

```

import java.util.ArrayList;
import java.util.List;

public class TransferTest extends BaseTest {
    private static List<String> list = new ArrayList<>();

    public static List<String> getAddressList() {
        if (list.isEmpty()) {
            //      for (int i = 0; i < 100; i++) {
            //          list.add(AddressTool.getStringAddressByBytes(AddressTool.getAddress(new
            ECKKey().getPubKey())));
            //      }
            list.add("NsdxMKjpm1Hp2FFmxXW1t9pwJ7JQXXxo");
        }
        return list;
    }

    private static int successCount = 0;

    public static void main(String[] args) {
        for (int i = 0; i < 1000000; i++) {
            doit();
        }
    }

    private static void doit() {
        //      List<String> addressList = getAddressList();
        //
        //      for (String toAddress : addressList) {
            String address = "NsdwFkL8FpQZ96ub6qfxNpLuKy72ar4b";
            String toAddress =
"Nsdw4KD2ERFax2j7vL2w8NZmtkxb9aiW";//01385ef69371c8fe003d2339158333e6b383eaf7a93
a38a77d022cf06024c82a
            long amount = 1001000L;
            String password = "";
            String remark = "test";

            String param = "{\"address\": \"" + address + "\", \"toAddress\": \"" + toAddress + "\",
\"password\": \"" + password + "\", \"amount\": \"" + amount + "\", \"remark\": \"" + remark + "\"}";

            String url = "http://192.168.1.27:8001/api/accountledger/transfer";

```

```

        for (int i = 0; i < 1; i++) {
            String res = post(url, param, "utf-8");
            if (res.indexOf("true") != -1) {
                successCount++;
            }
            System.out.println(successCount + " " + res);
            try {
                Thread.sleep(20L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

//
}

}

}

```

51:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-storage\src\main\java\io\nuls\account\ledger\storage\constant\AccountLedgerStorageConstant.java

a

\*/

```
package io.nuls.account.ledger.storage.constant;
```

```

/**
 * @desription:
 * @author: PierreLuo
 */
public interface AccountLedgerStorageConstant {
    /**
     *
     * The name of the account table
     */
    String DB_NAME_ACCOUNT_LEDGER_TX_INDEX = "account_ledger_tx_index";
    String DB_NAME_ACCOUNT_LEDGER_TX = "account_ledger_tx";
    String DB_NAME_ACCOUNT_LEDGER_COINDATA = "account_ledger_coindata";
}

```

```
52:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-  
storage\src\main\java\io\nuls\account\ledger\storage\po\TransactionInfoPo.java  
*/
```

```
package io.nuls.account.ledger.storage.po;
```

```
import io.nuls.account.ledger.model.TransactionInfo;  
import io.nuls.kernel.exception.NulsException;  
import io.nuls.kernel.model.Address;  
import io.nuls.kernel.model.BaseNulsData;  
import io.nuls.kernel.model.NulsDigestData;  
import io.nuls.kernel.model.Transaction;  
import io.nuls.kernel.utils.*;
```

```
import java.io.IOException;  
import java.util.List;
```

```
/**
```

```
 * @author Facjas
```

```
*/
```

```
public class TransactionInfoPo extends BaseNulsData {
```

```
    private NulsDigestData txHash;
```

```
    private long blockHeight;
```

```
    private long time;
```

```
    private byte[] addresses;
```

```
    private int txType;
```

```
    private byte status;
```

```
    public TransactionInfoPo() {
```

```
    }
```

```
    public TransactionInfoPo(Transaction tx) {
```

```
        if (tx == null) {
```

```
            return;
```

```
        }
```

```

this.txHash = tx.getHash();
this.blockHeight = tx.getBlockHeight();
this.time = tx.getTime();
List<byte[]> addressList = tx.getAllRelativeAddress();

byte[] addresses = new byte[addressList.size() * Address.ADDRESS_LENGTH];
for (int i = 0; i < addressList.size(); i++) {
    System.arraycopy(addressList.get(i), 0, addresses, Address.ADDRESS_LENGTH* i,
Address.ADDRESS_LENGTH);
}
this.addresses = addresses;
this.txType = tx.getType();
}

```

```

public TransactionInfoPo(TransactionInfo txInfo) {
    if (txInfo == null) {
        return;
    }
    //todo check weather need to clone the object
    this.txHash = txInfo.getTxHash();
    this.blockHeight = txInfo.getBlockHeight();
    this.time = txInfo.getTime();
    this.addresses = txInfo.getAddresses();
    this.txType = txInfo.getTxType();
    this.status = txInfo.getStatus();
}

```

```

public TransactionInfo toTransactionInfo() {
    //todo check weather need to clone the object
    TransactionInfo txInfo = new TransactionInfo();
    txInfo.setTxHash(this.txHash);
    txInfo.setBlockHeight(this.blockHeight);
    txInfo.setTime(this.time);
    txInfo.setAddresses(this.addresses);
    txInfo.setTxType(this.txType);
    txInfo.setStatus(this.status);
    return txInfo;
}

```

/\*\*

\* serialize important field

```
*/
```

```
@Override
```

```
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {  
    stream.writeNulsData(this.txHash);  
    stream.writeUInt32(blockHeight);  
    stream.writeUInt48(time);  
    stream.writeBytesWithLength(addresses);  
    stream.writeUInt16(txType);  
    stream.write(status);  
}
```

```
@Override
```

```
public void parse(NulsByteBuffer byteBuffer) throws NulsException {  
    this.txHash = byteBuffer.readHash();  
    this.blockHeight = byteBuffer.readUInt32();  
    this.time = byteBuffer.readUInt48();  
    this.addresses = byteBuffer.readByLengthByte();  
    this.txType = byteBuffer.readUInt16();  
    this.status = byteBuffer.readByte();  
}
```

```
@Override
```

```
public int size() {  
    int size = 0;  
    size += SerializeUtils.sizeOfNulsData(txHash);  
    size += SerializeUtils.sizeOfUInt32(); // blockHeight  
    size += SerializeUtils.sizeOfUInt48();  
    size += SerializeUtils.sizeOfBytes(addresses);  
    size += SerializeUtils.sizeOfUInt16(); // txType  
    size += 1;  
    return size;  
}
```

```
public NulsDigestData getTxHash() {  
    return txHash;  
}
```

```
public void setTxHash(NulsDigestData txHash) {  
    this.txHash = txHash;  
}
```

```
public byte[] getAddresses() {
```



```

        return addresses;
    }

    public void setAddresses(byte[] addresses) {
        this.addresses = addresses;
    }

    public byte getStatus() {
        return status;
    }

    public void setStatus(byte status) {
        this.status = status;
    }

    public long getBlockHeight() {
        return blockHeight;
    }

    public void setBlockHeight(long blockHeight) {
        this.blockHeight = blockHeight;
    }

    public long getTime() {
        return time;
    }

    public void setTime(long time) {
        this.time = time;
    }

    public int getTxType() {
        return txType;
    }

    public void setTxType(int txType) {
        this.txType = txType;
    }
}

```

53:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-storage\src\main\java\io\nuls\account\ledger\storage\po\UnconfirmedTxPo.java

```
*/
```

```
package io.nuls.account.ledger.storage.po;
```

```
import io.nuls.core.tools.crypto.Hex;
```

```
import io.nuls.core.tools.log.Log;
```

```
import io.nuls.kernel.exception.NulsException;
```

```
import io.nuls.kernel.model.BaseNulsData;
```

```
import io.nuls.kernel.model.Transaction;
```

```
import io.nuls.kernel.utils.NulsByteBuffer;
```

```
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
```

```
import io.nuls.kernel.utils.TransactionManager;
```

```
import java.io.IOException;
```

```
/**
```

```
 *
```

```
 * @author In
```

```
 */
```

```
public class UnconfirmedTxPo extends BaseNulsData {
```

```
    private Transaction tx;
```

```
    private long sequence;
```

```
    public UnconfirmedTxPo() {
```

```
    }
```

```
    public UnconfirmedTxPo(Transaction tx, long sequence) {
```

```
        this.tx = tx;
```

```
        this.sequence = sequence;
```

```
    }
```

```
    public UnconfirmedTxPo(byte[] txBytes) {
```

```
        super();
```

```
        try {
```

```
            parse(txBytes, 0);
```

```
        } catch (NulsException e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

```
    @Override
```

```

public int size() {
    return tx.size() + 8;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    tx.serializeToStream(stream);
    stream.writeInt64(sequence);
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    try {
        tx = TransactionManager.getInstance(byteBuffer);
    } catch (Exception e) {
        Log.info("Load local transaction Error");
    }
    sequence = byteBuffer.readInt64();
}

public Transaction getTx() {
    return tx;
}

public void setTx(Transaction tx) {
    this.tx = tx;
}

public long getSequence() {
    return sequence;
}

public void setSequence(long sequence) {
    this.sequence = sequence;
}
}

```

```

54:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
storage\src\main\java\io\nuls\account\ledger\storage\service\impl\LocalUtxoStorageServiceImpl.java
va
*/
package io.nuls.account.ledger.storage.service.impl;

```

```

import io.nuls.account.ledger.storage.constant.AccountLedgerStorageConstant;
import io.nuls.account.ledger.storage.service.LocalUtxoStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.model.Entry;
import io.nuls.db.service.BatchOperation;
import io.nuls.db.service.DBService;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Result;
import io.nuls.ledger.service.LedgerService;

import java.util.*;

import java.util.concurrent.ConcurrentHashMap;

/**
 * @author Facjas
 */
@Component
public class LocalUtxoStorageServiceImpl implements LocalUtxoStorageService, InitializingBean {
    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    @Autowired
    private LedgerService ledgerService;

    private Map<String, Entry<byte[], byte[]>> cacheMap;

    @Override
    public void afterPropertiesSet() throws NulsException {

        Result result =
dbService.createArea(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_COIND

```

```

ATA);
    if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
        throw new NulsRuntimeException(result.getErrorCode());
    }
}

@Override
public Collection<Entry<byte[], byte[]>> loadAllCoinList() {
    if(cacheMap == null) {
        cacheMap = new ConcurrentHashMap<>();

        List<Entry<byte[], byte[]>> coinList =
dbService.entryList(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_COINDAT
A);
        for(Entry<byte[], byte[]> entry : coinList) {
            cacheMap.put(new String(entry.getKey()), entry);
        }
    }
    return cacheMap.values();
}

@Override
public Result saveUTXO(byte[] key, byte[] value) {
    Result result =
dbService.put(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_COINDATA,
key, value);

    if(result.isSuccess() && cacheMap != null) {
        cacheMap.put(new String(key), new Entry(key, value));
    }

    return result;
}

@Override
public Result<Integer> batchSaveUTXO(Map<byte[], byte[]> utxos) {
    BatchOperation batch =
dbService.createWriteBatch(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_C
OINDATA);
    Set<Map.Entry<byte[], byte[]>> utxosToSaveEntries = utxos.entrySet();
    for(Map.Entry<byte[], byte[]> entry : utxosToSaveEntries) {
        batch.put(entry.getKey(), entry.getValue());
    }
}

```

```

    }
    Result batchResult = batch.executeBatch();
    if (batchResult.isFailed()) {
        return batchResult;
    }
    Result result = Result.getSuccess().setData(new Integer(utxos.size()));

    if(result.isSuccess() && cacheMap != null) {
        for(Map.Entry<byte[], byte[]> entry : utxosToSaveEntries) {
            cacheMap.put(new String(entry.getKey()), new Entry(entry.getKey(), entry.getValue()));
        }
    }

    return result;
}

@Override
public Result deleteUTXO(byte[] key) {
    Result result =
dbService.delete(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_COINDATA,
key);
    if(result.isSuccess() && cacheMap != null) {
        cacheMap.remove(new String(key));
    }
    return result;
}

@Override
public Result batchDeleteUTXO(Set<byte[]> utxos) {
    BatchOperation batch =
dbService.createWriteBatch(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_C
OINDATA);
    for (byte[] key : utxos) {
        batch.delete(key);
    }
    Result batchResult = batch.executeBatch();
    if (batchResult.isFailed()) {
        return batchResult;
    }
    Result result = Result.getSuccess().setData(new Integer(utxos.size()));

    if(result.isSuccess() && cacheMap != null) {

```

```

        for (byte[] key : utxos) {
            cacheMap.remove(new String(key));
        }
    }
    return result;
}

```

@Override

```

public Result batchSaveAndDeleteUTXO(List<Entry<byte[], byte[]>> utxosToSave, List<byte[]>
utxosToDelete) {
    BatchOperation batch =
dbService.createWriteBatch(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_C
OINDATA);
    for (byte[] key : utxosToDelete) {
        batch.delete(key);
    }
    for(Entry<byte[], byte[]> entry : utxosToSave) {
        batch.put(entry.getKey(), entry.getValue());
    }
    Result batchResult = batch.executeBatch();
    if (batchResult.isFailed()) {
        return batchResult;
    }
    Result result = Result.getSuccess().setData(new Integer(utxosToSave.size() +
utxosToDelete.size()));

```

```

    if(result.isSuccess() && cacheMap != null) {
        for(Entry<byte[], byte[]> entry : utxosToSave) {
            cacheMap.put(new String(entry.getKey()), entry);
        }
        for (byte[] key : utxosToDelete) {
            cacheMap.remove(new String(key));
        }
    }

```

```

    return result;
}

```

@Override

```

public byte[] getUtxoBytes(byte[] owner) {
    if (owner == null) {
        return null;
    }

```

```

    }
    return
dbService.get(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_COINDATA,
owner);
}

```

```

@Override
public Coin getUtxo(byte[] owner) {
    byte[] utxoBytes = getUtxoBytes(owner);
    Coin coin = null;
    try {
        if(utxoBytes != null) {
            coin = new Coin();
            coin.parse(utxoBytes, 0);
        }
    } catch (NulsException e) {
        Log.error(e);
        return null;
    }
    return coin;
}
}

```

55:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-storage\src\main\java\io\nuls\account\ledger\storage\service\impl\TransactionInfoStorageServiceImpl.java

\*/

```
package io.nuls.account.ledger.storage.service.impl;
```

```

import io.nuls.account.ledger.constant.AccountLedgerErrorCode;
import io.nuls.account.ledger.storage.constant.AccountLedgerStorageConstant;
import io.nuls.account.ledger.storage.po.TransactionInfoPo;
import io.nuls.account.ledger.storage.service.TransactionInfoStorageService;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;

```



```

import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.utils.AddressTool;

import javax.naming.PartialResultException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

/**
 * author Facjas
 * date 2018/5/22.
 */
@Component
public class TransactionInfoStorageServiceImpl implements TransactionInfoStorageService,
InitializingBean {
    @Autowired
    private DBService dbService;

    @Override
    public void afterPropertiesSet() throws NulsException {
        Result result =
dbService.createArea(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX_IND
EX);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveTransactionInfo(byte[] infoKey, TransactionInfoPo infoPo) {
        try {
dbService.put(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX_INDEX, info
Key, infoPo.serialize());
            return Result.getSuccess();
        } catch (Exception e) {
            return Result.getFailed();
        }
    }
}

```

```

@Override
public List<TransactionInfoPo> getTransactionInfoListByAddress(byte[] address) throws
NulsException {
    List<TransactionInfoPo> infoPoList = new ArrayList<>();
    Set<byte[]> keySet =
dbService.keySet(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX_INDEX)
;
    if (keySet == null || keySet.isEmpty()) {
        return infoPoList;
    }

    byte[] addressKey = new byte[Address.ADDRESS_LENGTH];
    for (byte[] key : keySet) {
        System.arraycopy(key, 0, addressKey, 0, Address.ADDRESS_LENGTH);
        if (java.util.Arrays.equals(addressKey, address)) {
            byte[] values =
dbService.get(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX_INDEX,
key);
            TransactionInfoPo transactionInfoPo = new TransactionInfoPo();
            transactionInfoPo.parse(values, 0);
            infoPoList.add(transactionInfoPo);
        }
    }
    return infoPoList;
}

```

```

@Override
public Result deleteTransactionInfo(byte[] infoKey) {
    return
dbService.delete(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX_INDEX,
infoKey);
}
}

```

56:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-storage\src\main\java\io\nuls\account\ledger\storage\service\impl\UnconfiredmTransactionStorageImpl.java

\*/

package io.nuls.account.ledger.storage.service.impl;

import io.nuls.account.ledger.storage.constant.AccountLedgerStorageConstant;  
import io.nuls.account.ledger.storage.po.UnconfirmedTxPo;

```

import io.nuls.account.ledger.storage.service.UnconfirmedTransactionStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.model.Entry;
import io.nuls.db.service.DBService;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;

```

```

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

```

```

/**

```

```

 * author Facjas

```

```

 * date 2018/5/22.

```

```

 */

```

```

@Component

```

```

public class UnconfiredmTransactionStorageImpl implements
UnconfirmedTransactionStorageService, InitializingBean {

```

```

    @Autowired

```

```

    private DBService dbService;

```

```

    private long sequence = System.currentTimeMillis();

```

```

    @Override

```

```

    public void afterPropertiesSet() throws NulsException {

```

```

        Result result =

```

```

        dbService.createArea(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX);

```

```

        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {

```

```

            throw new NulsRuntimeException(result.getErrorCode());

```

```

        }

```

```

    }

```

```

    @Override

```

```

public Result saveUnconfirmedTx(NulsDigestData hash, Transaction tx) {
    Result result;
    try {
        sequence++;
        UnconfirmedTxPo po = new UnconfirmedTxPo(tx, sequence);
        result =
dbService.put(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX,
hash.serialize(), po.serialize());
    } catch (Exception e) {
        e.printStackTrace();
        return Result.getFailed();
    }
    return result;
}

```

```

@Override
public Result deleteUnconfirmedTx(NulsDigestData hash) {
    try {
        return
dbService.delete(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX,
hash.serialize());
    } catch (Exception e) {
        Log.info("deleteUnconfirmedTx error");
        return Result.getFailed();
    }
}

```

```

@Override
public Result<Transaction> getUnconfirmedTx(NulsDigestData hash) {
    try {
        byte[] txBytes =
dbService.get(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX,
hash.serialize());
        if (txBytes == null) {
            return Result.getSuccess();
        }
        UnconfirmedTxPo po = new UnconfirmedTxPo(txBytes);
        Transaction tx = po.getTx();
        return Result.getSuccess().setData(tx);
    } catch (Exception e) {
        return Result.getFailed();
    }
}

```

```

    }

    @Override
    public Result<List<Transaction>> loadAllUnconfirmedList() {
        Result result;
        List<UnconfirmedTxPo> tmpList = new ArrayList<>();
        List<Entry<byte[], byte[]>> txs =
dbService.entryList(AccountLedgerStorageConstant.DB_NAME_ACCOUNT_LEDGER_TX);

        for (Entry<byte[], byte[]> txEntry : txs) {
            try {
                UnconfirmedTxPo tmpTx = new UnconfirmedTxPo(txEntry.getValue());
                if (tmpTx != null) {
                    NulsByteBuffer buffer = new NulsByteBuffer(txEntry.getKey(), 0);
                    tmpTx.getTx().setHash(buffer.readHash());
                    tmpList.add(tmpTx);
                }
            } catch (Exception e) {
                Log.warn("parse local tx error", e);
            }
        }

        tmpList.sort(new Comparator<UnconfirmedTxPo>() {
            @Override
            public int compare(UnconfirmedTxPo o1, UnconfirmedTxPo o2) {
                return (int) (o1.getSequence() - o2.getSequence());
            }
        });

        List<Transaction> resultList = new ArrayList<>();
        for (UnconfirmedTxPo po : tmpList) {
            resultList.add(po.getTx());
        }

        return Result.getSuccess().setData(resultList);
    }
}

```

```

57:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
storage\src\main\java\io\nuls\account\ledger\storage\service\LocalUtxoStorageService.java
*/
package io.nuls.account.ledger.storage.service;

```

```
import io.nuls.db.model.Entry;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Result;
```

```
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Set;
```

```
/**
 * @author Facjas
 */
public interface LocalUtxoStorageService {

    Result saveUTXO(byte[] key, byte[] value);

    Result batchSaveUTXO(Map<byte[], byte[]> utxos);

    Result deleteUTXO(byte[] key);

    Result batchDeleteUTXO(Set<byte[]> utxos);

    Collection<Entry<byte[], byte[]>> loadAllCoinList();

    Result batchSaveAndDeleteUTXO(List<Entry<byte[], byte[]>> utxosToSave, List<byte[]>
    utxosToDelete);

    byte[] getUtxoBytes(byte[] owner);

    Coin getUtxo(byte[] owner);
}
```

```
58:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-
storage\src\main\java\io\nuls\account\ledger\storage\service\TransactionInfoStorageService.java
*/
package io.nuls.account.ledger.storage.service;
```

```
import io.nuls.account.ledger.storage.po.TransactionInfoPo;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Result;
```

```
import java.util.List;
```

```
/**
```

```
 * author Facjas
```

```
 * date 2018/5/22.
```

```
 */
```

```
public interface TransactionInfoStorageService {
```

```
    Result saveTransactionInfo(byte[] key, TransactionInfoPo tx);
```

```
    Result deleteTransactionInfo(byte[] infoKey);
```

```
    List<TransactionInfoPo> getTransactionInfoListByAddress(byte[] address) throws  
NulsException;  
}
```

```
59:F:\git\coin\nuls\nuls-1.1.3\nuls\account-ledger-module\base\account-ledger-  
storage\src\main\java\io\nuls\account\ledger\storage\service\UnconfirmedTransactionStorageServi  
ce.java
```

```
 */
```

```
package io.nuls.account.ledger.storage.service;
```

```
import io.nuls.kernel.model.NulsDigestData;
```

```
import io.nuls.kernel.model.Result;
```

```
import io.nuls.kernel.model.Transaction;
```

```
import java.util.List;
```

```
/**
```

```
 * author Facjas
```

```
 * date 2018/5/22.
```

```
 */
```

```
public interface UnconfirmedTransactionStorageService {
```

```
    Result saveUnconfirmedTx(NulsDigestData hash, Transaction tx);
```

```
    Result deleteUnconfirmedTx(NulsDigestData hash);
```

```
    Result<Transaction> getUnconfirmedTx(NulsDigestData hash);
```

```
    Result<List<Transaction>> loadAllUnconfirmedList();
```

```
}
```

```
60:F:\git\coin\nuls\nuls-1.1.3\nuls\account-  
module\account\src\main\java\io\nuls\account\constant\AccountConstant.java  
*/
```

```
package io.nuls.account.constant;
```

```
import io.nuls.kernel.constant.NulsConstant;
```

```
import io.nuls.kernel.model.Na;
```

```
import java.util.Set;
```

```
import java.util.concurrent.ConcurrentHashMap;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
public interface AccountConstant extends NulsConstant {
```

```
    /**
```

```
     * The module id of the message-bus module
```

```
    */
```

```
    short MODULE_ID_ACCOUNT = 5;
```

```
    /**
```

```
     * The name of accouts cache
```

```
    */
```

```
    String ACCOUNT_LIST_CACHE = "ACCOUNT_LIST";
```

```
    /**
```

```
     * ()
```

```
     * The cost of setting an alias
```

```
    */
```

```
    Na ALIAS_NA = Na.parseNuls(1);
```

```
    /**
```

```
     *
```

```
     * Set the transaction type of account alias.
```

```
    */
```

```
    int TX_TYPE_ACCOUNT_ALIAS = 3;
```

```
    /**
```

```
     * (:)
```



```

    * Account unlock time maximum (unit: second)
    */
    int ACCOUNT_MAX_UNLOCK_TIME = 120;
/**
    * 20
    * If the change is more than 20, it can be changed.
    */
    int MIM_COUNT= 20;

/**
    * accountkeystore
    * The suffix of the accountkeystore file
    */
    String ACCOUNTKEYSTORE_FILE_SUFFIX=".keystore";
}

61:F:\git\coin\nuls\nuls-1.1.3\nuls\account-
module\account\src\main\java\io\nuls\account\constant\AccountErrorCode.java
*/

package io.nuls.account.constant;

import io.nuls.kernel.constant.ErrorCode;
import io.nuls.kernel.constant.KernelErrorCode;

/**
    * @author: Charlie
    */
public interface AccountErrorCode extends KernelErrorCode {

    ErrorCode PASSWORD_IS_WRONG = ErrorCode.init("50000");
    ErrorCode ACCOUNT_NOT_EXIST = ErrorCode.init("50001");
    ErrorCode ACCOUNT_IS_ALREADY_ENCRYPTED = ErrorCode.init("50002");
    ErrorCode ACCOUNT_EXIST = ErrorCode.init("50003");
    ErrorCode ADDRESS_ERROR = ErrorCode.init("50004");
    ErrorCode ALIAS_EXIST = ErrorCode.init("50005");
    ErrorCode ALIAS_NOT_EXIST = ErrorCode.init("50006");
    ErrorCode ACCOUNT_ALREADY_SET_ALIAS = ErrorCode.init("50007");
    ErrorCode ACCOUNT_UNENCRYPTED = ErrorCode.init("50008");
    ErrorCode ALIAS_CONFLICT = ErrorCode.init("50009");
    ErrorCode HAVE_ENCRYPTED_ACCOUNT = ErrorCode.init("50010");
    ErrorCode HAVE_UNENCRYPTED_ACCOUNT = ErrorCode.init("50011");

```

```

        ErrorCode PRIVATE_KEY_WRONG = ErrorCode.init("50012");
        ErrorCode ALIAS_ROLLBACK_ERROR = ErrorCode.init("50013");
        ErrorCode ACCOUNTKEYSTORE_FILE_NOT_EXIST = ErrorCode.init("50014");
        ErrorCode ACCOUNTKEYSTORE_FILE_DAMAGED = ErrorCode.init("50015");
        ErrorCode ALIAS_FORMAT_WRONG = ErrorCode.init("50016");
        ErrorCode PASSWORD_FORMAT_WRONG = ErrorCode.init("50017");
        ErrorCode DECRYPT_ACCOUNT_ERROR = ErrorCode.init("50018");
        ErrorCode ACCOUNT_IS_ALREADY_ENCRYPTED_AND_LOCKED = ErrorCode.init("50019");
        ErrorCode NICKNAME_TOO_LONG = ErrorCode.init("50020");
        ErrorCode INPUT_TOO_SMALL = ErrorCode.init("50021");
        ErrorCode MUST_BURN_A_NULS = ErrorCode.init("50022");
        ErrorCode SIGN_COUNT_TOO_LARGE = ErrorCode.init("50023");
    }

```

62:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\account\src\main\java\io\nuls\account\model\Account.java

```

package io.nuls.account.model;

```

```

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.core.tools.crypto.AESEncrypt;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.EncryptedData;
import io.nuls.core.tools.crypto.Exception.CryptoException;
import io.nuls.core.tools.crypto.Sha256Hash;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Result;
import org.spongycastle.crypto.params.KeyParameter;

```

```

import java.io.Serializable;
import java.math.BigInteger;
import java.util.Arrays;

```

```

/**
 * @author: Charlie
 */
public class Account implements Serializable {

```

```

/**

```

```
*
```

```
*/
```

```
private Address address;
```

```
/**
```

```
*
```

```
*/
```

```
private String alias;
```

```
/**
```

```
* is default acct
```

```
*/
```

```
private int status;
```

```
/**
```

```
*
```

```
*/
```

```
private byte[] pubKey;
```

```
/**
```

```
*
```

```
*/
```

```
private byte[] extend;
```

```
/**
```

```
*
```

```
*/
```

```
private Long createTime;
```

```
private byte[] encryptedPriKey;
```

```
/**
```

```
* Decrypted prikey
```

```
*/
```

```

private byte[] priKey;

/**
 * local field
 */

private ECKey ecKey;


private String remark;


/**
 * ()
 * Whether the account is encrypted (Whether the password is set)
 */
public boolean isEncrypted() {
    if (getEncryptedPriKey() != null && getEncryptedPriKey().length > 0) {
        return true;
    }
    return false;
}


/**
 *
 * Lock account
 */
public void lock() {
    if (!isEncrypted()) {
        return;
    }

    if (this.getEcKey().getEncryptedPrivateKey() != null) {
        ECKey result = ECKey.fromEncrypted(getEcKey().getEncryptedPrivateKey(),
getPubKey());
        this.setPriKey(new byte[0]);
        this.setEcKey(result);
    }
}


public byte[] getHash160() {
    return this.getAddress().getHash160();
}

```

```

}

/**
 *
 * Unlock account based on password
 */
public boolean unlock(String password) throws NulsException {
    decrypt(password);
    if (isLocked()) {
        return false;
    }
    return true;
}

/**
 * ()
 * Whether the account is locked (is there a cleartext private key)
 *
 * @return true: Locked, false: not Locked
 */
public boolean isLocked() {
    return (this.getPriKey() == null) || (this.getPriKey().length == 0);
}

/**
 *
 * Verify that the account password is correct
 */
public boolean validatePassword(String password) {
    boolean result = StringUtils.validPassword(password);
    if (!result) {
        return result;
    }
    byte[] unencryptedPrivateKey;
    try {
        unencryptedPrivateKey = AESEncrypt.decrypt(this.getEncryptedPriKey(), password);
    } catch (CryptoException e) {
        return false;
    }
    BigInteger newPriv = new BigInteger(1, unencryptedPrivateKey);
    ECKey key = ECKey.fromPrivate(newPriv);

```

```

        if (!Arrays.equals(key.getPubKey(), getPubKey())) {
            return false;
        }
        return true;
    }

    /**
     * ()
     * Password-encrypted account (set password for account)
     */
    public void encrypt(String password) throws NulsException {
        encrypt(password, false);
    }

    /**
     * ()
     * Password-encrypted account (set password for account)
     */
    public void encrypt(String password, boolean isForce) throws NulsException {
        if (this.isEncrypted()) {
            if (isForce) {
                if (isLocked()) {
                    throw new
NulsException(AccountErrorCode.ACCOUNT_IS_ALREADY_ENCRYPTED_AND_LOCKED);
                }
            } else {
                throw new NulsException(AccountErrorCode.ACCOUNT_IS_ALREADY_ENCRYPTED);
            }
        }
        ECKey eckey = this.getEcKey();
        byte[] privKeyBytes = eckey.getPrivKeyBytes();
        EncryptedData encryptedPrivateKey = AESEncrypt.encrypt(privKeyBytes,
EncryptedData.DEFAULT_IV, new KeyParameter(Sha256Hash.hash(password.getBytes())));
        eckey.setEncryptedPrivateKey(encryptedPrivateKey);
        ECKey result = ECKey.fromEncrypted(encryptedPrivateKey, getPubKey());
        this.setPriKey(new byte[0]);
        this.setEcKey(result);
        this.setEncryptedPriKey(encryptedPrivateKey.getEncryptedBytes());
    }

    /**
     * ,

```

```
* According to the decryption account, including generating the account plaintext private key
*/
```

```
private boolean decrypt(String password) throws NulsException {
    try {
        byte[] unencryptedPrivateKey = AESEncrypt.decrypt(this.getEncryptedPriKey(),
password);
        BigInteger newPriv = new BigInteger(1, unencryptedPrivateKey);
        ECKey key = ECKey.fromPrivate(newPriv);

        if (!Arrays.equals(key.getPubKey(), getPubKey())) {
            return false;
        }
        key.setEncryptedPrivateKey(new EncryptedData(this.getEncryptedPriKey()));
        this.setPriKey(key.getPrivKeyBytes());
        this.setEcKey(key);
    } catch (Exception e) {
        throw new NulsException(AccountErrorCode.PASSWORD_IS_WRONG);
    }
    return true;
}
```

```
public Object copy() {
    Account account = new Account();
    account.setAlias(alias);
    account.setAddress(address);
    account.setStatus(status);
    account.setPubKey(pubKey);
    account.setExtend(extend);
    account.setCreateTime(createTime);
    account.setEncryptedPriKey(encryptedPriKey);
    account.setPriKey(priKey);
    account.setEcKey(ecKey);
    account.setRemark(remark);
    return account;
}
```

```
public Address getAddress() {
    return address;
}
```

```
public void setAddress(Address address) {
```

```
    this.address = address;
}

public String getAlias() {
    return alias;
}

public void setAlias(String alias) {
    this.alias = alias;
}

public int getStatus() {
    return status;
}

public void setStatus(int status) {
    this.status = status;
}

public byte[] getPubKey() {
    return pubKey;
}

public void setPubKey(byte[] pubKey) {
    this.pubKey = pubKey;
}

public byte[] getExtend() {
    return extend;
}

public void setExtend(byte[] extend) {
    this.extend = extend;
}

public Long getCreateTime() {
    return createTime;
}

public void setCreateTime(Long createTime) {
    this.createTime = createTime;
}
```



```
public byte[] getEncryptedPriKey() {  
    return encryptedPriKey;  
}
```

```
public void setEncryptedPriKey(byte[] encryptedPriKey) {  
    this.encryptedPriKey = encryptedPriKey;  
}
```

```
public byte[] getPriKey() {  
    return priKey;  
}
```

```
public byte[] getPriKey(String password) throws NulsException {  
    if (!StringUtils.validPassword(password)) {  
        throw new NulsException(AccountErrorCode.PASSWORD_IS_WRONG);  
    }  
    byte[] unencryptedPrivateKey;  
    try {  
        unencryptedPrivateKey = AESEncrypt.decrypt(this.getEncryptedPriKey(), password);  
    } catch (CryptoException e) {  
        throw new NulsException(AccountErrorCode.PASSWORD_IS_WRONG);  
    }  
    BigInteger newPriv = new BigInteger(1, unencryptedPrivateKey);  
    ECKey key = ECKey.fromPrivate(newPriv);  
  
    if (!Arrays.equals(key.getPubKey(), getPubKey())) {  
        throw new NulsException(AccountErrorCode.PASSWORD_IS_WRONG);  
    }  
    return unencryptedPrivateKey;  
}
```

```
public void setPriKey(byte[] priKey) {  
    this.priKey = priKey;  
}
```

```
public ECKey getEcKey() {  
    return ecKey;  
}
```

```
public void setEcKey(ECKey ecKey) {  
    this.ecKey = ecKey;  
}
```

```

}

/**
 * ECKey
 */
public ECKey getEcKey(String password) throws NulsException {
    ECKey eckey = null;
    byte[] unencryptedPrivateKey;
    //
    BigInteger newPriv = null;
    if (this.isLocked()) {
        AssertUtil.canNotEmpty(password, "the password can not be empty");
        if (!validatePassword(password)) {
            throw new NulsException(AccountErrorCode.PASSWORD_IS_WRONG);
        }
        try {
            unencryptedPrivateKey = AESEncrypt.decrypt(this.getEncryptedPriKey(), password);
            newPriv = new BigInteger(1, unencryptedPrivateKey);
        } catch (CryptoException e) {
            throw new NulsException(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    } else {
        newPriv = new BigInteger(1, this.getPriKey());
    }
    eckey = ECKey.fromPrivate(newPriv);
    if (!Arrays.equals(eckey.getPubKey(), getPubKey())) {
        throw new NulsException(AccountErrorCode.PASSWORD_IS_WRONG);
    }
    return eckey;
}

public String getRemark() {
    return remark;
}

public void setRemark(String remark) {
    this.remark = remark;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {

```

```

        return false;
    }
    if (!(obj instanceof Account)) {
        return false;
    }
    Account other = (Account) obj;
    return Arrays.equals(pubKey, other.getPubKey());
}

```

```

@Override
public int hashCode() {
    return Arrays.hashCode(pubKey);
}
}

```

63:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\account\src\main\java\io\nuls\account\model\AccountKeyStore.java

```

package io.nuls.account.model;

```

```

/**

```

```

 * @author Facjas

```

```

 */

```

```

public class AccountKeyStore {

```

```

    private String address;
    private String encryptedPrivateKey;
    private byte[] prikey;
    private String alias;
    private byte[] pubKey;

```

```

    public AccountKeyStore() {
    }

```

```

    public AccountKeyStore(String address, String encryptedPrivateKey) {
        this.address = address;
        this.encryptedPrivateKey = encryptedPrivateKey;
    }

```

```

    public String getAddress() {
        return address;
    }

```

```

public void setAddress(String address) {
    this.address = address;
}

public String getEncryptedPrivateKey() {
    return encryptedPrivateKey;
}

public void setEncryptedPrivateKey(String encryptedPrivateKey) {
    this.encryptedPrivateKey = encryptedPrivateKey;
}

public String getAlias() {
    return alias;
}

public void setAlias(String alias) {
    this.alias = alias;
}

public byte[] getPubKey() {
    return pubKey;
}

public void setPubKey(byte[] pubKey) {
    this.pubKey = pubKey;
}

public byte[] getPrikey() {
    return prikey;
}

public void setPrikey(byte[] prikey) {
    this.prikey = prikey;
}
}

```

```

64:F:\git\coin\nuls\nuls-1.1.3\nuls\account-
module\account\src\main\java\io\nuls\account\model\Alias.java
*/

```

```
package io.nuls.account.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

/**
 * @author: Charlie
 */
public class Alias extends TransactionLogicData {

    private byte[] address;

    private String alias;

    public Alias() {
    }

    public Alias(byte[] address, String alias) {
        this.address = address;
        this.alias = alias;
    }

    public byte[] getAddress() {
        return address;
    }

    public void setAddress(byte[] address) {
        this.address = address;
    }

    public String getAlias() {
        return alias;
    }
}
```

```

public void setAlias(String alias) {
    this.alias = alias;
}

```

```

@Override
public Set<byte[]> getAddresses() {
    Set<byte[]> addressSet = new HashSet<>();
    addressSet.add(this.address);
    return addressSet;
}

```

```

@Override
public int size() {
    int s = 0;
    s += SerializeUtils.sizeOfBytes(address);
    s += SerializeUtils.sizeOfString(alias);
    return s;
}

```

```

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeBytesWithLength(address);
    stream.writeString(alias);
}

```

```

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.address = byteBuffer.readByLengthByte();
    this.alias = byteBuffer.readString();
}
}

```

```

65:F:\git\coin\nuls\nuls-1.1.3\nuls\account-
module\account\src\main\java\io\nuls\account\model\Balance.java
*/

```

```

package io.nuls.account.model;

```

```

import io.nuls.kernel.model.Na;

```

```
import java.io.Serializable;

/**
 * @author: Charlie
 */
public class Balance implements Serializable {

    private Na balance;

    private Na locked;

    private Na usable;

    public Balance() {
        this.balance = Na.ZERO;
        this.locked = Na.ZERO;
        this.usable = Na.ZERO;
    }

    public Balance(Na usable, Na locked) {
        this.usable = usable;
        this.locked = locked;
        this.balance = locked.add(usable);
    }

    public Na getBalance() {
        return balance;
    }

    public void setBalance(Na balance) {
        this.balance = balance;
    }

    public Na getLocked() {
        return locked;
    }

    public void setLocked(Na locked) {
        this.locked = locked;
    }
}
```

```

    public Na getUsable() {
        return usable;
    }

    public void setUsable(Na usable) {
        this.usable = usable;
    }
}

66:F:\git\coin\nuls\nuls-1.1.3\nuls\account-
module\account\src\main\java\io\nuls\account\model\MultiSigAccount.java

package io.nuls.account.model;

import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * @author: Niels Wang
 * @date: 2018/9/16
 */
public class MultiSigAccount extends BaseNulsData {

    private Address address;

    private List<byte[]> pubKeyList;

    private long m;
    private String alias;

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.write(address.getAddressBytes());
    }

```



```

        stream.writeUint32(m);
        stream.writeUint32(pubKeyList.size());
        for (int i = 0; i < pubKeyList.size(); i++) {
            stream.writeBytesWithLength(pubKeyList.get(i));
        }
    }
}

```

@Override

```

public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    byte[] bytes = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.address = Address.fromHashs(bytes);
    this.m = byteBuffer.readUint32();
    this.pubKeyList = new ArrayList<>();
    long count = byteBuffer.readUint32();
    for (int i = 0; i < count; i++) {
        pubKeyList.add(byteBuffer.readByLengthByte());
    }
}

```

@Override

```

public int size() {
    int size = Address.ADDRESS_LENGTH;
    size += SerializeUtils.sizeOfUint32();
    size += SerializeUtils.sizeOfUint32();
    for (int i = 0; i < pubKeyList.size(); i++) {
        size += SerializeUtils.sizeOfBytes(pubKeyList.get(i));
    }
    return size;
}

```

```

public Address getAddress() {
    return address;
}

```

```

public void setAddress(Address address) {
    this.address = address;
}

```

```

public List<byte[]> getPubKeyList() {
    return pubKeyList;
}

```

```

public void setPubKeyList(List<byte[]> pubKeyList) {
    this.pubKeyList = pubKeyList;
}

public long getM() {
    return m;
}

public void setM(long m) {
    this.m = m;
}

public void addPubkeys(List<String> pubkeys) {
    this.pubKeyList = new ArrayList<>();
    for (String pubkeyStr : pubkeys) {
        pubKeyList.add(Hex.decode(pubkeyStr));
    }
}

public void setAlias(String alias) {
    this.alias = alias;
}

public String getAlias() {
    return alias;
}
}

67:F:\git\coin\nuls\nuls-1.1.3\nuls\account-
module\account\src\main\java\io\nuls\account\module\AbstractAccountModuleBootstrap.java
*/

package io.nuls.account.module;

import io.nuls.account.constant.AccountConstant;
import io.nuls.kernel.module.BaseModuleBootstrap;

/**
 * @author: Niels Wang
 */
public abstract class AbstractAccountModuleBootstrap extends BaseModuleBootstrap {
    public AbstractAccountModuleBootstrap() {

```

```
        super(AccountConstant.MODULE_ID_ACCOUNT);
    }
}
```

68:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\account\src\main\java\io\nuls\account\service\AccountService.java

```
package io.nuls.account.service;
```

```
import io.nuls.account.model.Account;
import io.nuls.account.model.AccountKeyStore;
import io.nuls.account.model.Balance;
import io.nuls.account.model.MultiSigAccount;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsSignData;
import io.nuls.kernel.model.Result;
```

```
import java.util.Collection;
import java.util.List;
```

```
/**
 *
 * account service definition
 *
 * @author: Niels Wang
 */
public interface AccountService {
```

```
/**
 *
 * Create a specified number of accounts,and encrypt the accounts,
 * all the accounts are encrypted by the same password
 * if the password is NULL or "", the accounts will be unencrypted.
 *
 * @param count
 * @param count the number of account you want to create.
 * @param password the password of the accounts.
 * @return the account list created.
 */
```

```
Result<List<Account>> createAccount(int count, String password);
```

```
/**
```

```
*
```

```
* Create unencrypted accounts.
```

```
*
```

```
* @param count
```

```
* @param count the number of account you want to create.
```

```
* @return the account list created.
```

```
*/
```

```
Result<List<Account>> createAccount(int count);
```

```
/**
```

```
*
```

```
* Create an account and encrypt it,
```

```
* if the password is NULL or "", the accounts will be unencrypted.
```

```
*
```

```
* @param password the password of the accounts(only one account in the list).
```

```
* @return the account list created.
```

```
*/
```

```
Result<List<Account>> createAccount(String password);
```

```
/**
```

```
*
```

```
* Create an unencrypted account
```

```
*
```

```
* @return the account list created(only one account in the list).
```

```
*/
```

```
Result<List<Account>> createAccount();
```

```
/**
```

```
*
```

```
* delete an account by address.
```

```
*
```

```
* @param address the address of the account you want to delete.
```

```
* @param password the password of the account.
```

```
* @return the result of the operation.
```

```
*/
```

```
Result removeAccount(String address, String password);
```

```
/**
```

```

* keyStore
* Reset password by keyStore.
*
* @param keyStore the keyStore of the account.
* @param password the password of account
* @return the result of the operation.
*/

```

```

Result<Account> updatePasswordByAccountKeyStore(AccountKeyStore keyStore, String
password);

```

```

/**
* keyStore(keystore)
* 1.keyStore(,keyStoreencryptedPrivateKey)
* 2.keyStore,
* 3.,(keyStore,)
* 4.
* 5.
* import an account form account key store.
*
* @param keyStore the keyStore of the account.
* @param password the password of account
* @return the result of the operation.
*/

```

```

Result<Account> importAccountFormKeyStore(AccountKeyStore keyStore, String password);

```

```

/**
* keyStore
* 1.keyStore
* 2.keyStore,
* 3.,(keyStore,)
* 4.
* 5.
* import an account form account key store.
*
* @param keyStore the keyStore of the account.
* @return the result of the operation.
*/

```

```

Result<Account> importAccountFormKeyStore(AccountKeyStore keyStore);

```

```

/**
*
* import an account from plant private key and encrypt the account.

```

```
*/  
Result<Account> importAccount(String prikey, String password);
```

```
/**  
 *  
 * import an unencrypted account by plant private key.  
 */
```

```
Result<Account> importAccount(String prikey);
```

```
/**  
 * keyStore  
 * export an account to an account key store.  
 *  
 * @param address the address of the account.  
 * @param password the password of the account key store.  
 * @return the account key store object.  
 */
```

```
Result<AccountKeyStore> exportAccountToKeyStore(String address, String password);
```

```
/**  
 * byte[]  
 * Query account information by address.  
 *  
 * @param address the address of the account you want to query.  
 * @return the account.  
 */
```

```
Result<Account> getAccount(byte[] address);
```

```
/**  
 *  
 * Query account by address.  
 *  
 * @param address the address of the account you want to query.  
 * @return the account.  
 */
```

```
Result<Account> getAccount(String address);
```

```
/**  
 *  
 * Query account by account address.  
 *  
 * @param address the address of the account you want to query;
```

\* @return the account.

\*/

Result<Account> getAccount(Address address);

/\*\*

\*

\* Query account address by public key.

\*

\* @param pubKey public key string.

\* @return the account address.

\*/

Result<Address> getAddress(String pubKey);

/\*\*

\*

\* Gets the account address object from the account binary public key.

\*

\* @param pubKey public key binary array.

\* @return the account address.

\*/

Result<Address> getAddress(byte[] pubKey);

/\*\*

\*

\* Verify whether the account is encrypted.

\*

\* @param address The address of the account to be verified.

\* @return the result of the operation.

\*/

Result isEncrypted(String address);

// /\*\*

// \* Verify the account password.

// \* @param account account

// \* @param password password

// \* @return Result

// \*/

// Result validPassword(Account account, String password);

// /\*\*

// \*

// \* Verify the format of the address string.

```

//  *
//  * @param address To verify the address string.
//  * @return the result of the operation.
//  */
//  Result verifyAddressFormat(String address);

/**
 *
 * Query all account collections.
 *
 * @return account list of all accounts.
 */
Result<Collection<Account>> getAccountList();

// /**
//  *
//  * Sign data.
//  *
//  * @param data    Data to be signed.
//  * @param account Signed account
//  * @param password Account password
//  * @return The NulsSignData object.
//  * @throws NulsException nulsException
//  */
//  NulsSignData signData(byte[] data, Account account, String password) throws NulsException;
//
// /**
//  * ()
//  * Sign data.(no password)
//  *
//  * @param data    Data to be signed.
//  * @param account Signed account
//  * @return The NulsSignData object.
//  * @throws NulsException nulsException
//  */
//  NulsSignData signData(byte[] data, Account account) throws NulsException;
//
// /**
//  *
//  * Sign data.
//  *
//  * @param data    Data to be signed.

```



```

//  * @param ecKey eckey.
//  * @return The NulsSignData object.
//  * @throws NulsException nulsException
//  */
//  NulsSignData signData(byte[] data, ECKKey ecKey) throws NulsException;

/**
 *
 * Sign data.
 *
 * @param digest  data digest.
 * @param account  account to sign.
 * @param password password of account.
 * @return the NulsSignData object.
 * @throws NulsException nulsException
 */
NulsSignData signDigest(byte[] digest, Account account, String password) throws
NulsException;

/**
 *
 * Sign data digest
 *
 * @param digest to be signed.
 * @param ecKey  eckey
 * @return The NulsSignData object.
 */
NulsSignData signDigest(byte[] digest, ECKKey ecKey);

// /**
//  *
//  * Verify the signature.
//  *
//  * @param data  data to be validated.
//  * @param signData signature.
//  * @param pubKey  ublic key of account.
//  * @return the result of the opration
//  */
//  Result verifySignData(byte[] data, NulsSignData signData, byte[] pubKey);

/**
 *

```

```

* Query the balance of all accounts.
*
* @return Balance object.
* @throws NulsException nulsException
*/
Result<Balance> getBalance() throws NulsException;

//
// /**
//  *
//  * Query the balance of an account.
//  *
//  * @param account the account.
//  * @return Balance object.
//  * @throws NulsException nulsException
//  */
// Result<Balance> getBalance(Account account) throws NulsException;

// /**
//  *
//  * Query the balance of an account.
//  *
//  * @param address the address of the account.
//  * @return Balance object.
//  * @throws NulsException nulsException
//  */
// Result<Balance> getBalance(Address address) throws NulsException;

//
// /**
//  *
//  * Query the balance of an account.
//  *
//  * @param address the address of the account.
//  * @return Balance object.
//  * @throws NulsException nulsException
//  */
// Result<Balance> getBalance(String address) throws NulsException;

/**
 *
 * Get an account alias based on the array of account address bytes
 *

```

```
* @param address the address of the account.  
* @return alias string  
*/
```

```
Result<String> getAlias(byte[] address);
```

```
/**  
 *  
 * Get account alias according to account address  
 *  
 * @param address the address of the account.  
 * @return alias string  
 */
```

```
Result<String> getAlias(String address);
```

```
/**  
 *  
 * Create a specified number of accounts, and encrypt the accounts,  
 * all the accounts are encrypted by the same password  
 * if the password is NULL or "", the accounts will be unencrypted.  
 *  
 * @param pubkeys  
 * @param m  
 * @return the account list created.  
 */
```

```
Result<Address> createMultiAccount(List<String> pubkeys, int m);
```

```
/**  
 *  
 * Query all account collections.  
 *  
 * @return account list of all accounts.  
 */
```

```
Result<List<MultiSigAccount>> getMultiSigAccountList();
```

```
/**  
 *  
 * Get the details of the locally stored multi-sign account based on the address  
 *  
 * @param address  
 * @return  
 */
```

```
Result<MultiSigAccount> getMultiSigAccount(String address) throws Exception;
```

```

/**
 *
 * @param address
 * @param pubkeys
 * @param m
 * @return
 */
Result<Boolean> saveMultiSigAccount(String address, List<String> pubkeys, int m);

/**
 *
 * @param address
 * @return
 */
Result<Boolean> removeMultiSigAccount(String address);
}

```

69:F:\git\coin\nuls\nuls-1.1.3\nuls\account-  
module\account\src\main\java\io\nuls\account\service\AccountTxService.java

```

*/
package io.nuls.account.service;

import io.nuls.account.model.Account;
import io.nuls.kernel.model.*;

import java.util.List;

/**
 * @author: Niels Wang
 */
public interface AccountTxService {

    Result<Transaction> saveAccountTx(Transaction tx);

// /**
//  *
//  * @param tx
//  * @return
//  */
    Result<Transaction> checkAndSaveAccountTx(Transaction tx);

```

```

Result removeAccountTx(NulsDigestData txHash);

Result<Transaction> updateAccountTx(Transaction tx);

Result<List<Transaction>> getUnconfirmAccountTxList(Account account);
Result<List<Transaction>> getUnconfirmAccountTxList();

Result<List<Coin>> getUseableCoinList(Account account,Na na);
}

70:F:\git\coin\nuls\nuls-1.1.3\nuls\account-
module\account\src\main\java\io\nuls\account\tx\AliasTransaction.java
*/

package io.nuls.account.tx;

import io.nuls.account.constant.AccountConstant;
import io.nuls.account.model.Alias;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;

/**
 * @author: Charlie
 */
public class AliasTransaction extends Transaction<Alias> {

    public AliasTransaction() {
        super(AccountConstant.TX_TYPE_ACCOUNT_ALIAS);
    }

    protected AliasTransaction(int type) {
        super(type);
    }

    @Override
    public String getInfo(byte[] address) {
        return "-" + AccountConstant.ALIAS_NA.add(getCoinData().getFee()).toCoinString();
    }

```

```

    }

    @Override
    protected Alias parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new Alias());
    }
}

```

71:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\account\src\main\java\io\nuls\account\util\AccountTool.java  
 \*/

```

package io.nuls.account.util;

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.model.Account;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.kernel.model.Address;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.crypto.Sha256Hash;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.utils.SerializeUtils;

import java.math.BigInteger;

/**
 * @author: Charlie
 */
public class AccountTool {

    public static final int CREATE_MAX_SIZE = 100;

    public static Address newAddress(ECKey key) throws NulsException {
        return newAddress(key.getPubKey());
    }

    public static Address newAddress(byte[] publicKey) throws NulsException {

```

```

        return new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(publicKey));
    }

```

```

public static Account createAccount(String prikey) throws NulsException {
    ECKey key = null;
    if (StringUtils.isBlank(prikey)) {
        key = new ECKey();
    } else {
        try {
            key = ECKey.fromPrivate(new BigInteger(1, Hex.decode(prikey)));
        } catch (Exception e) {
            throw new NulsException(AccountErrorCode.PRIVATE_KEY_WRONG, e);
        }
    }
    Address address = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(key.getPubKey()));
    Account account = new Account();
    account.setEncryptedPriKey(new byte[0]);
    account.setAddress(address);
    account.setPubKey(key.getPubKey());
    account.setEcKey(key);
    account.setPriKey(key.getPrivKeyBytes());
    account.setCreateTime(TimeService.currentTimeMillis());
    return account;
}

```

```

public static Account createAccount() throws NulsException {
    return createAccount(null);
}

```

```

public static Address createContractAddress() throws NulsException {
    ECKey key = new ECKey();
    return new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.CONTRACT_ADDRESS_TYPE, SerializeUtils.sha256hash160(key.getPubKey()));
}

```

```

// /**

```

```

//  * Generate the corresponding account management private key or transaction private key
//  according to the seed private key and password

```

```

// */

```

```

public static BigInteger genPrivKey(byte[] encryptedPriKey, byte[] pw) {

```

```

byte[] privSeedSha256 = Sha256Hash.hash(encryptedPriKey);
//get sha256 of encryptedPriKey and sha256 of pw
byte[] pwSha256 = Sha256Hash.hash(pw);
//privSeedSha256 + pwPwSha256
byte[] pwPriBytes = new byte[privSeedSha256.length + pwSha256.length];
for (int i = 0; i < pwPriBytes.length; i += 2) {
    int index = i / 2;
    pwPriBytes[index] = privSeedSha256[index];
    pwPriBytes[index + 1] = pwSha256[index];
}
//get prikey
return new BigInteger(1, Sha256Hash.hash(pwPriBytes));
}

}

```

72:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\main\java\io\nuls\account\module\AccountModuleBootstrap.java

\*/

```
package io.nuls.account.module;
```

```
import io.nuls.account.service.AccountService;
import io.nuls.kernel.context.NulsContext;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class AccountModuleBootstrap extends AbstractAccountModuleBootstrap {
```

```
 /**
```

```
 *
```

```
 */
```

```
 @Override
```

```
 public void init() throws Exception {
```

```
 }
```

```
 /**
```

```
 * start the module
```

```
 */
```

```
 @Override
```

```
 public void start() {
```



```

        AccountService accountService = NulsContext.getServiceBean(AccountService.class);
        accountService.getAccountList();
    }

    /**
     * stop the module
     */
    @Override
    public void shutdown() {

    }

    /**
     * destroy the module
     */
    @Override
    public void destroy() {

    }

    /**
     * get all info of the module
     */
    @Override
    public String getInfo() {
        return null;
    }
}

```

73:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\main\java\io\nuls\account\process\AliasTxProcessor.java

```

package io.nuls.account.process;

import io.nuls.account.constant.AccountConstant;
import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.model.Alias;
import io.nuls.account.service.AliasService;
import io.nuls.account.storage.po.AliasPo;
import io.nuls.account.tx.AliasTransaction;
import io.nuls.core.tools.crypto.Hex;

```

```

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.processor.TransactionProcessor;
import io.nuls.kernel.validate.ValidateResult;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 *
 *
 * @author: Charlie
 */
@Component
public class AliasTxProcessor implements TransactionProcessor<AliasTransaction> {

    @Autowired
    private AliasService aliasService;

    @Override
    public Result onRollback(AliasTransaction tx, Object secondaryData) {
        Alias alias = tx.getTxData();
        try {
            return aliasService.rollbackAlias(new AliasPo(alias));
        } catch (NulsException e) {
            Log.error(e);
            return Result.getFailed(AccountErrorCode.ALIAS_ROLLBACK_ERROR);
        }
    }

    @Override
    public Result onCommit(AliasTransaction tx, Object secondaryData) {
        Alias alias = tx.getTxData();
        try {
            return aliasService.saveAlias(new AliasPo(alias));
        } catch (NulsException e) {
            Log.error(e);

```

```

        return Result.getFailed(AccountErrorCode.FAILED);
    }
}

/**
 *
 * 1.account
 * 2.
 * conflictDetect
 * 1.Detecting an account can only set one alias.
 * 2.Check if multiple aliasTransaction have the same alias.
 *
 * @param txList /A list of transactions to be checked.
 */
@Override
public ValidateResult conflictDetect(List<Transaction> txList) {
    if (null == txList || txList.isEmpty()) {
        return ValidateResult.getSuccessResult();
    }
    Set<String> aliasNames = new HashSet<>();
    Set<String> accountAddress = new HashSet<>();
    for (Transaction transaction : txList) {
        if (transaction.getType() == AccountConstant.TX_TYPE_ACCOUNT_ALIAS){
            AliasTransaction aliasTransaction = (AliasTransaction) transaction;
            Alias alias = aliasTransaction.getTxData();
            if (!aliasNames.add(alias.getAlias())) {
                return (ValidateResult) ValidateResult.getFailedResult(getClass().getName(),
AccountErrorCode.ALIAS_CONFLICT).setData(aliasTransaction);
            }
            if (!accountAddress.add(Hex.encode(alias.getAddress())) {
                return (ValidateResult) ValidateResult.getFailedResult(getClass().getName(),
AccountErrorCode.ACCOUNT_ALREADY_SET_ALIAS).setData(aliasTransaction);
            }
            break;
        }
    }
    return ValidateResult.getSuccessResult();
}
}

```

base\src\main\java\io\nuls\account\service\AccountBaseService.java

\*/

package io.nuls.account.service;

import io.nuls.account.constant.AccountErrorCode;

import io.nuls.account.model.Account;

import io.nuls.account.storage.po.AccountPo;

import io.nuls.account.storage.service.AccountStorageService;

import io.nuls.core.tools.crypto.Hex;

import io.nuls.core.tools.log.Log;

import io.nuls.core.tools.str.StringUtils;

import io.nuls.kernel.exception.NulsException;

import io.nuls.kernel.lite.annotation.Autowired;

import io.nuls.kernel.lite.annotation.Service;

import io.nuls.kernel.model.Result;

import io.nuls.kernel.utils.AddressTool;

import java.util.\*;

/\*\*

\* @author: Charlie

\*/

@Service

public class AccountBaseService {

@Autowired

private AccountService accountService;

@Autowired

private AccountStorageService accountStorageService;

private AccountCacheService accountCacheService = AccountCacheService.getInstance();

public Result setRemark(String address, String remark){

if (!AddressTool.validAddress(address)) {

return Result.getFailed(AccountErrorCode.ADDRESS\_ERROR);

}

Account account = accountService.getAccount(address).getData();

if (null == account) {

return Result.getFailed(AccountErrorCode.ACCOUNT\_NOT\_EXIST);

```

    }
    if (StringUtils.isBlank(remark)) {
        remark = null;
    }
    if (!StringUtils.validRemark(remark)) {
        return Result.getFailed(AccountErrorCode.NICKNAME_TOO_LONG);
    }
    account.setRemark(remark);
    Result result = accountStorageService.updateAccount(new AccountPo(account));
    if (result.isFailed()) {
        return Result.getFailed(AccountErrorCode.FAILED);
    }
    accountCacheService.localAccountMaps.put(account.getAddress().getBase58(), account);
    return Result.getSuccess().setData(true);
}

```

/\*\*

\*

\* Get the account private key

\*

\* @param address

\* @param password

\* @return

\*/

```

public Result getPrivateKey(String address, String password) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    Account account = accountService.getAccount(address).getData();
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    //((), Already encrypted(Added password) and did not unlock, verify password
    if (account.isEncrypted() && account.isLocked()) {
        try {
            byte[] priKeyBytes = account.getPriKey(password);
            return Result.getSuccess().setData(Hex.encode(priKeyBytes));
        } catch (NulsException e) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    } else {
        return Result.getSuccess().setData(Hex.encode(account.getPriKey()));
    }
}

```

```

    }
}

/**
 *
 *
 * Get the all local private keys
 *
 * @param password
 * @return
 */
public Result getAllPrivateKey(String password) {
    Collection<Account> localAccountList = accountService.getAccountList().getData();
    if (localAccountList == null || localAccountList.isEmpty()) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
        return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
    }
    List<String> list = new ArrayList<>();
    for (Account account : localAccountList) {
        if (account.isEncrypted()){
            if(StringUtils.isBlank(password)){
                //,
                return Result.getFailed(AccountErrorCode.HAVE_ENCRYPTED_ACCOUNT);
            }
            try {
                byte[] priKeyBytes = account.getPriKey(password);
                list.add(Hex.encode(priKeyBytes));
            } catch (NulsException e) {
                return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
            }
        }else {
            if (StringUtils.isNotBlank(password)) {
                //
                return Result.getFailed(AccountErrorCode.HAVE_UNENCRYPTED_ACCOUNT);
            }
            list.add(Hex.encode(account.getPriKey()));
        }
    }
    Map<String, List<String>> map = new HashMap<>();
    map.put("value", list);
}

```

```

        return Result.getSuccess().setData(map);
    }

    public Result getAllPrivateKey() {
        return getAllPrivateKey(null);
    }

    /**
     *
     * Set password (Encryption account)
     *
     * @param address
     * @param password
     * @return
     */
    public Result setPassword(String address, String password) {
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
        }
        if (StringUtils.isBlank(password)) {
            return Result.getFailed(AccountErrorCode.NULL_PARAMETER);
        }
        if (!StringUtils.validPassword(password)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG);
        }

        Account account = accountService.getAccount(address).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
        }
        if (account.isEncrypted()) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_IS_ALREADY_ENCRYPTED);
        }
        try {
            account.encrypt(password);
            Result result = accountStorageService.updateAccount(new AccountPo(account));
            if (result.isFailed()) {
                return Result.getFailed(AccountErrorCode.FAILED);
            }
            accountCacheService.localAccountMaps.put(account.getAddress().getBase58(), account);
        } catch (NulsException e) {

```

```

        Log.error(e);
        return Result.getFailed(AccountErrorCode.FAILED);
    }
    return Result.getSuccess().setData(true);
}

/**
 *
 * Change the account password according to the current password
 *
 * @param oldPassword
 * @param newPassword
 * @return
 */
public Result changePassword(String address, String oldPassword, String newPassword) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    if (StringUtils.isBlank(oldPassword)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
    if (StringUtils.isBlank(newPassword)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
    if (!StringUtils.validPassword(oldPassword)) {
        return Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG);
    }
    if (!StringUtils.validPassword(newPassword)) {
        return Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG);
    }
    Account account = accountService.getAccount(address).getData();
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    try {
        if (!account.isEncrypted()) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_UNENCRYPTED);
        }
        if (!account.validatePassword(oldPassword)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
        account.unlock(oldPassword);
    }

```



```

        account.encrypt(newPassword, true);
        AccountPo po = new AccountPo(account);
        Result result = accountStorageService.updateAccount(po);
        if (result.isFailed()) {
            return Result.getFailed(AccountErrorCode.FAILED);
        }
        accountCacheService.localAccountMaps.put(account.getAddress().getBase58(), account);
        return result.setData(true);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode());
    }
}
}
}

```

75:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\main\java\io\nuls\account\service\AccountCacheService.java  
\*/

```
package io.nuls.account.service;
```

```

import io.nuls.account.constant.AccountConstant;
import io.nuls.account.model.Account;
import io.nuls.kernel.model.Address;
import io.nuls.cache.CacheMap;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.kernel.utils.AddressTool;

```

```

import java.util.List;
import java.util.Map;

```

```

/**
 *
 * Account Cache Service
 *
 * @author: Charlie
 */

```

```
public class AccountCacheService {
```

```
    private static final AccountCacheService INSTANCE = new AccountCacheService();
```

```

private CacheMap<String, Account> cacheMap;

/**
 *
 * Collection of local accounts
 */
public Map<String, Account> localAccountMaps;

private AccountCacheService() {
    this.cacheMap = new CacheMap<>(AccountConstant.ACCOUNT_LIST_CACHE, 32,
String.class, Account.class);
}

public static AccountCacheService getInstance() {
    return INSTANCE;
}

/**
 *
 * Cache an account
 *
 * @param account Account to be cached
 */
public void putAccount(Account account) {
    this.cacheMap.put(account.getAddress().getBase58(), account);
}

/**
 *
 * Get accounts based on account address
 *
 * @param address Account to be operated
 */
public Account getAccountByAddress(String address) {
    List<Account> list = this.getAccountList();
    for (Account account : list) {
        if (account.getAddress().toString().equalsIgnoreCase(address)) {
            return account;
        }
    }
    return null;
}

```

```

}

/**
 *
 * Verify the existence of the account
 */
public boolean contains(byte[] address) {
    return this.cacheMap.containsKey(AddressTool.getStringAddressByBytes(address));
}

/**
 *
 * Get all accounts
 *
 * @return List<Account>
 */
public List<Account> getAccountList() {
    return this.cacheMap.values();
}

public void removeAccount(Address address) {
    this.cacheMap.remove(address.getBase58());
}

public void removeAccount(byte[] address) {
    this.cacheMap.remove(AddressTool.getStringAddressByBytes(address));
}

public void clear() {
    if (null == cacheMap) {
        return;
    }
    this.cacheMap.clear();
}

public void destroy() {
    this.cacheMap.destroy();
}

/**
 *

```

```

    * Cache multiple accounts
    */
    public void putAccountList(List<Account> list) {
        if (null != list) {
            for (Account account : list) {
                this.putAccount(account);
            }
        }
    }
}

```

76:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\main\java\io\nuls\account\service\AliasService.java

```

    */

```

```

package io.nuls.account.service;

```

```

import io.nuls.account.constant.AccountConstant;
import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.model.CoinDataResult;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.Account;
import io.nuls.account.model.Alias;
import io.nuls.account.model.MultiSigAccount;
import io.nuls.account.storage.po.AccountPo;
import io.nuls.account.storage.po.AliasPo;
import io.nuls.account.storage.service.AccountStorageService;
import io.nuls.account.storage.service.AliasStorageService;
import io.nuls.account.tx.AliasTransaction;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.*;

```

```
import io.nuls.kernel.script.*;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.TransactionFeeCalculator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.protocol.model.tx.TransferTransaction;
import io.nuls.protocol.service.TransactionService;
```

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
```

```
/**
 *
 * Account module internal function service class
 *
 * @author: Charlie
 */
```

```
@Service
```

```
public class AliasService {
```

```
    @Autowired
```

```
    private AccountService accountService;
```

```
    @Autowired
```

```
    private AccountStorageService accountStorageService;
```

```
    @Autowired
```

```
    private AccountLedgerService accountLedgerService;
```

```
    @Autowired
```

```
    private AliasStorageService aliasStorageService;
```

```
    @Autowired
```

```
    private MessageBusService messageBusService;
```

```
    @Autowired
```

```
    private TransactionService transactionService;
```

@Autowired

private LedgerService ledgerService;

private AccountCacheService accountCacheService = AccountCacheService.getInstance();

/\*\*

\*

\* Initiate a transaction to set alias.

\*

\* @param addr Address of account

\* @param password password of account

\* @param aliasName the alias to set

\* @return txhash

\*/

```
public Result<String> setAlias(String addr, String aliasName, String password) {
    if (!AddressTool.validAddress(addr)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    Account account = accountService.getAccount(addr).getData();
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    if (account.isEncrypted() && account.isLocked()) {
        if (!account.validatePassword(password)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    }
    if (StringUtils.isNotBlank(account.getAlias())) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_ALREADY_SET_ALIAS);
    }
    if (!StringUtils.validAlias(aliasName)) {
        return Result.getFailed(AccountErrorCode.ALIAS_FORMAT_WRONG);
    }
    if (!isAliasUsable(aliasName)) {
        return Result.getFailed(AccountErrorCode.ALIAS_EXIST);
    }
    byte[] addressBytes = account.getAddress().getAddressBytes();
    try {
        //
        AliasTransaction tx = new AliasTransaction();
        tx.setTime(TimeService.currentTimeMillis());
```

```
Alias alias = new Alias(addressBytes, aliasName);
tx.setTxData(alias);
```

```
CoinDataResult coinDataResult = accountLedgerService.getCoinData(addressBytes,
AccountConstant.ALIAS_NA, tx.size() ,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
```

```
if (!coinDataResult.isEnough()) {
    return Result.getFailed(AccountErrorCode.INSUFFICIENT_BALANCE);
}
CoinData coinData = new CoinData();
coinData.setFrom(coinDataResult.getCoinList());
Coin change = coinDataResult.getChange();
if (null != change) {
    //toList
    List<Coin> toList = new ArrayList<>();
    toList.add(change);
    coinData.setTo(toList);
}
```

```
Coin coin = new Coin(NulsConstant.BLACK_HOLE_ADDRESS,
AccountConstant.ALIAS_NA, 0);
coinData.addTo(coin);
```

```
tx.setCoinData(coinData);
tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
```

```
//
List<ECKey> signEckey = new ArrayList<>();
List<ECKey> scriptEckey = new ArrayList<>();
ECKey eckey = account.getEcKey(password);
```

```
//1
if((coinDataResult.getSignType() & 0x01) == 0x01){
    signEckey.add(eckey);
}
```

```
//1
if((coinDataResult.getSignType() & 0x02) == 0x02){
    scriptEckey.add(eckey);
}
```

```
SignatureUtil.createTransactionSignature(tx,scriptEckey,signEckey);
```

```
Result saveResult = accountLedgerService.verifyAndSaveUnconfirmedTransaction(tx);
```

```

        if (saveResult.isFailed()) {
            if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                //()
                Result rs =
accountLedgerService.getMaxAmountOfOnce(account.getAddress().getAddressBytes(), tx,
                TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
                if(rs.isSuccess()){
                    Na maxAmount = (Na)rs.getData();
                    rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
                    rs.setMsg(rs.getMsg() + maxAmount.toDouble());
                }
                return rs;

            }
            return saveResult;
        }

```

```

this.transactionService.newTx(tx);

```

```

Result sendResult = this.transactionService.broadcastTx(tx);
if (sendResult.isFailed()) {
    accountLedgerService.deleteTransaction(tx);
    return sendResult;
}
String hash = tx.getHash().getDigestHex();
return Result.getSuccess().setData(hash);
} catch (Exception e) {
    Log.error(e);
    return Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION);
}
}

```

```

/**

```

```

* ()

```

```

* 1.alias

```

```

* 2.account,account

```

```

* 3.account

```

```

* saveAlias

```

```

* 1. Save the alias to the database.

```

```

* 2. Take the corresponding account from the database, set the alias to account and save it to

```



the database.

- \* 3. Re-cache the modified account.

\*/

```
public Result saveAlias(AliasPo aliaspo) throws NulsException {
    try {
        Result result = aliasStorageService.saveAlias(aliaspo);
        if (result.isFailed()) {
            this.rollbackAlias(aliaspo);
        }
        AccountPo po = accountStorageService.getAccount(aliaspo.getAddress()).getData();
        if (null != po) {
            po.setAlias(aliaspo.getAlias());
            Result resultAcc = accountStorageService.updateAccount(po);
            if (resultAcc.isFailed()) {
                this.rollbackAlias(aliaspo);
            }
            Account account = po.toAccount();
            accountCacheService.localAccountMaps.put(account.getAddress().getBase58(),
account);
        }
    } catch (Exception e) {
        this.rollbackAlias(aliaspo);
        Log.error(e);
        return Result.getFailed(AccountErrorCode.FAILED);
    }
    return Result.getSuccess().setData(true);
}
```

```
public Alias getAlias(String alias) {
    AliasPo aliasPo = aliasStorageService.getAlias(alias).getData();
    return aliasPo == null ? null : aliasPo.toAlias();
}
```

```
public boolean isAliasUsable(String alias) {
    return null == getAlias(alias);
}
```

/\*\*

- \* (())

- \* 1.

- \* 2.account,

- \* 3.account

```

* rollbackAlias
* 1.Delete the alias data from the database.
* 2. Remove the corresponding account to clear the alias and restore it in the database.
* 3. Recache the account.
*/
public Result rollbackAlias(AliasPo aliasPo) throws NulsException {
    try {
        AliasPo po = aliasStorageService.getAlias(aliasPo.getAlias()).getData();
        if (po != null && Arrays.equals(po.getAddress(), aliasPo.getAddress())) {
            aliasStorageService.removeAlias(aliasPo.getAlias());
            Result<AccountPo> rs = accountStorageService.getAccount(aliasPo.getAddress());
            if (rs.isSuccess()) {
                AccountPo accountPo = rs.getData();
                accountPo.setAlias("");
                Result result = accountStorageService.updateAccount(accountPo);
                if (result.isFailed()) {
                    return Result.getFailed(AccountErrorCode.FAILED);
                }
                Account account = accountPo.toAccount();
                accountCacheService.localAccountMaps.put(account.getAddress().getBase58(),
account);
            }
        }
    } catch (Exception e) {
        Log.error(e);
        throw new NulsException(AccountErrorCode.ALIAS_ROLLBACK_ERROR);
    }
    return Result.getSuccess().setData(true);
}

/**
 *
 * Gets to set the alias transaction fee
 *
 * @param address
 * @param aliasName
 * @return
 */
public Result<Na> getAliasFee(String address, String aliasName) {
    if (!AddressTool.validAddress(address)) {
        Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
}

```

```

    }
    Account account = accountService.getAccount(address).getData();
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    byte[] addressBytes = account.getAddress().getAddressBytes();
    try {
        //
        AliasTransaction tx = new AliasTransaction();
        tx.setTime(TimeService.currentTimeMillis());
        Alias alias = new Alias(addressBytes, aliasName);
        tx.setTxData(alias);
        CoinDataResult coinDataResult = accountLedgerService.getCoinData(addressBytes,
AccountConstant.ALIAS_NA, tx.size() ,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        if (!coinDataResult.isEnough()) {
            return Result.getFailed(AccountErrorCode.INSUFFICIENT_BALANCE);
        }
        CoinData coinData = new CoinData();
        coinData.setFrom(coinDataResult.getCoinList());
        Coin change = coinDataResult.getChange();
        if (null != change) {
            //toList
            List<Coin> toList = new ArrayList<>();
            toList.add(change);
            coinData.setTo(toList);
        }
        Coin coin = new Coin(NulsConstant.BLACK_HOLE_ADDRESS, Na.parseNuls(1), 0);
        coinData.addTo(coin);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size() );
        return Result.getSuccess().setData(fee);
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION);
    }
}

```

/\*\*

\*

\* Gets to set the alias transaction fee

```

*
* @param address
* @param aliasName
* @return
*/
public Result<Na> getMultiAliasFee(String address, String aliasName) {
    if (!AddressTool.validAddress(address)) {
        Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    byte[] addressBytes = AddressTool.getAddress(address);
    try {
        //
        AliasTransaction tx = new AliasTransaction();
        tx.setTime(TimeService.currentTimeMillis());
        Alias alias = new Alias(addressBytes, aliasName);
        tx.setTxData(alias);
        CoinDataResult coinDataResult = accountLedgerService.getMutilCoinData(addressBytes,
AccountConstant.ALIAS_NA, tx.size() ,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        if (!coinDataResult.isEnough()) {
            return Result.getFailed(AccountErrorCode.INSUFFICIENT_BALANCE);
        }
        CoinData coinData = new CoinData();
        coinData.setFrom(coinDataResult.getCoinList());
        Coin change = coinDataResult.getChange();
        if (null != change) {
            //toList
            List<Coin> toList = new ArrayList<>();
            toList.add(change);
            coinData.setTo(toList);
        }
        Coin coin = new Coin(NulsConstant.BLACK_HOLE_ADDRESS, Na.parseNuls(1), 0);
        coinData.addTo(coin);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size() );
        return Result.getSuccess().setData(fee);
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION);
    }
}
}
/**

```

\*

\* Initiate a transaction to set alias.

\*

\* @param addr Address of account

\* @param password password of account

\* @param aliasName the alias to set

\* @return txhash

\*/

```
public Result<String> setMutilAlias(String addr,String signAddr, String aliasName, String password,List<String> pubKeys,int m,String txdata) {
```

```
//
```

```
Account account = accountService.getAccount(signAddr).getData();
```

```
if (null == account) {
```

```
    return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
```

```
}
```

```
if (account.isEncrypted() && account.isLocked()) {
```

```
    if (!account.validatePassword(password)) {
```

```
        return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
```

```
    }
```

```
}
```

```
try {
```

```
    AliasTransaction tx = new AliasTransaction();
```

```
    TransactionSignature transactionSignature = new TransactionSignature();
```

```
    List<P2PHKSignature> p2PHKSignatures = new ArrayList<>();
```

```
    List<Script> scripts = new ArrayList<>();
```

```
    byte[] addressBytes = AddressTool.getAddress(addr);
```

```
//txdata
```

```
if(txdata == null || txdata.trim().length() == 0){
```

```
    if (!StringUtils.validAlias(aliasName)) {
```

```
        return Result.getFailed(AccountErrorCode.ALIAS_FORMAT_WRONG);
```

```
    }
```

```
    if (!isAliasUsable(aliasName)) {
```

```
        return Result.getFailed(AccountErrorCode.ALIAS_EXIST);
```

```
    }
```

```
//
```

```
tx = new AliasTransaction();
```

```
Script redeemScript = ScriptBuilder.createNulsRedeemScript(m,pubKeys);
```

```
tx.setTime(TimeService.currentTimeMillis());
```

```
Alias alias = new Alias(addressBytes, aliasName);
```

```
tx.setTxData(alias);
```

```
//m*+
```

```
int scriptSignLenth = redeemScript.getProgram().length + m*72;
```

```

        CoinDataResult coinDataResult =
accountLedgerService.getMutilCoinData(addressBytes, AccountConstant.ALIAS_NA,
tx.size()+scriptSignLenth , TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        if (!coinDataResult.isEnough()) {
            return Result.getFailed(AccountErrorCode.INSUFFICIENT_BALANCE);
        }
        CoinData coinData = new CoinData();
        coinData.setFrom(coinDataResult.getCoinList());
        Coin change = coinDataResult.getChange();
        if (null != change) {
            //toList
            List<Coin> toList = new ArrayList<>();
            toList.add(change);
            coinData.setTo(toList);
        }
        Coin coin = new Coin(NulsConstant.BLACK_HOLE_ADDRESS, Na.parseNuls(1), 0);
        coinData.addTo(coin);
        tx.setCoinData(coinData);
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        //
        scripts.add(redeemScript);
        transactionSignature.setScripts(scripts);
    }
    //txdata
    else{
        byte[] txByte = Hex.decode(txdata);
        tx.parse(new NulsByteBuffer(txByte));
        transactionSignature.parse(new NulsByteBuffer(tx.getTransactionSignature()));
        p2PHKSignatures = transactionSignature.getP2PHKSignatures();
        scripts = transactionSignature.getScripts();
    }
    //
    P2PHKSignature p2PHKSignature = new P2PHKSignature();
    ECKey eckey = account.getEcKey(password);
    p2PHKSignature.setPublicKey(eckey.getPubKey());
    //hash
    p2PHKSignature.setSignData(accountService.signDigest(tx.getHash().getDigestBytes(),eckey));
    p2PHKSignatures.add(p2PHKSignature);
    Result result =
txMutilProcessing(tx,p2PHKSignatures,scripts,transactionSignature,addressBytes);
    return result;
} catch (Exception e) {

```

```

        Log.error(e);
        return Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION);
    }
}

/**
 *
 * Initiate a transaction to set alias.
 *
 * @param addr    Address of account
 * @param signAddr Address of account
 * @param password password of account
 * @param aliasName the alias to set
 * @return Result
 */
public Result<String> setMutilAlias(String addr,String signAddr, String aliasName, String
password) {
    //
    Account account = accountService.getAccount(signAddr).getData();
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    if (account.isEncrypted() && account.isLocked()) {
        if (!account.validatePassword(password)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    }
    try {
        byte[] addressBytes = AddressTool.getAddress(addr);
        Result<MultiSigAccount> sigAccountResult = accountService.getMultiSigAccount(addr);
        MultiSigAccount multiSigAccount = sigAccountResult.getData();
        Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
        if(redeemScript == null){
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
        }
        AliasTransaction tx = new AliasTransaction();
        TransactionSignature transactionSignature = new TransactionSignature();
        List<P2PHKSignature> p2PHKSignatures = new ArrayList<>();
        List<Script> scripts = new ArrayList<>();
        tx.setTime(TimeService.currentTimeMillis());
        Alias alias = new Alias(addressBytes, aliasName);

```

```

tx.setTxData(alias);
//m*+
int scriptSignLenth = redeemScript.getProgram().length + ((int)multiSigAccount.getM())*72;
CoinDataResult coinDataResult = accountLedgerService.getMutilCoinData(addressBytes,
AccountConstant.ALIAS_NA, tx.size()+scriptSignLenth ,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
if (!coinDataResult.isEnough()) {
    return Result.getFailed(AccountErrorCode.INSUFFICIENT_BALANCE);
}
CoinData coinData = new CoinData();
coinData.setFrom(coinDataResult.getCoinList());
Coin change = coinDataResult.getChange();
if (null != change) {
    //toList
    List<Coin> toList = new ArrayList<>();
    toList.add(change);
    coinData.setTo(toList);
}
Coin coin = new Coin(NulsConstant.BLACK_HOLE_ADDRESS, Na.parseNuls(1), 0);
coinData.addTo(coin);
tx.setCoinData(coinData);
tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
//
scripts.add(redeemScript);
transactionSignature.setScripts(scripts);
//
P2PHKSignature p2PHKSignature = new P2PHKSignature();
ECKey eckey = account.getEcKey(password);
p2PHKSignature.setPublicKey(eckey.getPubKey());
//hash
p2PHKSignature.setSignData(accountService.signDigest(tx.getHash().getDigestBytes(),eckey));
p2PHKSIGNATURES.add(p2PHKSignature);
Result result =
txMutilProcessing(tx,p2PHKSIGNATURES,scripts,transactionSignature,addressBytes);
return result;
} catch (Exception e) {
    Log.error(e);
    return Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION);
}
}

```



```

/**
 * A transfers NULS to B
 *
 * @param signAddr
 * @param password password of A
 * @param txdata
 * @return Result
 */
public Result signMultiAliasTransaction(String signAddr,String password,String txdata){
    try {
        Result<Account> accountResult = accountService.getAccount(signAddr);
        if (accountResult.isFailed()) {
            return accountResult;
        }
        Account account = accountResult.getData();
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(password, "the password can not be empty");
            if (!account.validatePassword(password)) {
                return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
            }
        }
        AliasTransaction tx = new AliasTransaction();
        TransactionSignature transactionSignature = new TransactionSignature();
        byte[] txByte = Hex.decode(txdata);
        tx.parse(new NulsByteBuffer(txByte));
        transactionSignature.parse(new NulsByteBuffer(tx.getTransactionSignature()));
        return accountLedgerService.txMultiProcess(tx,transactionSignature,account,password);
    }catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode());
    }catch (Exception e){
        Log.error(e);
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
}

public boolean isMutilAliasUsable(byte[] address,String aliasName) {
    List<AliasPo> list = aliasStorageService.getAliasList().getData();
    for (AliasPo aliasPo : list) {
        if (Arrays.equals(aliasPo.getAddress(), address)) {
            return false;
        }
    }
}

```

```

    }
    for (AliasPo aliasPo : list) {
        if (aliasName.equals(aliasPo.getAlias())) {
            return false;
        }
    }
    return true;
}

```

```

    public Result<String> txMutilProcessing(Transaction tx, List<P2PHKSignature>
p2PHKSignatures,List<Script> scripts,TransactionSignature transactionSignature,byte[] fromAddr)
throws NulsException,IOException {
    //M
    if(p2PHKSignatures.size() == SignatureUtil.getM(scripts.get(0))){
        //P2PHKSignatures
        Collections.sort(p2PHKSignatures,P2PHKSignature.PUBKEY_COMPARATOR);
        //P2PHKSignatures
        List<byte[]> signatures= new ArrayList<>();
        for (P2PHKSignature p2PHKSignatureTemp:p2PHKSignatures) {
            signatures.add(p2PHKSignatureTemp.getSignData().getSignBytes());
        }
        transactionSignature.setP2PHKSignatures(null);
        Script scriptSign =
ScriptBuilder.createNulsP2SHMultiSigInputScript(signatures,scripts.get(0));
        transactionSignature.getScripts().clear();
        transactionSignature.getScripts().add(scriptSign);
        tx.setTransactionSignature(transactionSignature.serialize());
        //
        Result saveResult = accountLedgerService.verifyAndSaveUnconfirmedTransaction(tx);
        if (saveResult.isFailed()) {
            if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                //()
                Result rs = accountLedgerService.getMaxAmountOfOnce(fromAddr, tx,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
                if (rs.isSuccess()) {
                    Na maxAmount = (Na) rs.getData();
                    rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
                    rs.setMsg(rs.getMsg() + maxAmount.toDouble());
                }
                return rs;
            }
        }
    }
}

```

```

        }
        return saveResult;
    }
    transactionService.newTx(tx);
    Result sendResult = transactionService.broadcastTx(tx);
    if (sendResult.isFailed()) {
        accountLedgerService.deleteTransaction(tx);
        return sendResult;
    }
    return Result.getSuccess().setData(tx.getHash().getDigestHex());
}
//
else{
    transactionSignature.setP2PHKSignatures(p2PHKSignatures);
    tx.setTransactionSignature(transactionSignature.serialize());
    return Result.getSuccess().setData(Hex.encode(tx.serialize()));
}
}
}
}

```

77:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\main\java\io\nuls\account\service\impl\AccountServiceImpl.java  
\*/

```

package io.nuls.account.service.impl;

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.*;
import io.nuls.account.service.AccountCacheService;
import io.nuls.account.service.AccountService;
import io.nuls.account.service.AliasService;
import io.nuls.account.storage.po.AccountPo;
import io.nuls.account.storage.po.AliasPo;
import io.nuls.account.storage.service.AccountStorageService;
import io.nuls.account.storage.service.AliasStorageService;
import io.nuls.account.storage.service.MultiSigAccountStorageService;
import io.nuls.account.util.AccountTool;
import io.nuls.core.tools.crypto.AESEncrypt;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.EncryptedData;
import io.nuls.core.tools.crypto.Exception.CryptoException;

```

```
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.NulsSignData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.script.Script;
import io.nuls.kernel.script.ScriptBuilder;
import io.nuls.kernel.script.ScriptUtil;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.kernel.validate.ValidateResult;
```

```
import java.io.IOException;
import java.math.BigInteger;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
@Service
```

```
public class AccountServiceImpl implements AccountService {
```

```
    private Lock locker = new ReentrantLock();
```

```
    @Autowired
```

```
    private AccountStorageService accountStorageService;
```

```
    @Autowired
```

```
    private MultiSigAccountStorageService multiSigAccountStorageService;
```

@Autowired

private AccountLedgerService accountLedgerService;

@Autowired

private AliasService aliasService;

@Autowired

private AliasStorageService aliasStorageService;

private AccountCacheService accountCacheService = AccountCacheService.getInstance();

@Override

```
public Result<List<Account>> createAccount(int count, String password) {
    if (count <= 0 || count > AccountTool.CREATE_MAX_SIZE) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
    if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
        return Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG);
    }
    locker.lock();
    try {
        List<Account> accounts = new ArrayList<>();
        List<AccountPo> accountPos = new ArrayList<>();
        for (int i = 0; i < count; i++) {
            Account account = AccountTool.createAccount();
            if (StringUtils.isNotBlank(password)) {
                account.encrypt(password);
            }
            accounts.add(account);
            AccountPo po = new AccountPo(account);
            accountPos.add(po);
        }
        Result result = accountStorageService.saveAccountList(accountPos);
        if (result.isFailed()) {
            return result;
        }
        for (Account account : accounts) {
            accountCacheService.localAccountMaps.put(account.getAddress().getBase58(),
account);
        }
        return Result.getSuccess().setData(accounts);
    }
```

```

    } catch (Exception e) {
        Log.error(e);
        throw new NulsRuntimeException(KernelErrorCode.FAILED);
    } finally {
        locker.unlock();
    }
}

```

@Override

```

public Result<List<Account>> createAccount(String password) {
    return createAccount(1, password);
}

```

@Override

```

public Result<List<Account>> createAccount(int count) {
    return createAccount(count, null);
}

```

@Override

```

public Result<List<Account>> createAccount() {
    return createAccount(1, null);
}

```

@Override

```

public Result removeAccount(String address, String password) {

    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    Account account = getAccountByAddress(address);
    if (account == null) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    //((), Already encrypted(Added password) and did not unlock, verify password
    if (account.isEncrypted() && account.isLocked()) {
        if (!account.validatePassword(password)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    }
    Result result = accountStorageService.removeAccount(account.getAddress());
    if (result.isFailed()) {
        return result;
    }
}

```

```

    }
    accountLedgerService.deleteUnconfirmedTx(account.getAddress().getAddressBytes());
    accountCacheService.localAccountMaps.remove(account.getAddress().getBase58());
    return Result.getSuccess().setData(true);
}

```

@Override

```

public Result<Account> updatePasswordByAccountKeyStore(AccountKeyStore keyStore,
String password) {
    AssertUtil.canNotEmpty(keyStore, AccountErrorCode.PARAMETER_ERROR.getMsg());
    AssertUtil.canNotEmpty(keyStore.getAddress(),
AccountErrorCode.PARAMETER_ERROR.getMsg());
    AssertUtil.canNotEmpty(password, AccountErrorCode.PARAMETER_ERROR.getMsg());
    Account account;
    byte[] priKey = null;
    if (null != keyStore.getPrikey() && keyStore.getPrikey().length > 0) {
        if (!ECKey.isValidPrivteHex(Hex.encode(keyStore.getPrikey()))) {
            return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
        }
        priKey = keyStore.getPrikey();
        try {
            account = AccountTool.createAccount(Hex.encode(priKey));
        } catch (NulsException e) {
            return Result.getFailed(AccountErrorCode.FAILED);
        }
    } else {
        try {
            account = AccountTool.createAccount();
        } catch (NulsException e) {
            return Result.getFailed(AccountErrorCode.FAILED);
        }
        account.setAddress(new Address(keyStore.getAddress()));
    }
    try {
        account.encrypt(password);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode());
    }
    if (StringUtils.isNotBlank(keyStore.getAlias())) {
        Alias aliasDb = aliasService.getAlias(keyStore.getAlias());
        if (null != aliasDb &&

```

```

account.getAddress().toString().equals(AddressTool.getStringAddressByBytes(aliasDb.getAddresses())) {
    account.setAlias(aliasDb.getAlias());
} else {
    List<AliasPo> list = aliasStorageService.getAliasList().getData();
    for (AliasPo aliasPo : list) {
        //,
        if
(AddressTool.getStringAddressByBytes(aliasPo.getAddress()).equals(account.getAddress().toString())) {
            account.setAlias(aliasPo.getAlias());
            break;
        }
    }
}
}

```

```

AccountPo po = new AccountPo(account);
Result result = accountStorageService.saveAccount(po);
if (result.isFailed()) {
    return result;
}
accountCacheService.localAccountMaps.put(account.getAddress().getBase58(), account);
accountLedgerService.importLedgerByAddress(account.getAddress().getBase58());
return Result.getSuccess().setData(account);
}

```

@Override

```

public Result<Account> importAccountFormKeyStore(AccountKeyStore keyStore, String password) {
    if (null == keyStore || StringUtils.isBlank(keyStore.getAddress())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
    if (!AddressTool.validAddress(keyStore.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    Account account;
    byte[] priKey = null;
    if (null != keyStore.getPrikey() && keyStore.getPrikey().length > 0) {
        if (!ECKKey.isValidPrivteHex(Hex.encode(keyStore.getPrikey()))) {
            return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
        }
    }
}

```



```

    priKey = keyStore.getPrikey();
    try {
        account = AccountTool.createAccount(Hex.encode(priKey));
    } catch (NulsException e) {
        return Result.getFailed(e.getErrorCode());
    }
    //keystore
    if (!account.getAddress().getBase58().equals(keyStore.getAddress())) {
        return Result.getFailed(AccountErrorCode.PRIVATE_KEY_WRONG);
    }
} else if (null == keyStore.getPrikey() && null != keyStore.getEncryptedPrivateKey()) {
    if (!StringUtils.validPassword(password)) {
        return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
    }
    try {
        priKey = AESEncrypt.decrypt(Hex.decode(keyStore.getEncryptedPrivateKey()),
password);
        account = AccountTool.createAccount(Hex.encode(priKey));
    } catch (CryptoException e) {
        return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
    } catch (NulsException e) {
        return Result.getFailed(e.getErrorCode());
    }
    //keystore
    if (!account.getAddress().getBase58().equals(keyStore.getAddress())) {
        return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
    }
} else {
    return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
}
Alias aliasDb = null;
if (StringUtils.isNotBlank(keyStore.getAlias())) {
    aliasDb = aliasService.getAlias(keyStore.getAlias());
}
if (null != aliasDb &&
AddressTool.getStringAddressByBytes(aliasDb.getAddress()).equals(account.getAddress().toStrin
g())) {
    account.setAlias(aliasDb.getAlias());
} else {
    List<AliasPo> list = aliasStorageService.getAliasList().getData();
    for (AliasPo aliasPo : list) {
        //,

```

```

        if
(AddressTool.getStringAddressByBytes(aliasPo.getAddress()).equals(account.getAddress().toStri
ng())) {
            account.setAlias(aliasPo.getAlias());
            break;
        }
    }
}
if (StringUtils.validPassword(password)) {
    try {
        account.encrypt(password);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode());
    }
}
AccountPo po = new AccountPo(account);
Result result = accountStorageService.saveAccount(po);
if (result.isFailed()) {
    return result;
}
accountCacheService.localAccountMaps.put(account.getAddress().getBase58(), account);
accountLedgerService.importLedgerByAddress(account.getAddress().getBase58());
return Result.getSuccess().setData(account);
}

```

@Override

```

public Result<Account> importAccountFormKeyStore(AccountKeyStore keyStore) {
    return importAccountFormKeyStore(keyStore, null);
}

```

@Override

```

public Result<Account> importAccount(String prikey, String password) {
    if (!ECKey.isValidPrivteHex(prikey)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
    Account account;
    try {
        account = AccountTool.createAccount(prikey);
    } catch (NulsException e) {
        return Result.getFailed(AccountErrorCode.PRIVATE_KEY_WRONG);
    }
}

```

```

    if (StringUtils.validPassword(password)) {
        try {
            account.encrypt(password);
        } catch (NulsException e) {
            Log.error(e);
            return Result.getFailed(e.getErrorCode());
        }
    }
    //
    //String alias = null;
    Account acc = getAccountByAddress(account.getAddress().toString());
    if (null == acc) {
        List<AliasPo> list = aliasStorageService.getAliasList().getData();
        for (AliasPo aliasPo : list) {
            //,
            if
(AddressTool.getStringAddressByBytes(aliasPo.getAddress()).equals(account.getAddress().toStri
ng())) {
                account.setAlias(aliasPo.getAlias());
                break;
            }
        }
    } else {
        account.setAlias(acc.getAlias());
    }
    Result res =
accountLedgerService.importLedgerByAddress(account.getAddress().getBase58());
    if (res.isFailed()) {
        return res;
    }
    AccountPo po = new AccountPo(account);
    Result result = accountStorageService.saveAccount(po);
    if (result.isFailed()) {
        return result;
    }
    accountCacheService.localAccountMaps.put(account.getAddress().getBase58(), account);

    return Result.getSuccess().setData(account);
}

```

@Override

```

public Result<Account> importAccount(String prikey) {

```

```

    return importAccount(prikey, null);
}

```

@Override

```

public Result<AccountKeyStore> exportAccountToKeyStore(String address, String password) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
    Account account = getAccountByAddress(address);
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    AccountKeyStore accountKeyStore = new AccountKeyStore();
    //(),
    if (account.isEncrypted() && account.isLocked()) {
        if (!account.validatePassword(password)) {
            return Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG);
        }
    }
    //(), If the account is encrypted (regardless of unlocked), the plaintext private key is not
    exported
    if (account.isEncrypted()) {
        EncryptedData encryptedData = new EncryptedData(account.getEncryptedPriKey());
        accountKeyStore.setEncryptedPrivateKey(Hex.encode(encryptedData.getEncryptedBytes()));
    } else {
        accountKeyStore.setPrikey(account.getPriKey());
    }
    accountKeyStore.setAddress(account.getAddress().toString());
    accountKeyStore.setAlias(account.getAlias());
    accountKeyStore.setPubKey(account.getPubKey());
    return Result.getSuccess().setData(accountKeyStore);
}

```

/\*\*

\* ,()

\* Get account object based on account address string

\*

\* @return Account

\*/

```

private Account getAccountByAddress(String address) {

```

```

    if (!AddressTool.validAddress(address)) {
        return null;
    }
    //,. If the account is unlocked, return directly to the unlocked account
    Account accountCache = accountCacheService.getAccountByAddress(address);
    if (null != accountCache) {
        return accountCache;
    }
    if (accountCacheService.localAccountMaps == null) {
        getAccountList();
    }
    return accountCacheService.localAccountMaps.get(address);
}

```

@Override

```

public Result<Account> getAccount(byte[] address) {
    if (null == address || address.length == 0) {
        return Result.getFailed(AccountErrorCode.NULL_PARAMETER);
    }
    String addr = AddressTool.getStringAddressByBytes(address);
    if (!AddressTool.validAddress(addr)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    return getAccount(addr);
}

```

@Override

```

public Result<Account> getAccount(String address) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    Account account = getAccountByAddress(address);
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    return Result.getSuccess().setData(account);
}

```

@Override

```

public Result<Account> getAccount(Address address) {
    if (null == address) {
        return Result.getFailed(AccountErrorCode.NULL_PARAMETER);
    }
}

```

```

    }
    return getAccount(address.toString());
}

```

@Override

```

public Result<Collection<Account>> getAccountList() {
    List<Account> list = new ArrayList<>();
    if (accountCacheService.localAccountMaps != null) {
        Collection<Account> values = accountCacheService.localAccountMaps.values();
        Iterator<Account> iterator = values.iterator();
        while (iterator.hasNext()) {
            list.add(iterator.next());
        }
    } else {
        accountCacheService.localAccountMaps = new ConcurrentHashMap<>();
        Result<List<AccountPo>> result = accountStorageService.getAccountList();
        if (result.isFailed()) {
            return Result.getFailed().setData(list);
        }
        List<AccountPo> poList = result.getData();
        Set<String> addressList = new HashSet<>();
        if (null == poList || poList.isEmpty()) {
            return Result.getSuccess().setData(list);
        }
        for (AccountPo po : poList) {
            Account account = po.toAccount();
            list.add(account);
            addressList.add(account.getAddress().getBase58());
        }
        for (Account account : list) {
            accountCacheService.localAccountMaps.put(account.getAddress().getBase58(),
account);
        }
    }
    list.sort(new Comparator<Account>() {
        @Override
        public int compare(Account o1, Account o2) {
            return (o2.getCreateTime().compareTo(o1.getCreateTime()));
        }
    });
    return Result.getSuccess().setData(list);
}

```

@Override

```
public Result<Address> getAddress(String pubKey) {
    AssertUtil.canNotEmpty(pubKey, "");
    try {
        Address address =
AccountTool.newAddress(ECKey.fromPublicOnly(Hex.decode(pubKey)));
        return Result.getSuccess().setData(address);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
}
```

@Override

```
public Result<Address> getAddress(byte[] pubKey) {
    AssertUtil.canNotEmpty(pubKey, "");
    try {
        Address address = AccountTool.newAddress(pubKey);
        return Result.getSuccess().setData(address);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
}
```

@Override

```
public Result isEncrypted(String address) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    Account account = getAccountByAddress(address);
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    Result result = new Result();
    boolean rs = account.isEncrypted();
    result.setSuccess(true);
    result.setData(rs);
    return result;
}
```

@Override

public NulsSignData signDigest(byte[] digest, Account account, String password) throws NulsException {

```
    if (null == digest || digest.length == 0) {
        throw new NulsException(AccountErrorCode.PARAMETER_ERROR);
    }
    //((), Already encrypted(Added password) and did not unlock, verify password
    if (account.isEncrypted() && account.isLocked()) {
        AssertUtil.canNotEmpty(password, "password can not be empty");
        return this.signDigest(digest, account.getPriKey(password));
    } else {
        return this.signDigest(digest, account.getPriKey());
    }
}
```

```
private NulsSignData signDigest(byte[] digest, byte[] priKey) {
    ECKey ecKey = ECKey.fromPrivate(new BigInteger(1, priKey));
    return signDigest(digest, ecKey);
}
```

@Override

```
public NulsSignData signDigest(byte[] digest, ECKey ecKey) {
    byte[] signbytes = ecKey.sign(digest);
    NulsSignData nulsSignData = new NulsSignData();
    nulsSignData.setSignAlgType(NulsSignData.SIGN_ALG_ECC);
    nulsSignData.setSignBytes(signbytes);
    return nulsSignData;
}
```

@Override

```
public Result<Balance> getBalance() throws NulsException {
    List<Account> list = new ArrayList<>();
    Balance balance = new Balance();
    Result<List<AccountPo>> result = accountStorageService.getAccountList();
    if (result.isFailed()) {
        return Result.getFailed().setData(balance);
    }
    List<AccountPo> poList = result.getData();
    if (null == poList || poList.isEmpty()) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    for (AccountPo po : poList) {
```



```

        Account account = po.toAccount();
        list.add(account);
    }

    for (Account account : list) {
        Result<Balance> resultBalance =
accountLedgerService.getBalance(account.getAddress().getAddressBytes());
        if (resultBalance.isSuccess()) {
            Balance temp = resultBalance.getData();
            if (null == temp) {
                continue;
            }
            balance.setBalance(balance.getBalance().add(temp.getBalance()));
            balance.setLocked(balance.getLocked().add(temp.getLocked()));
            balance.setUsable(balance.getUsable().add(temp.getUsable()));
        }
    }
    return Result.getSuccess().setData(balance);
}

```

@Override

```

public Result<String> getAlias(byte[] address) {
    return getAlias(AddressTool.getStringAddressByBytes(address));
}

```

@Override

```

public Result<String> getAlias(String address) {
    Account account = getAccountByAddress(address);
    if (null != account) {
        return Result.getSuccess().setData(account.getAlias());
    }
    String alias = null;
    List<AliasPo> list = aliasStorageService.getAliasList().getData();
    for (AliasPo aliasPo : list) {
        if (AddressTool.getStringAddressByBytes(aliasPo.getAddress()).equals(address)) {
            alias = aliasPo.getAlias();
            break;
        }
    }
    return Result.getSuccess().setData(alias);
}

```

```

@Override
public Result<Address> createMultiAccount(List<String> pubkeys, int m) {
    locker.lock();
    try {
        Script redeemScript = ScriptBuilder.createNulsRedeemScript(m, pubkeys);
        Address address = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.P2SH_ADDRESS_TYPE,
SerializeUtils.sha256hash160(redeemScript.getProgram()));
        MultiSigAccount account = new MultiSigAccount();
        account.setAddress(address);
        account.setM(m);
        account.addPubkeys(pubkeys);
        Result result = this.multiSigAccountStorageService.saveAccount(account.getAddress(),
account.serialize());
        if (result.isFailed()) {
            return result;
        }
        return result.setData(account);
    } catch (Exception e) {
        Log.error(e);
        throw new NulsRuntimeException(KernelErrorCode.FAILED);
    } finally {
        locker.unlock();
    }
}

```

```

/**
 *
 * Query all account collections.
 *
 * @return account list of all accounts.
 */

```

```

@Override
public Result<List<MultiSigAccount>> getMultiSigAccountList() {
    List<byte[]> list = this.multiSigAccountStorageService.getAccountList().getData();
    if (null == list) {
        return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND);
    }
    List<MultiSigAccount> accountList = new ArrayList<>();
    for (byte[] bytes : list) {
        MultiSigAccount account = new MultiSigAccount();
        try {

```

```

        account.parse(new NulsByteBuffer(bytes, 0));
    } catch (NulsException e) {
        Log.error(e);
    }
    accountList.add(account);
}
return new Result<List<MultiSigAccount>>().setData(accountList);
}

/**
 *
 * Get the details of the locally stored multi-sign account based on the address
 *
 * @param address
 * @return
 */
@Override
public Result<MultiSigAccount> getMultiSigAccount(String address) throws Exception {
    byte[] bytes =
this.multiSigAccountStorageService.getAccount(Address.fromHashs(address)).getData();
    if (null == bytes) {
        return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND);
    }
    MultiSigAccount account = new MultiSigAccount();
    account.parse(new NulsByteBuffer(bytes, 0));
    List<AliasPo> list = aliasStorageService.getAliasList().getData();
    for (AliasPo aliasPo : list) {
        if (aliasPo.getAddress()[2] != NulsContext.P2SH_ADDRESS_TYPE) {
            continue;
        }
        if (Arrays.equals(aliasPo.getAddress(), account.getAddress().getAddressBytes())) {
            account.setAlias(aliasPo.getAlias());
            break;
        }
    }
    return Result.getSuccess().setData(account);
}

/**
 *
 *
 * @param addressStr

```

```

* @param pubkeys
* @param m
* @return
*/
@Override
public Result<Boolean> saveMultiSigAccount(String addressStr, List<String> pubkeys, int m) {
    Script redeemScript = ScriptBuilder.createNulsRedeemScript(m, pubkeys);
    Address address = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.P2SH_ADDRESS_TYPE,
SerializeUtils.sha256hash160(redeemScript.getProgram()));
    if (!AddressTool.getStringAddressByBytes(address.getAddressBytes()).equals(addressStr)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
    }
    MultiSigAccount account = new MultiSigAccount();
    account.setAddress(address);
    account.setM(m);
    account.addPubkeys(pubkeys);
    Result result = null;
    try {
        result = this.multiSigAccountStorageService.saveAccount(account.getAddress(),
account.serialize());
    } catch (IOException e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.SERIALIZE_ERROR);
    }
    if (result.isFailed()) {
        return result;
    }
    return result.setData(addressStr);
}

/**
*
*/
@Override
public Result<Boolean> removeMultiSigAccount(String address) {
    try {
        Address addressObj = Address.fromHashs(address);
        Result result = this.multiSigAccountStorageService.getAccount(addressObj);
        if (result.isFailed() || result.getData() == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
        }
    }
}

```

```

        return this.multiSigAccountStorageService.removeAccount(addressObj);
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed();
    }
}
}
}

```

78:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\main\java\io\nuls\account\validator\AliasTransactionValidator.java  
\*/

```
package io.nuls.account.validator;
```

```

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.Alias;
import io.nuls.account.service.AccountService;
import io.nuls.account.service.AliasService;
import io.nuls.account.storage.po.AliasPo;
import io.nuls.account.storage.service.AliasStorageService;
import io.nuls.account.tx.AliasTransaction;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.constant.SeverityLevelEnum;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.CoinData;

```

```

import io.nuls.kernel.model.Na;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.script.TransactionSignature;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

```

```

import java.util.Arrays;
import java.util.List;

/**
 * @author: Charlie
 */
@Component
public class AliasTransactionValidator implements NulsDataValidator<AliasTransaction> {

    @Autowired
    private AliasService aliasService;

    @Autowired
    private AliasStorageService aliasStorageService;

    @Autowired
    private AccountLedgerService accountLedgerService;

    @Autowired
    private AccountService accountService;

    @Override
    public ValidateResult validate(AliasTransaction tx) {
        Alias alias = tx.getTxData();
        if (tx.isSystemTx()) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_TYPE_ERROR);
        }
        if (NulsContext.CONTRACT_ADDRESS_TYPE == alias.getAddress()[2]) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
AccountErrorCode.ADDRESS_ERROR);
        }
        if (!StringUtils.validAlias(alias.getAlias())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
AccountErrorCode.ALIAS_FORMAT_WRONG);
        }
        if (!aliasService.isAliasUsable(alias.getAlias())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
AccountErrorCode.ALIAS_EXIST);
        }
        List<AliasPo> list = aliasStorageService.getAliasList().getData();
        for (AliasPo aliasPo : list) {

```

```

        if (Arrays.equals(aliasPo.getAddress(), alias.getAddress())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
AccountErrorCode.ACCOUNT_ALREADY_SET_ALIAS);
        }
    }
    CoinData coinData = tx.getCoinData();
    if (null == coinData) {
        return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.COINDATA_NOT_FOUND);
    }
    if (null == coinData.getTo() || coinData.getTo().isEmpty()) {
        boolean burned = false;
        for (Coin coin : coinData.getTo()) {
            if (ArraysTool.arrayEquals(coin.getOwner(), NulsConstant.BLACK_HOLE_ADDRESS)
&& coin.getNa().equals(Na.NA)) {
                burned = true;
            }
        }
        if (!burned) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
AccountErrorCode.MUST_BURN_A_NULS);
        }
    }

    TransactionSignature sig = new TransactionSignature();
    try {
        sig.parse(tx.getTransactionSignature(), 0);
    } catch (NulsException e) {
        Log.error(e);
        return ValidateResult.getFailedResult(this.getClass().getName(), e.getErrorCode());
    }

    boolean sign;
    try {
        sign = SignatureUtil.containsAddress(tx, tx.getTxData().getAddress());
    } catch (NulsException e) {
        sign = false;
    }
    if (!sign) {
        ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
AccountErrorCode.ADDRESS_ERROR);
    }

```

```

        result.setLevel(SeverityLevelEnum.FLAGRANT_FOUL);
        return result;
    }
    return ValidateResult.getSuccessResult();
}

}

```

79:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\test\java\io\nuls\account\model\AddressTest.java  
 \*/

```
package io.nuls.account.model;
```

```
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.SerializeUtils;
import org.junit.Test;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class AddressTest {
```

```
    @Test
```

```
    public void test() {
```

```
        short chainId = 8964;
```

```
        while (true) {
```

```
            ECKey ecKey = new ECKey();
```

```
            String address = getAddress(chainId, ecKey.getPubKey());
```

```
            System.out.println(address );//+ "::::::::" + ecKey.getPrivateKeyAsHex());
```

```
        }
```

```
    }
```

```
    private String getAddress(short chainId, byte[] publicKey) {
```

```
        if (publicKey == null) {
```

```
            return null;
```

```
        }
```

```
        byte[] hash160 = SerializeUtils.sha256hash160(publicKey);
```

```
        Address address = new Address(chainId, (byte) 1, hash160);
```



```

        return address.getBase58();
    }

    private byte getXor(byte[] body) {
        byte xor = 0x00;
        for (int i = 0; i < body.length; i++) {
            xor ^= body[i];
        }

        return xor;
    }
}

80:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
base\src\test\java\io\nuls\account\service\AccountBaseServiceTest.java
*/

package io.nuls.account.service;

import io.nuls.account.model.Account;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.core.SpringLiteContext;
import io.nuls.kernel.model.Result;
import org.junit.BeforeClass;
import org.junit.Test;

import java.util.List;

import static org.junit.Assert.*;

public class AccountBaseServiceTest {

    protected static AccountService accountService;
    protected static AccountBaseService accountBaseService;
    @BeforeClass
    public static void beforeTest() {
        MicroKernelBootstrap kernel = MicroKernelBootstrap.getInstance();
        kernel.init();
        kernel.start();
    }
}

```

```

    accountService = SpringLiteContext.getBean(AccountService.class);
    accountBaseService = SpringLiteContext.getBean(AccountBaseService.class);

}

@Test
public void getPrivateKeyTest() {
    List<Account> accounts = this.accountService.createAccount(1, "nuls123456").getData();
    Account account = accounts.get(0);
    Result result = accountBaseService.getPrivateKey(account.getAddress().toString(),
"nuls123456");
    assertTrue(result.isSuccess());
    try {
        account.unlock("nuls123456");
    } catch (NulsException e) {
        e.printStackTrace();
    }
    assertEquals(Hex.decode((String)result.getData()), account.getPriKey());

    List<Account> accounts2 = this.accountService.createAccount(1, "").getData();
    Account account2 = accounts2.get(0);
    Result result2 = accountBaseService.getPrivateKey(account2.getAddress().toString(), "");
    assertTrue(result2.isSuccess());
    assertEquals(Hex.decode((String)result2.getData()), account2.getPriKey());
}

@Test
public void setPassword() {
    List<Account> accounts = this.accountService.createAccount(1, "").getData();
    Account account = accounts.get(0);
    accountBaseService.setPassword(account.getAddress().toString(),"nuls123456");
    Account acc = accountService.getAccount(account.getAddress()).getData();
    try {
        assertTrue(acc.unlock("nuls123456"));
        assertEquals(acc.getPriKey(), account.getPriKey());
    } catch (NulsException e) {
        e.printStackTrace();
    }
}

@Test
public void changePassword() {

```

```

        List<Account> accounts = this.accountService.createAccount(1, "nuls123456").getData();
        Account account = accounts.get(0);
        accountBaseService.changePassword(account.getAddress().toString(),"nuls123456",
        "nuls111111");
        Account acc = accountService.getAccount(account.getAddress()).getData();
        try {
            assertFalse(acc.unlock("nuls123456"));
            assertTrue(acc.unlock("nuls111111"));
            assertEquals(acc.getPriKey(), account.getPriKey());
        } catch (NulsException e) {

        }
    }
}

```

81:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\test\java\io\nuls\account\service\AccountServiceTest.java

```

package io.nuls.account.service;

import io.nuls.account.model.AccountKeyStore;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.db.module.impl.LevelDbModuleBootstrap;
import io.nuls.account.model.Account;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;

import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.kernel.lite.core.SpringLiteContext;
import io.nuls.kernel.model.Result;
import org.junit.BeforeClass;
import org.junit.Test;
import java.util.List;
import static org.junit.Assert.*;
import static org.junit.Assert.assertEquals;

/**
 * @author: Niels Wang
 */
public class AccountServiceTest {

```

```
protected static AccountService accountService;
```

```
@BeforeClass
```

```
public static void beforeTest() {  
    MicroKernelBootstrap kernel = MicroKernelBootstrap.getInstance();  
    kernel.init();  
    kernel.start();  
    LevelDbModuleBootstrap db = new LevelDbModuleBootstrap();  
    db.init();  
    db.start();  
    accountService = SpringLiteContext.getBean(AccountService.class);  
  
}
```

```
@Test
```

```
public void createAccount() {  
  
    Result<List<Account>> result = this.accountService.createAccount(0, null);  
    assertTrue(result.isFailed());  
    assertNotNull(result.getMsg());  
  
    //  
    result = this.accountService.createAccount(1, null);  
    assertTrue(result.isSuccess());  
    assertNotNull(result.getData());  
    assertEquals(result.getData().size(), 1);  
    //  
  
    result = this.accountService.createAccount(5, null);  
    assertTrue(result.isSuccess());  
    assertNotNull(result.getData());  
    assertEquals(result.getData().size(), 5);  
    //  
  
    //  
    result = this.accountService.createAccount(10000, null);  
    assertTrue(result.isFailed());  
    assertNotNull(result.getMsg());  
  
    //  
    result = this.accountService.createAccount(1, null);
```

```
assertTrue(result.isSuccess());
assertNotNull(result.getMsg());
```

```
// nuls123456
```

```
result = this.accountService.createAccount(1, "nuls123456");
assertTrue(result.isSuccess());
assertNotNull(result.getData());
assertEquals(result.getData().size(), 1);
//
```

```
result = this.accountService.createAccount(6, "nuls123456");
assertTrue(result.isSuccess());
assertNotNull(result.getData());
assertEquals(result.getData().size(), 6);
//
```

```
result = this.accountService.createAccount(10000, null);
assertTrue(result.isFailed());
assertNotNull(result.getMsg());
```

```
}
```

```
@Test
```

```
public void removeAccount() {
    List<Account> accounts = this.accountService.createAccount(2, "nuls123456").getData();
    Result result0 = accountService.removeAccount(accounts.get(0).getAddress().toString(),
"nuls123456");
    assertTrue(result0.isSuccess());
    Result result1 = accountService.removeAccount(accounts.get(1).getAddress().toString(),
"123456");
    assertTrue(result1.isFailed());
}
```

```
@Test
```

```
public void getAccount(){
    List<Account> accounts = this.accountService.createAccount(2, "nuls123456").getData();
    Account account = accounts.get(0);
    assertNotNull(accountService.getAccount(account.getAddress()).getData());
assertEquals(accountService.getAccount(account.getAddress()).getData().getAddress().toString(),
account.getAddress().toString());
```

```

    Account acc1 = accountService.getAccount(account.getAddress().toString()).getData();
    assertNotNull(acc1);
    assertEquals(acc1.getAddress().toString(), account.getAddress().toString());

    Account acc2 =
accountService.getAccount(account.getAddress().getAddressBytes()).getData();
    assertNotNull(acc2);
    assertEquals(acc2.getAddress().toString(), account.getAddress().toString());
}

@Test
public void getAccountlist(){
    this.accountService.createAccount(50, "nuls123456").getData();
    assertTrue(this.accountService.getAccountList().getData().size()==50);
}

@Test
public void exportAccountToKeyStore(){
    List<Account> accounts = this.accountService.createAccount(1, "nuls123456").getData();
    Account account = accounts.get(0);
    Result<AccountKeyStore> result =
accountService.exportAccountToKeyStore(account.getAddress().toString(), "nuls123456");
    try {
        System.out.println(JSONUtils.obj2PrettyJson(result.getData()));
    } catch (Exception e) {
        e.printStackTrace();
    }
    assertNotNull(result.getData());
}

@Test
public void importAccount(){
    AccountKeyStore accountKeyStore = new AccountKeyStore();
    accountKeyStore.setAddress("Ns5fRyLX5Z6aNrxSGijUcR9SwjnVivi");
    accountKeyStore.setAlias(null);
accountKeyStore.setEncryptedPrivateKey("8fd44822ecf4589c02722f2b8f8e8636cd3106c8b85f0fb
c87c78bdef64512f7c604e42e3d829fdb981fb135ed46dc8");
    accountKeyStore.setPrikey(null);
accountKeyStore.setPubKey(Hex.decode("025e11c5bba00490c15ff9f0c5e24c7141204282fec3ef9
b179cc77d947161c4cc"));
    Result<Account> result = accountService.importAccountFormKeyStore(accountKeyStore,

```

```

"nuls123456");
    assertTrue(result.isSuccess());
    assertNotNull(accountService.getAccount(result.getData().getAddress()));
}

@Test
public void isEncrypted(){
    List<Account> accounts = this.accountService.createAccount(1, "nuls123456").getData();
    Account account = accounts.get(0);
    assertTrue(accountService.isEncrypted(account.getAddress().toString()).isSuccess());

    List<Account> accounts2 = this.accountService.createAccount(1, "").getData();
    Account account2 = accounts2.get(0);
    assertTrue(accountService.isEncrypted(account2.getAddress().toString()).isFailed());
}

public static void showAccount(Account account) {
    System.out.println("---- account info ----");
    System.out.println("Address " + account.getAddress().getBase58());
    System.out.println("Public key " + Hex.encode(account.getPubKey()));
    System.out.println("Private key" + Hex.encode(account.getPriKey()));
    System.out.println("Encrypted pri key " + Hex.encode(account.getEncryptedPriKey()));
    System.out.println("key object");
    System.out.println("\tpublic key" + account.getEcKey().getPublicKeyAsHex());
    try {
        System.out.println("\tprivate key" + Hex.encode(account.getEcKey().getPrivKeyBytes()));
    } catch (EKey.MissingPrivateKeyException e) {
        System.out.println("\tprivate key is NULL");
    }
    System.out.println("\tencrypted pkey " + account.getEcKey().getEncryptedPrivateKey());
    System.out.println("---- account info end----");
}
}

```

82:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\test\java\io\nuls\account\service\AccountTest.java

```

*/
package io.nuls.account.service;

import io.nuls.account.model.Account;
import io.nuls.account.util.AccountTool;

```

```

import io.nuls.core.tools.crypto.AESEncrypt;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.utils.SerializeUtils;
import org.junit.Test;

import java.math.BigInteger;
import java.util.Arrays;
import java.util.Random;

/**
 * @author: Charlie
 * @date: 2018/9/3
 */
public class AccountTest {

    /**
     *
     * keystorekeystore
     */
    @Test
    public void decryptTest(){
        String acc = "NsdvWJqEtRcekQVf88UoKH4Y789ka9YD";
        String encryptedPrivateKey =
"3254fad53298a1fbf1fcda27acdb4bcd1c895001a8443233d4d9ce9ca803c7e6dcd9bd32813668a9a
ad2687eb4e1173b0";

        while (true){
            String pwd = getPwd();
            try {
                byte[] priKey = AESEncrypt.decrypt(Hex.decode(encryptedPrivateKey), pwd);
                Account account = AccountTool.createAccount(Hex.encode(priKey));
                if(!acc.equals(account.getAddress().getBase58())){
                    System.out.println("===== ");
                    System.out.println("
3254fad53298a1fbf1fcda27acdb4bcd1c895001a8443233d4d9ce9ca803c7e6dcd9bd32813668a9a
d2687eb4e1173b0");
                    System.out.println("" + pwd);
                    System.out.println("" + Hex.encode(priKey));

```



```

        System.out.println();
        System.out.println("nuls123456");
        System.out.println("'" +
Hex.encode(AESEncrypt.decrypt(Hex.decode(encryptedPrivateKey), "nuls123456")));
        //break;
    }
} catch (Exception e) {
    System.out.println("'" + pwd);
}
}
}
}

```

```

private String getPwd(){
    char[] word = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};
        //,'A','B','C','D','E','F','G','H','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
    char[] number = {'0','1','2','3','4','5','6','7','8','9'};

    Random rand = new Random();
    String pwd = "nuls";
    for(int i = 0; i<10;i++){
        int randNumber = rand.nextInt(word.length);
        if(i>3){
            randNumber = rand.nextInt(number.length);
            char num =number[randNumber];
            pwd += num;
            continue;
        }
        /* char w =word[randNumber];
        pwd += w;*/
    }
    return pwd;
}
}

```

//

@Test

```

public void privateKeyTest(){
    for(int i=1; i<1000000000;i++){
        ECKey ecKey = new ECKey();
        String prikey = Hex.encode(ecKey.getPrivKeyBytes());
        ECKey ecKey2 = ECKey.fromPrivate(new BigInteger(1, Hex.decode(prikey)));
    }
}

```

```

    ECKey ecKey3 = ECKey.fromPrivate(new BigInteger(Hex.decode(prikey)));
    if(!Arrays.equals(ecKey.getPubKey(), ecKey3.getPubKey())){
        //1.0.1bug, BigInteger,
        System.out.println("error: " + prikey);
        Address address1 = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(ecKey.getPubKey()));
        Address address3 = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(ecKey3.getPubKey()));
        System.out.println(": " + address1.getBase58());
        System.out.println("3: " + address3.getBase58());
        try {
            System.out.println("ecKey: " + JSONUtils.obj2json(ecKey));
            System.out.println("ecKey3: " + JSONUtils.obj2json(ecKey3));
        } catch (Exception e) {
            e.printStackTrace();
        }
        break;
    }
    if (!Arrays.equals(ecKey.getPubKey(), ecKey2.getPubKey())) {
        System.out.println("error: " + prikey);
        Address address1 = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(ecKey.getPubKey()));
        Address address2 = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(ecKey2.getPubKey()));
        System.out.println(": " + address1.getBase58());
        System.out.println(": " + address2.getBase58());
        try {
            System.out.println("ecKey: " + JSONUtils.obj2json(ecKey));
            System.out.println("ecKey: " + JSONUtils.obj2json(ecKey2));
        } catch (Exception e) {
            e.printStackTrace();
        }
        break;
    } else {
        System.out.println(i + " ok: " + prikey);
        Address addr = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(ecKey.getPubKey()));
        if(addr.getBase58().endsWith("lichao")||addr.getBase58().endsWith("Charlie")){
            System.out.println(" yeah yeah yeah: " + addr.getBase58());
            System.out.println(" yeah yeah yeah: " + prikey);
            break;
        }
    }
}

```

```

    }
    /* try {
        Thread.sleep(1L);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }*/
}
}
}
}
}

```

83:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-base\src\test\java\io\nuls\account\service\AliasServiceTest.java  
\*/

```
package io.nuls.account.service;
```

```

import io.nuls.account.model.Account;
import io.nuls.account.model.Alias;
import io.nuls.account.storage.po.AliasPo;
import io.nuls.account.storage.service.AccountStorageService;
import io.nuls.db.module.impl.LevelDbModuleBootstrap;
import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.core.SpringLiteContext;
import io.nuls.kernel.model.Result;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

```

```
import java.util.List;
```

```

import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertTrue;

```

```
public class AliasServiceTest {
```

```

    protected static AccountService accountService;
    protected static AliasService aliasService;

```

```
@Before
```

```
public void beforeClass(){
```

```
    MicroKernelBootstrap kernel = MicroKernelBootstrap.getInstance();
```

```

kernel.init();
kernel.start();
LevelDbModuleBootstrap db = new LevelDbModuleBootstrap();
db.init();
db.start();
accountService = SpringLiteContext.getBean(AccountService.class);
aliasService = SpringLiteContext.getBean(AliasService.class);
}

@Test
public void setAlias() {
    List<Account> accounts = accountService.createAccount(1, "nuls123456").getData();
    Account account = accounts.get(0);
    Result result = aliasService.setAlias(account.getAddress().toString(), "nuls123456",
"Charlie555");
    assertTrue(result.isSuccess());
}

@Test
public void saveAlias() {
    List<Account> accounts = accountService.createAccount(1, "nuls123456").getData();
    Account account = accounts.get(0);
    Alias alias = new Alias(account.getAddress().getAddressBytes(), "lichao");
    try {
        assertTrue(aliasService.saveAlias(new AliasPo(alias)).isSuccess());
    } catch (NulsException e) {
        e.printStackTrace();
    }
}
}
}

```

84:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\BackupAccountProcessor.java  
\*/

```

package io.nuls.account.rpc.cmd;

import io.nuls.account.constant.AccountConstant;
import io.nuls.kernel.model.Address;
import io.nuls.account.rpc.model.AccountKeyStoreDto;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.Log;

```

```

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

/**
 * @author: Charlie
 */
public class BackupAccountProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "backup";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> the account you want to back up - Required")
            .newLine("\t[path] The folder of the export file, defaults to the current directory");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "backup <address> [path] --backup the account key store";
    }
}

```

@Override

```
public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length < 2 || length > 3) {
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    if (!AddressTool.validAddress(args[1])) {
        return false;
    }
    return true;
}
```

@Override

```
public CommandResult execute(String[] args) {
    String address = args[1];
    String path = args.length == 3 ? args[2] : System.getProperty("user.dir");
    RpcClientResult res = CommandHelper.getPassword(address, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("password", password);
    RpcClientResult result = restFul.post("/account/export/" + address, parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    AccountKeyStoreDto accountKeyStoreDto = new AccountKeyStoreDto((Map<String,
Object>) result.getData());
    Result rs = backUpFile(path, accountKeyStoreDto);
    if (rs.isFailed()) {
        return CommandResult.getFailed(rs.getMsg());
    }
    return CommandResult.getSuccess((String)rs.getData());
}
```

/\*\*

\*

\* Export file

```

*
* @param path
* @param accountKeyStoreDto
* @return
*/
private Result backUpFile(String path, AccountKeyStoreDto accountKeyStoreDto) {
    File backupFile = new File(path);
    //if not directory , create directory
    if (!backupFile.isDirectory()) {
        if (!backupFile.mkdirs()) {
            return Result.getFailed(KernelErrorCode.FILE_OPERATION_FAILED);
        }
        if (!backupFile.exists() && !backupFile.mkdir()) {
            return Result.getFailed(KernelErrorCode.FILE_OPERATION_FAILED);
        }
    }
    String fileName =
accountKeyStoreDto.getAddress().concat(AccountConstant.ACCOUNTKEYSTORE_FILE_SUFFIX);
    backupFile = new File(backupFile, fileName);
    try {
        if (!backupFile.exists() && !backupFile.createNewFile()) {
            return Result.getFailed(KernelErrorCode.FILE_OPERATION_FAILED);
        }
    } catch (IOException e) {
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    }
    FileOutputStream fileOutputStream = null;
    try {
        fileOutputStream = new FileOutputStream(backupFile);
        fileOutputStream.write(JSONUtils.obj2json(accountKeyStoreDto).getBytes());
    } catch (Exception e) {
        return Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION);
    } finally {
        if (fileOutputStream != null) {
            try {
                fileOutputStream.close();
            } catch (IOException e) {
                Log.error(e);
            }
        }
    }
}

```

```

        return Result.getSuccess().setData("The path to the backup file is " + path + File.separator +
fileName);
    }

}

```

85:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\CreateMultiAliasProcess.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

```

```

import java.util.HashMap;
import java.util.Map;

```

```
/**
```

```
* @author: tag
```

```
*/
```

```

public class CreateMultiAliasProcess implements CommandProcessor {
    private RestFulUtils restFul = RestFulUtils.getInstance();
    @Override
    public String getCommand() {
        return "setMultiAlias";
    }
}

```

```
@Override
```

```

public String getHelp() {
    CommandBuilder builder = new CommandBuilder();
    builder.newLine(getCommandDescription())
        .newLine("\t<address> The address of the account, - Required")
        .newLine("\t<alias> The alias of the account, the bytes for the alias is between 1 and 20

```

```
" +
```

```

        "(only lower case letters, Numbers and underline, the underline should not be at
the begin and end), - Required")

```



```
        .newLine("\t<signAddress> \tsign address address - Required");  
    return builder.toString();  
}
```

@Override

```
public String getCommandDescription() {  
    return "setMultiAlias <address> <alias> <signAddress> --Set an alias for the multi account ";  
}
```

@Override

```
public boolean argsValidate(String[] args) {  
    if(args.length != 4){  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
    if (StringUtils.isBlank(args[1])||StringUtils.isBlank(args[3])) {  
        return false;  
    }  
    if (!StringUtils.validAlias(args[2])) {  
        return false;  
    }  
    return true;  
}
```

@Override

```
public CommandResult execute(String[] args) {  
    String signAddress = args[3];  
    RpcClientResult res = CommandHelper.getPassword(signAddress, restFul);  
    if(!res.isSuccess()){  
        return CommandResult.getFailed(res);  
    }  
    String password = (String)res.getData();  
    Map<String, Object> parameters = new HashMap<>();  
    parameters.put("address", args[1]);  
    parameters.put("alias", args[2]);  
    parameters.put("signAddress", args[3]);  
    parameters.put("password", password);  
    RpcClientResult result = restFul.post("/account/multiAccount/mutilAlias", parameters);  
    if(result.isFailed()){
```

```

        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(CommandResult.dataMultiTransformValue(result));
}
}

```

86:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\CreateMultiSigAccountProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;
import java.util.Map;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class CreateMultiSigAccountProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {
        return "createmultiaccount";
    }

```

```
    @Override
```

```
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t[pks] Multiple public keys separated by '\',\''. -required")
            .newLine("\t[number] The minimum number of signatures required to initiate a  
transaction,0 is all. -required");
    }

```

```
    return builder.toString();  
}
```

@Override

```
public String getCommandDescription() {  
    return "createmultiaccount <pk> <m> --create Multi-signature account";  
}
```

@Override

```
public boolean argsValidate(String[] args) {  
    int length = args.length;  
    if (length != 3) {  
        return false;  
  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
    if (!StringUtils.isNumeric(args[2])) {  
        return false;  
    }  
    if (StringUtils.isBlank(args[1])) {  
        return false;  
    }  
    return true;  
}
```

@Override

```
public CommandResult execute(String[] args) {  
  
    Map<String, Object> parameters = new HashMap<>();  
    String pubkeysStr = args[1];  
    parameters.put("pubkeys", pubkeysStr.split(","));  
    parameters.put("m", args[2]);  
    RpcClientResult result = restFul.post("/account/createMultiAccount", parameters);  
    if (result.isFailed()) {  
        return CommandResult.getFailed(result);  
    }  
    return CommandResult.getResult(result);  
}  
}
```

```
87:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\cmd\CreateProcessor.java  
*/
```

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.core.tools.str.StringUtils;  
import io.nuls.kernel.model.CommandResult;  
import io.nuls.kernel.model.RpcClientResult;  
import io.nuls.kernel.processor.CommandProcessor;  
import io.nuls.kernel.utils.CommandBuilder;  
import io.nuls.kernel.utils.CommandHelper;  
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;  
import java.util.Map;
```

```
/**  
 * @author: Charlie  
 */  
public class CreateProcessor implements CommandProcessor {  
  
    private RestFulUtils restFul = RestFulUtils.getInstance();  
  
    @Override  
    public String getCommand() {  
        return "create";  
    }  
  
    @Override  
    public String getHelp() {  
        CommandBuilder builder = new CommandBuilder();  
        builder.newLine(getCommandDescription())  
            .newLine("\t[number] The count of accounts you want to create, - default 1");  
        return builder.toString();  
    }  
  
    @Override  
    public String getCommandDescription() {  
        return "create [number] --create account, [number] the number of accounts you want to  
create, - default 1";  
    }  
}
```

@Override

```
public boolean argsValidate(String[] args) {  
    int length = args.length;  
    if (length < 1 || length > 2) {  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
    if (length == 2 && !StringUtils.isNumeric(args[1])) {  
        return false;  
    }  
    if (length == 2 && Integer.parseInt(args[1]) < 1) {  
        return false;  
    }  
    return true;  
}
```

@Override

```
public CommandResult execute(String[] args) {  
    String password = CommandHelper.getPwdOptional();  
    if (StringUtils.isNotBlank(password)) {  
        CommandHelper.confirmPwd(password);  
    }  
    int count = 1;  
    if (args.length == 2) {  
        count = Integer.parseInt(args[1]);  
    }  
    Map<String, Object> parameters = new HashMap<>();  
    parameters.put("password", password);  
    parameters.put("count", count);  
    RpcClientResult result = restFul.post("/account", parameters);  
    if (result.isFailed()) {  
        return CommandResult.getFailed(result);  
    }  
    return CommandResult.getResult(CommandResult.dataTransformList(result));  
}
```

```
rpc\src\main\java\io\nuls\account\rpc\cmd\GetAccountProcessor.java
```

```
*/
```

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.core.tools.date.DateUtil;  
import io.nuls.core.tools.str.StringUtils;  
import io.nuls.kernel.model.CommandResult;  
import io.nuls.kernel.model.RpcClientResult;  
import io.nuls.kernel.processor.CommandProcessor;  
import io.nuls.kernel.utils.CommandBuilder;  
import io.nuls.kernel.utils.CommandHelper;  
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.Date;  
import java.util.Map;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
public class GetAccountProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {  
        return "getaccount";  
    }
```

```
    @Override
```

```
    public String getHelp() {  
        CommandBuilder builder = new CommandBuilder();  
        builder.newLine(getCommandDescription())  
            .newLine("\t<address> the account address - Required");  
        return builder.toString();  
    }
```

```
    @Override
```

```
    public String getCommandDescription() {  
        return "getaccount <address> --get account information";  
    }
```

@Override

```
public boolean argsValidate(String[] args) {  
    if (args.length != 2) {  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
    if (StringUtils.isBlank(args[1])) {  
        return false;  
    }  
    return true;  
}
```

@Override

```
public CommandResult execute(String[] args) {  
    String address = args[1];  
    RpcClientResult result = restFul.get("/account/" + address, null);  
    if (result.isFailed()) {  
        return CommandResult.getFailed(result);  
    }  
    Map<String, Object> map = (Map) result.getData();  
    map.put("createTime", DateUtil.convertDate(new Date((Long) map.get("createTime"))));  
    result.setData(map);  
    return CommandResult.getResult(result);  
}  
}
```

89:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\GetAccountsProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.core.tools.date.DateUtil;  
import io.nuls.kernel.model.RpcClientResult;  
import io.nuls.kernel.utils.CommandBuilder;  
import io.nuls.kernel.utils.CommandHelper;  
import io.nuls.core.tools.str.StringUtils;  
import io.nuls.kernel.model.CommandResult;  
import io.nuls.kernel.processor.CommandProcessor;  
import io.nuls.kernel.utils.RestFulUtils;
```

```

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author: Charlie
 */
public class GetAccountsProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "getaccounts";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<pageNumber>    pageNumber -required")
            .newLine("\t<pageSize>    pageSize -required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getaccounts <pageNumber> <pageSize> --get all account info list int the wallet";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length != 3) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.isNumeric(args[1]) || !StringUtils.isNumeric(args[2])){

```



```

        return false;
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    int pageNumber = Integer.parseInt(args[1]);
    int pageSize = Integer.parseInt(args[2]);
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("pageNumber", pageNumber);
    parameters.put("pageSize", pageSize);
    RpcClientResult result = restFul.get("/account", parameters);
    if(result.isFailed()){
        return CommandResult.getFailed(result);
    }
    List<Map<String, Object>> list = (List<Map<String,
Object>>)((Map)result.getData()).get("list");
    for(Map<String, Object> map : list){
        map.put("createTime", DateUtil.convertDate(new Date((Long)map.get("createTime"))));
    }
    result.setData(list);
    return CommandResult.getResult(result);
}
}

```

90:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\GetAssetProcessor.java  
\*/

```

package io.nuls.account.rpc.cmd;

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

import java.util.List;
import java.util.Map;

```

```

/**
 * @author: Charlie
 */
public class GetAssetProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "getasset";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> address - Required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getasset <address> --get your assets";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length != 2) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if (!AddressTool.validAddress(args[1])) {
            return false;
        }
        return true;
    }

    @Override

```

```

public CommandResult execute(String[] args) {
    String address = args[1];
    RpcClientResult result = restFul.get("/account/assets/" + address, null);
    if(result.isFailed()){
        return CommandResult.getFailed(result);
    }
    List<Map<String, Object>> list = (List<Map<String,
Object>>)((Map)result.getData()).get("list");
    for(Map<String, Object> map : list){
        map.put("balance", CommandHelper.naToNuls(map.get("balance")));
        map.put("usable", CommandHelper.naToNuls(map.get("usable")));
        map.put("locked", CommandHelper.naToNuls(map.get("locked")));
    }
    result.setData(list);
    return CommandResult.getResult(result);
}
}

```

91:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\GetBalanceProcessor.java  
\*/

```

package io.nuls.account.rpc.cmd;

```

```

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

```

```

import java.util.Map;

```

```

/**

```

```

 * @author: Charlie

```

```

 */

```

```

public class GetBalanceProcessor implements CommandProcessor {

```

```

    private RestFulUtils restFul = RestFulUtils.getInstance();

```

```

    @Override

```

```
public String getCommand() {  
    return "getbalance";  
}
```

@Override

```
public String getHelp() {  
    CommandBuilder builder = new CommandBuilder();  
    builder.newLine(getCommandDescription())  
        .newLine("\t<address> the account address - require");  
    return builder.toString();  
}
```

@Override

```
public String getCommandDescription() {  
    return "getbalance <address> --get the balance of a address";  
}
```

@Override

```
public boolean argsValidate(String[] args) {  
    int length = args.length;  
    if(length != 2) {  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
    if(!AddressTool.validAddress(args[1])){  
        return false;  
    }  
    return true;  
}
```

@Override

```
public CommandResult execute(String[] args) {  
    String address = args[1];  
    RpcClientResult result = restFul.get("/accountledger/balance/" + address, null);  
    if(result.isFailed()){  
        return CommandResult.getFailed(result);  
    }  
    Map<String, Object> map = (Map)result.getData();  
    map.put("balance", CommandHelper.naToNuls(((Map)map.get("balance")).get("value")));  
    map.put("usable", CommandHelper.naToNuls(((Map)map.get("usable")).get("value")));  
    map.put("locked", CommandHelper.naToNuls(((Map)map.get("locked")).get("value")));  
}
```

```
        result.setData(map);
        return CommandResult.getResult(result);
    }
}
```

92:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\GetMultiSigAccountCountProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class GetMultiSigAccountCountProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {
        return "getmultiaccountscount";
    }
```

```
    @Override
```

```
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription());
        return builder.toString();
    }
```

```
    @Override
```

```
    public String getCommandDescription() {
```

```
        return "getmultiaccountscount --Get the count of local multi signature accounts";
    }
}
```

@Override

```
public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length != 1) {
        return false;
    }
    return true;
}
```

@Override

```
public CommandResult execute(String[] args) {
    RpcClientResult result = restFul.get("/account/multiAccounts", null);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    List list = (List) result.getData();
    Map<String,Integer> map = new HashMap<>();
    map.put("count",list.size());
    result.setData(map);

    return CommandResult.getResult(result);
}
}
```

```
93:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\cmd\GetMultiSigAccountListProcessor.java
*/
```

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.RestFulUtils;
```

```
/**
```

```
* @author: Niels Wang
```

```

*/
public class GetMultiSigAccountListProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "getmultiaccounts";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription());
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getmultiaccounts --Get all local multi signature accounts";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 1) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        RpcClientResult result = restFul.get("/account/multiAccounts", null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}

```

```
94:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\cmd\GetMultiSigAccountProcessor.java  
*/
```

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.core.tools.str.StringUtils;  
import io.nuls.kernel.model.CommandResult;  
import io.nuls.kernel.model.RpcClientResult;  
import io.nuls.kernel.processor.CommandProcessor;  
import io.nuls.kernel.utils.AddressTool;  
import io.nuls.kernel.utils.CommandBuilder;  
import io.nuls.kernel.utils.CommandHelper;  
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;  
import java.util.Map;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class GetMultiSigAccountProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {  
        return "getmultiaccount";  
    }
```

```
    @Override
```

```
    public String getHelp() {  
        CommandBuilder builder = new CommandBuilder();  
        builder.newLine(getCommandDescription())  
            .newLine("\t[address] Address of multi signature account. -required");  
        return builder.toString();  
    }
```

```
    @Override
```

```
    public String getCommandDescription() {  
        return "getmultiaccount <address> --Obtaining local multi signature account information  
based on address";  
    }
```



```
}
```

```
@Override
```

```
public boolean argsValidate(String[] args) {  
    int length = args.length;  
    if (length != 2) {  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
  
    if (StringUtils.isBlank(args[1])) {  
        return false;  
    }  
    return true;  
}
```

```
@Override
```

```
public CommandResult execute(String[] args) {  
    RpcClientResult result = restFul.get("/account/multiAccount/" + args[1], null);  
    if (result.isFailed()) {  
        return CommandResult.getFailed(result);  
    }  
    return CommandResult.getResult(result);  
}  
}
```

```
95:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\cmd\GetPrivateKeyProcessor.java  
*/
```

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.kernel.model.CommandResult;  
import io.nuls.kernel.model.RpcClientResult;  
import io.nuls.kernel.processor.CommandProcessor;  
import io.nuls.kernel.utils.AddressTool;  
import io.nuls.kernel.utils.CommandBuilder;  
import io.nuls.kernel.utils.CommandHelper;  
import io.nuls.kernel.utils.RestFulUtils;
```

```

import java.util.HashMap;
import java.util.Map;

/**
 * @author: Charlie
 */
public class GetPrivateKeyProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "getprikey";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> address of the account - Required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getprikey <address> --get the private key of your account";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 2) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if (!AddressTool.validAddress(args[1])) {
            return false;
        }
        return true;
    }

```

```

    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        RpcClientResult res = CommandHelper.getPassword(address, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("password", password);
        RpcClientResult result = restFul.post("/account/prikey/" + address, parameters);
        if(result.isFailed()){
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataTransformValue(result));
    }
}

```

96:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\GetWalletBalanceProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

```

```
import java.util.Map;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
public class GetWalletBalanceProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```

@Override
public String getCommand() {
    return "getwalletbalance";
}

```

```

@Override
public String getHelp() {
    CommandBuilder builder = new CommandBuilder();
    builder.newLine(getCommandDescription());
    return builder.toString();
}

```

```

@Override
public String getCommandDescription() {
    return "getwalletbalance --get total balance of all account in the wallet";
}

```

```

@Override
public boolean argsValidate(String[] args) {
    if(args.length > 1) {
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    return true;
}

```

```

@Override
public CommandResult execute(String[] args) {
    RpcClientResult result = restFul.get("/account/balance", null);
    if(result.isFailed()){
        return CommandResult.getFailed(result);
    }
    Map<String, Object> map = (Map)result.getData();
    map.put("balance", CommandHelper.naToNuls(((Map)map.get("balance")).get("value")));
    map.put("usable", CommandHelper.naToNuls(((Map)map.get("usable")).get("value")));
    map.put("locked", CommandHelper.naToNuls(((Map)map.get("locked")).get("value")));
    result.setData(map);
    return CommandResult.getResult(result);
}
}

```

```
97:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\cmd\ImportByKeyStoreProcessor.java  
*/
```

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.account.constant.AccountErrorCode;  
import io.nuls.account.rpc.model.AccountKeyStoreDto;  
import io.nuls.core.tools.json.JSONUtils;  
import io.nuls.core.tools.log.Log;  
import io.nuls.kernel.model.CommandResult;  
import io.nuls.kernel.model.Result;  
import io.nuls.kernel.model.RpcClientResult;  
import io.nuls.kernel.processor.CommandProcessor;  
import io.nuls.kernel.utils.CommandBuilder;  
import io.nuls.kernel.utils.CommandHelper;  
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.io.*;  
import java.net.URLDecoder;  
import java.util.HashMap;  
import java.util.Map;
```

```
/**  
 * keystore,  
 * (keystore), keystore  
 * @author: Charlie  
 */
```

```
public class ImportByKeyStoreProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override  
    public String getCommand() {  
        return "importkeystore";  
    }  
}
```

```
    @Override  
    public String getHelp() {  
        CommandBuilder builder = new CommandBuilder();  
        builder.newLine(getCommandDescription())
```

```

        .newLine("\t<path> The path to the AccountKeystore file ");
    return builder.toString();
}

```

@Override

```

public String getCommandDescription() {
    return "importkeystore <path> -- import accounts according to AccountKeystore files";
}

```

@Override

```

public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length != 2) {
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    return true;
}

```

@Override

```

public CommandResult execute(String[] args) {
    String path = args[1];
    String password = CommandHelper.getPwdOptional();
    Result rs = getAccountKeystoreDto(path);
    if(rs.isFailed()){
        return CommandResult.getFailed(rs.getMsg());
    }
    AccountKeyStoreDto accountKeyStoreDto = (AccountKeyStoreDto)rs.getData();
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("accountKeyStoreDto", accountKeyStoreDto);
    parameters.put("password", password);
    parameters.put("overwrite", false);
    RpcClientResult result = restFul.post("/account/import", parameters);
    if(result.isFailed()){
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(CommandResult.dataTransformValue(result));
}

```

```

/**
 * AccountKeystoreDto
 * Gets the AccountKeystoreDto object based on the file address
 * @param path
 * @return
 */
private Result<AccountKeyStoreDto> getAccountKeystoreDto(String path) {
    File file = null;
    try {
        file = new File(URLDecoder.decode(path, "UTF-8"));
    } catch (UnsupportedEncodingException e) {
        Log.error(e);
    }
    if (null != file && file.isFile()) {
        StringBuilder ks = new StringBuilder();
        BufferedReader bufferedReader = null;
        String str;
        try {
            bufferedReader = new BufferedReader(new FileReader(file));
            while ((str = bufferedReader.readLine()) != null) {
                if (!str.isEmpty()) {
                    ks.append(str);
                }
            }
            AccountKeyStoreDto accountKeyStoreDto = JSONUtils.json2pojo(ks.toString(),
AccountKeyStoreDto.class);
            return Result.getSuccess().setData(accountKeyStoreDto);
        } catch (FileNotFoundException e) {
            return Result.getFailed(AccountErrorCode.ACCOUNTKEYSTORE_FILE_NOT_EXIST);
        } catch (IOException e) {
            return Result.getFailed(AccountErrorCode.ACCOUNTKEYSTORE_FILE_DAMAGED);
        } catch (Exception e) {
            return Result.getFailed(AccountErrorCode.ACCOUNTKEYSTORE_FILE_DAMAGED);
        } finally {
            if (bufferedReader != null) {
                try {
                    bufferedReader.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
        }
    }
}

```

```

    }
    return Result.getFailed(AccountErrorCode.ACCOUNTKEYSTORE_FILE_NOT_EXIST);
}
}

```

98:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\ImportByPrivateKeyProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;
import java.util.Map;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
public class ImportByPrivateKeyProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {
        return "import";
    }

```

```
    @Override
```

```
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<privatekey> private key - Required");
        return builder.toString();
    }

```



```

@Override
public String getCommandDescription() {
    return "import <privatekey> --import the account according to the private key, if the account exists, it will not be executed ";
}

```

```

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length != 2) {
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    return true;
}

```

```

@Override
public CommandResult execute(String[] args) {
    String prikey = args[1];
    String password = CommandHelper.getPwdOptional();
    if (StringUtils.isNotBlank(password)){
        CommandHelper.confirmPwd(password);
    }
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("priKey", prikey);
    parameters.put("password", password);
    parameters.put("overwrite", false);
    RpcClientResult result = restFul.post("/account/import/pri", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(CommandResult.dataTransformValue(result));
}
}

```

```

99:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\cmd\ImportForcedByPrivateKeyProcessor.java
*/

```

```

package io.nuls.account.rpc.cmd;

```

```
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;
import java.util.Map;
```

```
/**
 *
 * Overwrite import
 * @author: Charlie
 */
```

```
public class ImportForcedByPrivateKeyProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {
        return "importforced";
    }
```

```
    @Override
```

```
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<privatekey> private key - Required");
        return builder.toString();
    }
```

```
    @Override
```

```
    public String getCommandDescription() {
        return "importforced <privatekey> --import the account according to the private key, if the account exists, it will be overwritten";
    }
```

```
    @Override
```

```
    public boolean argsValidate(String[] args) {
```

```

int length = args.length;
if (length != 2) {
    return false;
}
if (!CommandHelper.checkArgsIsNull(args)) {
    return false;
}
return true;
}

```

@Override

```

public CommandResult execute(String[] args) {
    String prikey = args[1];
    String password = CommandHelper.getPwdOptional();
    if(StringUtils.isNotBlank(password)){
        CommandHelper.confirmPwd(password);
    }
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("priKey", prikey);
    parameters.put("password", password);
    parameters.put("overwrite", true);
    RpcClientResult result = restFul.post("/account/import/pri", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(CommandResult.dataTransformValue(result));
}
}

```

100:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
 rpc\src\main\java\io\nuls\account\rpc\cmd\ImportMultiSigAccountProcessor.java  
 \*/

```

package io.nuls.account.rpc.cmd;

```

```

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;

```

```

import io.nuls.kernel.utils.RestFulUtils;

import java.util.HashMap;
import java.util.Map;

/**
 * @author: Niels Wang
 */
public class ImportMultiSigAccountProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "importmultiaccount";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t[address] Address of multi signature account. -required")
            .newLine("\t[pks] Multiple public keys separated by '\",\"'. -required")
            .newLine("\t[number] The minimum number of signatures required to initiate a
transaction,0 is all. -required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "importmultiaccount <address> <pks> <m> --create Multi-signature account";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 4) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
    }

```

```

    }

    if (StringUtils.isBlank(args[1])) {
        return false;
    }

    if (!StringUtils.isNumeric(args[3])) {
        return false;
    }

    if (StringUtils.isBlank(args[2])) {
        return false;
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    Map<String, Object> parameters = new HashMap<>();
    String pubkeysStr = args[2];
    parameters.put("address", args[1]);
    parameters.put("pubkeys", pubkeysStr.split(", "));
    parameters.put("m", args[3]);
    RpcClientResult result = restFul.post("/account/importMultiAccount", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(result);
}
}

```

101:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\RemoveAccountProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;

```

```

import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

import java.util.HashMap;
import java.util.Map;

/**
 * @author: Charlie
 */
public class RemoveAccountProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "remove";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> The account address - Required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "remove <address> --remove an account";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 2) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if (!AddressTool.validAddress(args[1])) {
            return false;
        }
    }
}

```

```

    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    String address = args[1];
    RpcClientResult res = CommandHelper.getPassword(address, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("password", password);
    RpcClientResult result = restFul.post("/account/remove/" + address, parameters);
    if(result.isFailed()){
        return CommandResult.getFailed(result);
    }
    return CommandResult.getSuccess("Success");
}
}

```

102:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\RemoveMultiSigAccountProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class RemoveMultiSigAccountProcessor implements CommandProcessor {
```

```
private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
@Override
```

```
public String getCommand() {  
    return "removemultiaccount";  
}
```

```
@Override
```

```
public String getHelp() {  
    CommandBuilder builder = new CommandBuilder();  
    builder.newLine(getCommandDescription())  
        .newLine("\t[address] Address of multi signature account. -required");  
    return builder.toString();  
}
```

```
@Override
```

```
public String getCommandDescription() {  
    return "removemultiaccount <address> --Remove multiple signature accounts by address";  
}
```

```
@Override
```

```
public boolean argsValidate(String[] args) {  
    int length = args.length;  
    if (length != 2) {  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
  
    if (StringUtils.isBlank(args[1]) || !StringUtils.validAddressSimple(args[1])) {  
        return false;  
    }  
    return true;  
}
```

```
@Override
```

```
public CommandResult execute(String[] args) {  
    RpcClientResult result = restFul.delete("/account/multiAccount/" + args[1], null);  
    if (result.isFailed()) {  
        return CommandResult.getFailed(result);  
    }  
}
```



```

    }
    return CommandResult.getResult(result);
}
}

```

103:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\cmd\ResetPasswordProcessor.java  
\*/

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;
import java.util.Map;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
public class ResetPasswordProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {
        return "resetpwd";
    }

```

```
    @Override
```

```
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> address of the account - Required");
        return builder.toString();
    }

```

```
    @Override
```

```

public String getCommandDescription() {
    return "resetpwd <address> --reset password for account";
}

```

@Override

```

public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length != 2) {
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    return true;
}

```

@Override

```

public CommandResult execute(String[] args) {
    String address = args[1];
    RpcClientResult res = CommandHelper.getPassword(address, restFul, "Enter your old
password:");
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    if(res.isSuccess() && null == res.getData()){
        return CommandResult.getFailed("No password has been set up yet");
    }
    String password = (String)res.getData();
    String newPassword = CommandHelper.getNewPwd();
    CommandHelper.confirmPwd(newPassword);
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("password", password);
    parameters.put("newPassword", newPassword);
    RpcClientResult result = restFul.put("/account/password/" + address, parameters);
    if(result.isFailed()){
        return CommandResult.getFailed(result);
    }
    return CommandResult.getSuccess("Success");
}
}

```

```
rpc\src\main\java\io\nuls\account\rpc\cmd\SetAliasProcessor.java
```

```
*/
```

```
package io.nuls.account.rpc.cmd;
```

```
import io.nuls.kernel.model.Address;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;
import java.util.Map;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
public class SetAliasProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {
        return "setalias";
    }
```

```
    @Override
```

```
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> The address of the account, - Required")
            .newLine("\t<alias> The alias of the account, the bytes for the alias is between 1 and 20"
" +
                "(only lower case letters, Numbers and underline, the underline should not be at
the begin and end), - Required");
        return builder.toString();
    }
```

@Override

```
public String getCommandDescription() {  
    return "setalias <address> <alias> --Set an alias for the account ";  
}
```

@Override

```
public boolean argsValidate(String[] args) {  
    int length = args.length;  
    if (length != 3) {  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
    if (!AddressTool.validAddress(args[1])) {  
        return false;  
    }  
    if (!StringUtils.validAlias(args[2])) {  
        return false;  
    }  
  
    return true;  
}
```

@Override

```
public CommandResult execute(String[] args) {  
    String address = args[1];  
    RpcClientResult res = CommandHelper.getPassword(address, restFul);  
    if(!res.isSuccess()){  
        return CommandResult.getFailed(res);  
    }  
    String password = (String)res.getData();  
    Map<String, Object> parameters = new HashMap<>();  
    parameters.put("alias", args[2]);  
    parameters.put("password", password);  
    RpcClientResult result = restFul.post("/account/alias/" + address, parameters);  
    if(result.isFailed()){  
        return CommandResult.getFailed(result);  
    }  
    return CommandResult.getResult(CommandResult.dataTransformValue(result));  
}
```

```

}

105:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\cmd\SetPasswordProcessor.java
*/

package io.nuls.account.rpc.cmd;

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

import java.util.HashMap;
import java.util.Map;

/**
 * @author: Charlie
 */
public class SetPasswordProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "setpwd";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
            .newLine("\t<address> address of the account - Required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "setpwd <address> --set password for the account";
    }

```

```
}
```

```
@Override
```

```
public boolean argsValidate(String[] args) {  
    int length = args.length;  
    if (length != 2) {  
        return false;  
    }  
    if (!CommandHelper.checkArgsIsNull(args)) {  
        return false;  
    }  
    if (!AddressTool.validAddress(args[1])) {  
        return false;  
    }  
    return true;  
}
```

```
@Override
```

```
public CommandResult execute(String[] args) {  
    String address = args[1];  
    RpcClientResult rs = restFul.get("/account/encrypted/" + address, null);  
    if (!rs.isSuccess()) {  
        return CommandResult.getFailed(rs);  
    }  
    if(rs.isSuccess() && rs.dataToBooleanValue()){  
        return CommandResult.getFailed("This account already has a password.");  
    }  
    String password = CommandHelper.getNewPwd();  
    CommandHelper.confirmPwd(password);  
    Map<String, Object> parameters = new HashMap<>();  
    parameters.put("password", password);  
    RpcClientResult result = restFul.post("/account/password/" + address, parameters);  
    if(result.isFailed()){  
        return CommandResult.getFailed(result);  
    }  
    return CommandResult.getSuccess("Success");  
}  
}
```

106:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\model\AccountDto.java

\*/

```
package io.nuls.account.rpc.model;

import io.nuls.account.model.Account;
import io.nuls.core.tools.crypto.Hex;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Charlie
 */
@ApiModel(value = "accountJSON")
public class AccountDto {
    @ApiModelProperty(name = "address", value = "")
    private String address;

    @ApiModelProperty(name = "alias", value = "")
    private String alias;

    @ApiModelProperty(name = "pubKey", value = "Hex.encode(byte[])")
    private String pubKey;

    @ApiModelProperty(name = "priKey", value = "Hex.encode(byte[])")
    private String priKey;

    @ApiModelProperty(name = "encryptedPriKey", value = "Hex.encode(byte[])")
    private String encryptedPriKey;

    @ApiModelProperty(name = "extend", value = "Hex.encode(byte[])")
    private String extend;

    @ApiModelProperty(name = "createTime", value = "")
    private Long createTime;

    @ApiModelProperty(name = "encrypted", value = "")
    private boolean encrypted;

    @ApiModelProperty(name = "remark", value = "")
    private String remark;

    public AccountDto() {
```

```
}
```

```
public AccountDto(Account account) {  
    this.address = account.getAddress().getBase58();  
    this.alias = account.getAlias();  
    this.pubKey = Hex.encode(account.getPubKey());  
    this.createTime = account.getCreateTime();  
    if (account.getExtend() != null) {  
        this.extend = Hex.encode(account.getExtend());  
    }  
    this.encrypted = account.isEncrypted();  
    if (encrypted) {  
        this.encryptedPriKey = Hex.encode(account.getEncryptedPriKey());  
        this.priKey = "";  
    } else {  
        this.priKey = Hex.encode(account.getPriKey());  
        this.encryptedPriKey = "";  
    }  
    this.remark = account.getRemark();  
}
```

```
public String getAddress() {  
    return address;  
}
```

```
public void setAddress(String address) {  
    this.address = address;  
}
```

```
public String getAlias() {  
    return alias;  
}
```

```
public void setAlias(String alias) {  
    this.alias = alias;  
}
```

```
public String getPubKey() {  
    return pubKey;  
}
```



```
public void setPubKey(String pubKey) {  
    this.pubKey = pubKey;  
}
```

```
public String getExtend() {  
    return extend;  
}
```

```
public void setExtend(String extend) {  
    this.extend = extend;  
}
```

```
public Long getCreateTime() {  
    return createTime;  
}
```

```
public void setCreateTime(Long createTime) {  
    this.createTime = createTime;  
}
```

```
public boolean isEncrypted() {  
    return encrypted;  
}
```

```
public void setEncrypted(boolean encrypted) {  
    this.encrypted = encrypted;  
}
```

```
public String getPriKey() {  
    return priKey;  
}
```

```
public void setPriKey(String priKey) {  
    this.priKey = priKey;  
}
```

```
public String getEncryptedPriKey() {  
    return encryptedPriKey;  
}
```

```
public void setEncryptedPriKey(String encryptedPriKey) {  
    this.encryptedPriKey = encryptedPriKey;  
}
```

```

    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }
}

107:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\AccountKeyStoreDto.java
*/

```

```

package io.nuls.account.rpc.model;

import io.nuls.account.model.AccountKeyStore;
import io.nuls.core.tools.crypto.Hex;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.util.Map;

/**
 * @author: Charlie
 */
@ApiModel(value = "JSON")
public class AccountKeyStoreDto {

    @ApiModelProperty(name = "address", value = "")
    private String address;
    @ApiModelProperty(name = "encryptedPrivateKey", value = "")
    private String encryptedPrivateKey;
    @ApiModelProperty(name = "alias", value = "")
    private String alias;
    @ApiModelProperty(name = "pubKey", value = "")
    private String pubKey;
    @ApiModelProperty(name = "prikey", value = "")
    private String prikey;

    public AccountKeyStoreDto() {

```

```

    }

    public AccountKeyStoreDto(AccountKeyStore accountKeyStore) {
        this.address = accountKeyStore.getAddress();
        this.encryptedPrivateKey = null == accountKeyStore.getEncryptedPrivateKey() ? null :
accountKeyStore.getEncryptedPrivateKey();
        this.alias = accountKeyStore.getAlias();
        this.pubKey = Hex.encode(accountKeyStore.getPubKey());
        this.prikey = null == accountKeyStore.getPrikey() ? null :
Hex.encode(accountKeyStore.getPrikey());
    }

    public AccountKeyStore toAccountKeyStore() {
        AccountKeyStore accountKeyStore = new AccountKeyStore();
        accountKeyStore.setAddress(this.address);
        accountKeyStore.setAlias(this.alias);
        accountKeyStore.setEncryptedPrivateKey(this.encryptedPrivateKey);
        if (null == this.prikey || "null".toUpperCase().equals(this.prikey.trim().toUpperCase()) ||
"".equals(prikey.trim())) {
            accountKeyStore.setPrikey(null);
        } else {
            try {
                accountKeyStore.setPrikey(Hex.decode(this.prikey.trim()));
            } catch (Exception e) {
                accountKeyStore.setPrikey(null);
            }
        }
        accountKeyStore.setPubKey(Hex.decode(this.pubKey));
        return accountKeyStore;
    }

    public AccountKeyStoreDto(Map<String, Object> map) {
        this.address = (String) map.get("address");
        this.encryptedPrivateKey = null == map.get("encryptedPrivateKey") ? null : (String)
map.get("encryptedPrivateKey");
        this.alias = null == map.get("alias") ? null : (String) map.get("alias");
        this.pubKey = null == map.get("pubKey") ? null : (String) map.get("pubKey");
        this.prikey = null == map.get("prikey") ? null : (String) map.get("prikey");
    }

    public String getAddress() {

```

```

        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getEncryptedPrivateKey() {
        return encryptedPrivateKey;
    }

    public void setEncryptedPrivateKey(String encryptedPrivateKey) {
        this.encryptedPrivateKey = encryptedPrivateKey;
    }

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = alias;
    }

    public String getPubKey() {
        return pubKey;
    }

    public void setPubKey(String pubKey) {
        this.pubKey = pubKey;
    }

    public String getPrikey() {
        return prikey;
    }

    public void setPrikey(String prikey) {
        this.prikey = prikey;
    }
}

```

108:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\model\AssetDto.java

```
*/
```

```
package io.nuls.account.rpc.model;
```

```
import io.nuls.account.model.Balance;  
import io.nuls.contract.constant.ContractConstant;  
import io.nuls.contract.dto.ContractTokenInfo;  
import io.nuls.contract.util.ContractUtil;  
import io.nuls.kernel.model.Na;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;
```

```
import java.math.BigInteger;
```

```
@ApiModel(value = "assetJSON")
```

```
public class AssetDto {
```

```
    @ApiModelProperty(name = "asset", value = "")  
    private String asset;
```

```
    @ApiModelProperty(name = "address", value = "")  
    private String address;
```

```
    @ApiModelProperty(name = "balance", value = "")  
    private String balance;
```

```
    @ApiModelProperty(name = "usable", value = "")  
    private String usable;
```

```
    @ApiModelProperty(name = "locked", value = "")  
    private String locked;
```

```
    @ApiModelProperty(name = "decimals", value = "")  
    private long decimals;
```

```
    @ApiModelProperty(name = "status", value = "")  
    private int status;
```

```
    public AssetDto(String asset, Balance balance) {  
        this.balance = BigInteger.valueOf(balance.getBalance().getValue()).toString();  
        this.usable = BigInteger.valueOf(balance.getUsable().getValue()).toString();  
        this.locked = BigInteger.valueOf(balance.getLocked().getValue()).toString();  
    }
```

```
    this.asset = asset;
    this.decimals = Na.SMALLEST_UNIT_EXPONENT;
    this.status = ContractConstant.NORMAL;
}
```

```
public AssetDto(ContractTokenInfo tokenInfo) {
    this.balance = ContractUtil.bigInteger2String(tokenInfo.getAmount());
    this.usable = this.balance;
    this.locked = BigInteger.ZERO.toString();
    this.asset = tokenInfo.getSymbol();
    this.address = tokenInfo.getContractAddress();
    this.decimals = tokenInfo.getDecimals();
    this.status = tokenInfo.getStatus();
}
```

```
public String getAsset() {
    return asset;
}
```

```
public void setAsset(String asset) {
    this.asset = asset;
}
```

```
public String getAddress() {
    return address;
}
```

```
public void setAddress(String address) {
    this.address = address;
}
```

```
public String getBalance() {
    return balance;
}
```

```
public void setBalance(String balance) {
    this.balance = balance;
}
```

```
public String getUsable() {
    return usable;
}
```

```

    public void setUsable(String usable) {
        this.usable = usable;
    }

    public String getLocked() {
        return locked;
    }

    public void setLocked(String locked) {
        this.locked = locked;
    }

    public long getDecimals() {
        return decimals;
    }

    public void setDecimals(long decimals) {
        this.decimals = decimals;
    }

    public int getStatus() {
        return status;
    }

    public void setStatus(int status) {
        this.status = status;
    }
}

109:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\BalanceDto.java
*/

package io.nuls.account.rpc.model;

import io.nuls.account.model.Balance;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "balanceJSON")
public class BalanceDto {

```

```
@ApiModelProperty(name = "balance", value = "")
private long balance;
```

```
@ApiModelProperty(name = "usable", value = "")
private long usable;
```

```
@ApiModelProperty(name = "locked", value = "")
private long locked;
```

```
public BalanceDto(Balance balance) {
    if (balance == null) {
        this.balance = 0;
        this.usable = 0;
        this.locked = 0;
    } else {
        this.balance = balance.getBalance().getValue();
        this.usable = balance.getUsable().getValue();
        this.locked = balance.getLocked().getValue();
    }
}
```

```
public long getBalance() {
    return balance;
}
```

```
public void setBalance(long balance) {
    this.balance = balance;
}
```

```
public long getUsable() {
    return usable;
}
```

```
public void setUsable(long usable) {
    this.usable = usable;
}
```

```
public long getLocked() {
    return locked;
}
```



```
    public void setLocked(long locked) {  
        this.locked = locked;  
    }  
}
```

110:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\model\form\AccountAliasFeeForm.java  
\*/

```
package io.nuls.account.rpc.model.form;
```

```
import io.nuls.core.tools.str.StringUtils;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;
```

```
import javax.ws.rs.QueryParam;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
@ApiModel(value = "")
```

```
public class AccountAliasFeeForm {
```

```
    @ApiModelProperty(name = "address", value = "", required = true)
```

```
    @QueryParam("address")
```

```
    private String address;
```

```
    @ApiModelProperty(name = "alias", value = "", required = true)
```

```
    @QueryParam("alias")
```

```
    private String alias;
```

```
    public String getAddress() {
```

```
        return address;
```

```
    }
```

```
    public void setAddress(String address) {
```

```
        this.address = address;
```

```
    }
```

```
    public String getAlias() {
```

```
        return alias;
```

```

    }

    public void setAlias(String alias) {
        this.alias = StringUtils.formatStringPara(alias);
    }
}

111:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountAliasForm.java
*/

package io.nuls.account.rpc.model.form;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Charlie
 */
@ApiModel(value = "")
public class AccountAliasForm {

    @ApiModelProperty(name = "alias", value = "", required = true)
    private String alias;

    @ApiModelProperty(name = "password", value = "", required = true)
    private String password;

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = StringUtils.formatStringPara(alias);
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {

```

```
        this.password = password;
    }
}
```

112:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\model\form\AccountCreateForm.java  
\*/

```
package io.nuls.account.rpc.model.form;
```

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
@ApiModel(value = "")
```

```
public class AccountCreateForm {
```

```
    @ApiModelProperty(name = "count", value = "")
```

```
    private int count;
```

```
    @ApiModelProperty(name = "password", value = "")
```

```
    private String password;
```

```
    public int getCount() {
```

```
        return count;
```

```
    }
```

```
    public void setCount(int count) {
```

```
        this.count = count;
```

```
    }
```

```
    public String getPassword() {
```

```
        return password;
```

```
    }
```

```
    public void setPassword(String password) {
```

```
        this.password = password;
```

```
    }
```

```
}
```

```
113:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountKeyStoreImportForm.java  
*/
```

```
package io.nuls.account.rpc.model.form;
```

```
import io.nuls.account.rpc.model.AccountKeyStoreDto;  
import io.nuls.core.tools.str.StringUtils;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
@ApiModel(value = "KeyStore")
```

```
public class AccountKeyStoreImportForm {
```

```
    @ApiModelProperty(name = "accountKeyStoreDto", value = "", required = true)  
    private AccountKeyStoreDto accountKeyStoreDto;
```

```
    @ApiModelProperty(name = "password", value = "")  
    private String password;
```

```
    @ApiModelProperty(name = "overwrite", value = ": false:, true:")  
    private Boolean overwrite = false;
```

```
    public AccountKeyStoreDto getAccountKeyStoreDto() {  
        return accountKeyStoreDto;  
    }
```

```
    public void setAccountKeyStoreDto(AccountKeyStoreDto accountKeyStoreDto) {  
        this.accountKeyStoreDto = accountKeyStoreDto;  
    }
```

```
    public String getPassword() {  
        return password;  
    }
```

```
    public void setPassword(String password) {  
        this.password = password;  
    }
```

```

    public Boolean getOverwrite() {
        return overwrite;
    }

    public void setOverwrite(Boolean overwrite) {
        this.overwrite = overwrite;
    }
}

114:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountKeyStoreResetPasswordForm.java
*/

package io.nuls.account.rpc.model.form;

import io.nuls.account.rpc.model.AccountKeyStoreDto;
import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Charlie
 */
@ApiModel(value = "KeyStore")
public class AccountKeyStoreResetPasswordForm {
    @ApiModelProperty(name = "accountKeyStoreDto", value = "", required = true)
    private AccountKeyStoreDto accountKeyStoreDto;

    @ApiModelProperty(name = "password", value = "")
    private String password;

    public AccountKeyStoreDto getAccountKeyStoreDto() {
        return accountKeyStoreDto;
    }

    public void setAccountKeyStoreDto(AccountKeyStoreDto accountKeyStoreDto) {
        this.accountKeyStoreDto = accountKeyStoreDto;
    }

    public String getPassword() {
        return password;
    }
}

```

```
    public void setPassword(String password) {  
        this.password = password;  
    }  
}
```

```
115:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountParamForm.java  
*  
*/
```

```
package io.nuls.account.rpc.model.form;
```

```
import io.nuls.core.tools.str.StringUtils;
```

```
/**
```

```
 * @author Niels
```

```
 */
```

```
@Deprecated
```

```
public class AccountParamForm {
```

```
    private String address;
```

```
    private String password;
```

```
    private String alias;
```

```
    private int count;
```

```
    private String prikey;
```

```
    private String newPassword;
```

```
    public String getPassword() {  
        return password;  
    }
```

```
    public void setPassword(String password) {  
        this.password = password;  
    }
```

```
    public int getCount() {
```

```

        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = StringUtils.formatStringPara(address);
    }

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = StringUtils.formatStringPara(alias);
    }

    public String getPrikey() {
        return prikey;
    }

    public void setPrikey(String prikey) {
        this.prikey = StringUtils.formatStringPara(prikey);
    }

    public String getNewPassword() {
        return newPassword;
    }

    public void setNewPassword(String newPassword) {
        this.newPassword = newPassword;
    }
}

```

116:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\model\form\AccountPasswordForm.java

```
*/
```

```
package io.nuls.account.rpc.model.form;
```

```
import io.swagger.annotations.ApiModel;
```

```
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
@ApiModel(value = "")
```

```
public class AccountPasswordForm {
```

```
    @ApiModelProperty(name = "password", value = "", required = true)
```

```
    private String password;
```

```
    public String getPassword() {
```

```
        return password;
```

```
    }
```

```
    public void setPassword(String password) {
```

```
        this.password = password;
```

```
    }
```

```
}
```

```
117:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountPriKeyChangePasswordForm.java
```

```
*/
```

```
package io.nuls.account.rpc.model.form;
```

```
import io.swagger.annotations.ApiModel;
```

```
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
@ApiModel(value = "")
```

```
public class AccountPriKeyChangePasswordForm {
```

```
    @ApiModelProperty(name = "priKey", value = "", required = true)
```



```

private String priKey;

@ApiModelProperty(name = "password", value = "")
private String password;

public String getPriKey() {
    return priKey;
}

public void setPriKey(String priKey) {
    this.priKey = priKey;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}
}

118:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountPriKeyPasswordForm.java
*/

package io.nuls.account.rpc.model.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Charlie
 */
@ApiModel(value = "")
public class AccountPriKeyPasswordForm {

    @ApiModelProperty(name = "priKey", value = "", required = true)
    private String priKey;

    @ApiModelProperty(name = "password", value = "")

```

```

private String password;

@ApiModelProperty(name = "overwrite", value = ": false:, true:")
private Boolean overwrite = false;

public String getPriKey() {
    return priKey;
}

public void setPriKey(String priKey) {
    this.priKey = priKey;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public Boolean getOverwrite() {
    return overwrite;
}

public void setOverwrite(Boolean overwrite) {
    this.overwrite = overwrite;
}
}

119:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountPriKeysPasswordForm.java
*/
package io.nuls.account.rpc.model.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.util.List;

/**
 * @author: Charlie

```

```

* @date: 2018/7/21
*/
@ApiModel(value = "")
public class AccountPriKeysPasswordForm {

    @ApiModelProperty(name = "priKey", value = "", required = true)
    private List<String> priKey;

    @ApiModelProperty(name = "password", value = "")
    private String password;

    public List<String> getPriKey() {
        return priKey;
    }

    public void setPriKey(List<String> priKey) {
        this.priKey = priKey;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

}

120:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountRemarkForm.java
*/
package io.nuls.account.rpc.model.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import javax.ws.rs.FormParam;

/**
* @author: Charlie
* @date: 2018/7/27

```

```

*/
@ApiModel(value = "")
public class AccountRemarkForm {

    @ApiModelProperty(name = "remark", value = "")
    @FormParam("remark")
    private String remark;

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }

}

```

121:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\model\form\AccountUnlockForm.java

```

*/

```

```

package io.nuls.account.rpc.model.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Charlie
 */
@ApiModel(value = "")
public class AccountUnlockForm {

    @ApiModelProperty(name = "password", value = "", required = true)
    private String password;

    @ApiModelProperty(name = "unlockTime", value = "", required = true)
    private Integer unlockTime;

    public String getPassword() {
        return password;
    }

}

```

```

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getUnlockTime() {
        return unlockTime;
    }

    public void setUnlockTime(Integer unlockTime) {
        this.unlockTime = unlockTime;
    }
}

122:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\AccountUpdatePasswordForm.java
*/

package io.nuls.account.rpc.model.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Charlie
 */
@ApiModel(value = "")
public class AccountUpdatePasswordForm {

    @ApiModelProperty(name = "password", value = "", required = true)
    private String password;

    @ApiModelProperty(name = "newPassword", value = "", required = true)
    private String newPassword;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

```

```

    public String getNewPassword() {
        return newPassword;
    }

    public void setNewPassword(String newPassword) {
        this.newPassword = newPassword;
    }
}

123:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\CreateMultiAliasForm.java
*/
package io.nuls.account.rpc.model.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: tag
 */
@ApiModel(value = "")
public class CreateMultiAliasForm {
    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;

    @ApiModelProperty(name = "alias", value = "", required = true)
    private String alias;

    @ApiModelProperty(name = "password", value = "", required = true)
    private String password;

    @ApiModelProperty(name = "signAddress", value = "", required = true)
    private String signAddress;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}

```

```

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = alias;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getSignAddress() {
        return signAddress;
    }

    public void setSignAddress(String signAddress) {
        this.signAddress = signAddress;
    }
}

```

```

124:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\MultiAccountAliasFeeForm.java
*/

```

```

package io.nuls.account.rpc.model.form;

```

```

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

```

```

import javax.ws.rs.QueryParam;
import java.util.List;

```

```

/**
 * @author: tag
 */
@ApiModel(value = "")

```

```
public class MultiAccountAliasFeeForm {
    @ApiModelProperty(name = "address", value = "", required = true)
    @QueryParam("address")
    private String address;

    @ApiModelProperty(name = "alias", value = "", required = true)
    @QueryParam("alias")
    private String alias;

    @ApiModelProperty(name = "pubkeys", value = "", required = true)
    private List<String> pubkeys;

    @ApiModelProperty(name = "m", value = "", required = true)
    private int m;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = StringUtils.formatStringPara(alias);
    }

    public List<String> getPubkeys() {
        return pubkeys;
    }

    public void setPubkeys(List<String> pubkeys) {
        this.pubkeys = pubkeys;
    }

    public int getM() {
        return m;
    }
}
```



```
    public void setM(int m) {  
        this.m = m;  
    }  
}
```

```
125:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
rpc\src\main\java\io\nuls\account\rpc\model\form\MultiAccountCreateForm.java  
*/
```

```
package io.nuls.account.rpc.model.form;
```

```
import io.nuls.core.tools.str.StringUtils;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;
```

```
import javax.ws.rs.QueryParam;  
import java.util.List;
```

```
/**
```

```
 * @author: tangag
```

```
 */
```

```
@ApiModel(value = "")
```

```
public class MultiAccountCreateForm {
```

```
    @ApiModelProperty(name = "pubkeys", value = "", required = true)
```

```
    private List<String> pubkeys;
```

```
    @ApiModelProperty(name = "m", value = "", required = true)
```

```
    private int m;
```

```
    public List<String> getPubkeys() {
```

```
        return pubkeys;
```

```
    }
```

```
    public void setPubkeys(List<String> pubkeys) {
```

```
        this.pubkeys = pubkeys;
```

```
    }
```

```
    public int getM() {
```

```
        return m;
```

```
    }
```

```
    public void setM(int m) {
```

```
        this.m = m;
    }
}
```

```
126:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\MultiAccountImportForm.java
*/
```

```
package io.nuls.account.rpc.model.form;
```

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
import java.util.List;
```

```
/**
```

```
 * @author: Niels
```

```
 */
```

```
@ApiModel(value = "")
```

```
public class MultiAccountImportForm {
```

```
    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;
```

```
    @ApiModelProperty(name = "pubkeys", value = "", required = true)
    private List<String> pubkeys;
```

```
    @ApiModelProperty(name = "m", value = "", required = true)
    private int m;
```

```
    public String getAddress() {
        return address;
    }
```

```
    public void setAddress(String address) {
        this.address = address;
    }
```

```
    public List<String> getPubkeys() {
        return pubkeys;
    }
```

```
    public void setPubkeys(List<String> pubkeys) {
```

```

        this.pubkeys = pubkeys;
    }

    public int getM() {
        return m;
    }

    public void setM(int m) {
        this.m = m;
    }
}

```

```

127:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\MultiAliasFeeForm.java
*/

```

```

package io.nuls.account.rpc.model.form;

```

```

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

```

```

import javax.ws.rs.QueryParam;
import java.util.List;

```

```

/**

```

```

 * @author: tangag

```

```

 */

```

```

@ApiModel(value = "")

```

```

public class MultiAliasFeeForm {

```

```

    @ApiModelProperty(name = "address", value = "", required = true)

```

```

    @QueryParam("address")

```

```

    private String address;

```

```

    @ApiModelProperty(name = "alias", value = "", required = true)

```

```

    @QueryParam("alias")

```

```

    private String alias;

```

```

    public String getAddress() {

```

```

        return address;
    }

```

```

    public void setAddress(String address) {

```

```

        this.address = address;
    }

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = StringUtils.formatStringPara(alias);
    }
}

```

128:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\model\form\MultiTransactionSignForm.java  
 \*/

```
package io.nuls.account.rpc.model.form;
```

```
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
 * @author tag
```

```
 */
```

```
@ApiModel(value = "form")
```

```
public class MultiTransactionSignForm {
```

```
    @ApiModelProperty(name = "signAddress", value = "", required = true)
```

```
    private String signAddress;
```

```
    @ApiModelProperty(name = "password", value = "", required = false)
```

```
    private String password;
```

```
    @ApiModelProperty(name = "txdata", value = "")
```

```
    private String txdata;
```

```
    public String getSignAddress() {
```

```
        return signAddress;
```

```
    }
```

```
    public void setSignAddress(String signAddress) {
```

```
        this.signAddress = signAddress;
```

```
    }
```

```

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getTxdata() {
    return txdata;
}

public void setTxdata(String txdata) {
    this.txdata = txdata;
}
}

129:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\form\OfflineAccountPasswordForm.java
*/

package io.nuls.account.rpc.model.form;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Charlie
 */
@ApiModel(value = "")
public class OfflineAccountPasswordForm {

    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;

    @ApiModelProperty(name = "priKey", value = "", required = true)
    private String priKey;

    @ApiModelProperty(name = "password", value = "", required = true)
    private String password;

    @ApiModelProperty(name = "newPassword", value = "", required = true)

```

```

private String newPassword;

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getPriKey() {
    return priKey;
}

public void setPriKey(String priKey) {
    this.priKey = priKey;
}

public String getNewPassword() {
    return newPassword;
}

public void setNewPassword(String newPassword) {
    this.newPassword = newPassword;
}
}

130:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
rpc\src\main\java\io\nuls\account\rpc\model\MultiSigAccountDto.java
*/

package io.nuls.account.rpc.model;

import io.nuls.account.model.MultiSigAccount;

```

```

import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author: Charlie
 */
@ApiModel(value = "multiSigAccountJSON")
public class MultiSigAccountDto {

    @ApiModelProperty(name = "address", value = "")
    private String address;

    @ApiModelProperty(name = "m", value = "")
    private long m;

    @ApiModelProperty(name = "alias", value = "")
    private String alias;

    @ApiModelProperty(name = "pubkeys", value = "")
    private List<Map<String, String>> pubkeys;

    public MultiSigAccountDto() {

    }

    public MultiSigAccountDto(MultiSigAccount account) {
        this.address = account.getAddress().getBase58();
        this.pubkeys = new ArrayList<>();
        for (byte[] bytes : account.getPubKeyList()) {
            Map<String, String> map = new HashMap<>();
            map.put("pubkey", Hex.encode(bytes));
            map.put("address",
AddressTool.getStringAddressByBytes(AddressTool.getAddress(bytes)));
            pubkeys.add(map);
        }
    }

```

```

        this.m = account.getM();
        this.alias = account.getAlias();
        if (null == alias) {
            this.alias = "";
        }
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public long getM() {
        return m;
    }

    public void setM(long m) {
        this.m = m;
    }

    public List<Map<String, String>> getPubkeys() {
        return pubkeys;
    }

    public void setPubkeys(List<Map<String, String>> pubkeys) {
        this.pubkeys = pubkeys;
    }

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = alias;
    }
}

```

131:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-rpc\src\main\java\io\nuls\account\rpc\resource\AccountResource.java



\*/

```
package io.nuls.account.rpc.resource;

import io.nuls.account.constant.AccountConstant;
import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.model.CoinDataResult;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.*;
import io.nuls.account.rpc.model.AccountDto;
import io.nuls.account.rpc.model.AccountKeyStoreDto;
import io.nuls.account.rpc.model.AssetDto;
import io.nuls.account.rpc.model.MultiSigAccountDto;
import io.nuls.account.rpc.model.form.*;
import io.nuls.account.service.AccountBaseService;
import io.nuls.account.service.AccountCacheService;
import io.nuls.account.service.AccountService;
import io.nuls.account.service.AliasService;
import io.nuls.account.tx.AliasTransaction;
import io.nuls.account.util.AccountTool;
import io.nuls.contract.dto.ContractTokenInfo;
import io.nuls.contract.service.ContractService;
import io.nuls.core.tools.crypto.AESEncrypt;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.page.Page;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.*;
import io.nuls.kernel.script.Script;
import io.nuls.kernel.script.ScriptBuilder;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.kernel.utils.TransactionFeeCalculator;
```

```

import io.nuls.ledger.constant.LedgerErrorCode;
import io.nuls.protocol.model.validator.TxMaxSizeValidator;
import io.swagger.annotations.*;
import org.glassfish.jersey.media.multipart.FormDataParam;

import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.*;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import java.io.*;
import java.math.BigInteger;
import java.util.*;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

/**
 * @author: Charlie
 */
@Path("/account")
@Api(value = "account", description = "account")
@Component
public class AccountResource {

    @Autowired
    private AccountService accountService;

    @Autowired
    private AliasService aliasService;

    @Autowired
    private AccountBaseService accountBaseService;

    @Autowired
    private AccountLedgerService accountLedgerService;

    @Autowired
    private ContractService contractService;

    private AccountCacheService accountCacheService = AccountCacheService.getInstance();

```

```
private ScheduledThreadPoolExecutor scheduler = new ScheduledThreadPoolExecutor(1);
```

```
private Map<String, ScheduledFuture> accountUnlockSchedulerMap = new HashMap<>();
```

```
@POST
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
@ApiOperation(value = "[ ] ", notes = "result.data: List<String>")
```

```
@ApiResponses(value = {
```

```
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
```

```
})
```

```
public RpcClientResult create(@ApiParam(name = "form", value = "", required = true)
```

```
    AccountCreateForm form) {
```

```
    int count = form.getCount() < 1 ? 1 : form.getCount();
```

```
    String password = form.getPassword();
```

```
    if (StringUtils.isBlank(password)) {
```

```
        password = null;
```

```
    }
```

```
    Result result = accountService.createAccount(count, password);
```

```
    if (result.isFailed()) {
```

```
        return result.toRpcClientResult();
```

```
    }
```

```
    List<Account> listAccount = (List<Account>) result.getData();
```

```
    List<String> list = new ArrayList<>();
```

```
    for (Account account : listAccount) {
```

```
        list.add(account.getAddress().toString());
```

```
    }
```

```
    Map<String, List<String>> map = new HashMap<>();
```

```
    map.put("list", list);
```

```
    return Result.getSuccess().setData(map).toRpcClientResult();
```

```
}
```

```
@POST
```

```
@Path("/offline")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
@ApiOperation(value = "[ ] , , ", notes = "result.data: List<Account>")
```

```
@ApiResponses(value = {
```

```
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
```

```
})
```

```
public RpcClientResult createOfflineAccount(@ApiParam(name = "form", value = "", required = true)
```

```
    AccountCreateForm form) {
```

```
    int count = form.getCount() < 1 ? 1 : form.getCount();
```

```

    if (count <= 0 || count > AccountTool.CREATE_MAX_SIZE) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    String password = form.getPassword();
    if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }
    List<AccountDto> accounts = new ArrayList<>();
    try {
        for (int i = 0; i < count; i++) {
            Account account = AccountTool.createAccount();
            if (StringUtils.isNotBlank(password)) {
                account.encrypt(password);
            }
            accounts.add(new AccountDto(account));
        }
    } catch (NulsException e) {
        return Result.getFailed().toRpcClientResult();
    }
    Map<String, List<AccountDto>> map = new HashMap<>();
    map.put("list", accounts);
    return Result.getSuccess().setData(map).toRpcClientResult();
}

```

@GET

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "[ ] ", notes = "result.data: Page<AccountDto>")

@ApiResponses(value = {

@ApiResponse(code = 200, message = "success", response = RpcClientResult.class)

})

public RpcClientResult accountList(@ApiParam(name = "pageNumber", value = "")

@QueryParam("pageNumber") int pageNumber,

@ApiParam(name = "pageSize", value = "")

@QueryParam("pageSize") int pageSize) {

if (pageNumber < 0 || pageSize < 0) {

return Result.getFailed(AccountErrorCode.PARAMETER\_ERROR).toRpcClientResult();

}

if (pageNumber == 0) {

pageNumber = 1;

}

if (pageSize == 0) {

```

        pageSize = 100;
    }
    Collection<Account> accounts = accountService.getAccountList().getData();
    List<Account> accountList = new ArrayList<>(accounts);
    Page<Account> page = new Page<>(pageNumber, pageSize);
    page.setTotal(accountList.size());
    int start = (pageNumber - 1) * pageSize;
    if (start >= accountList.size()) {
        return Result.getSuccess().setData(page).toRpcClientResult();
    }
    int end = pageNumber * pageSize;
    if (end > accountList.size()) {
        end = accountList.size();
    }
    accountList = accountList.subList(start, end);
    Page<AccountDto> resultPage = new Page<>(page);
    List<AccountDto> dtoList = new ArrayList<>();
    for (Account account : accountList) {
        dtoList.add(new AccountDto(account));
    }
    resultPage.setList(dtoList);
    return Result.getSuccess().setData(resultPage).toRpcClientResult();
}

```

```

@GET
@Path("/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult get(@ApiParam(name = "address", value = "", required = true)
    @PathParam("address") String address) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Account account = accountService.getAccount(address).getData();
    if (null == account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
    }
}

```

```

        return Result.getSuccess().setData(new AccountDto(account)).toRpcClientResult();
    }

    @GET
    @Path("/encrypted/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("[] ")
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult isEncrypted(@ApiParam(name = "address", value = "", required = true)
        @PathParam("address") String address) {
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        Result result = accountService.isEncrypted(address);
        Map<String, Boolean> map = new HashMap<>();
        map.put("value", (Boolean) result.getData());
        result.setData(map);
        return result.toRpcClientResult();
    }

    @POST
    @Path("/password/validation/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("[] ")
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult validationPassword(@PathParam("address") String address,
        @ApiParam(name = "form", value = "", required = true)
        AccountPasswordForm form) {
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        if (StringUtils.isBlank(form.getPassword())) {
            return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        Result<Account> rs = accountService.getAccount(address);
        if (rs.isFailed() || null == rs.getData()) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
    }

```

```

Account account = rs.getData();
boolean result = account.validatePassword(form.getPassword());
Map<String, Boolean> map = new HashMap<>(2);
map.put("value", result);
return Result.getSuccess().setData(map).toRpcClientResult();
}

@POST
@Path("/alias/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult alias(@PathParam("address") String address,
    @ApiParam(name = "form", value = "", required = true)
    AccountAliasForm form) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(form.getAlias())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Result result = aliasService.setAlias(address, form.getAlias().trim(), form.getPassword());
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("value", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

@GET
@Path("/alias/fee")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult aliasFee(@BeanParam() AccountAliasFeeForm form) {
    if (!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }

```

```

    }
    if (StringUtils.isBlank(form.getAlias())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Result result = aliasService.getAliasFee(form.getAddress(), form.getAlias());

    Long fee = null;
    Long maxAmount = null;
    Map<String, Long> map = new HashMap<>();
    if (result.isSuccess()) {
        fee = ((Na) result.getData()).getValue();
        //
        long feeMax =
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES.multiply(TxMaxSizeValidator.MA
X_TX_BYTES).getValue();
        if(fee > feeMax){
            AliasTransaction tx = new AliasTransaction();
            tx.setTime(TimeService.currentTimeMillis());
            Alias alias = new Alias(AddressTool.getAddress(form.getAddress()), form.getAlias());
            tx.setTxData(alias);
            try {
                CoinDataResult coinDataResult =
accountLedgerService.getCoinData(AddressTool.getAddress(form.getAddress()),
AccountConstant.ALIAS_NA, tx.size(),
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
                if (!coinDataResult.isEnough()) {
                    return
Result.getFailed(AccountErrorCode.INSUFFICIENT_BALANCE).toRpcClientResult();
                }
                CoinData coinData = new CoinData();
                coinData.setFrom(coinDataResult.getCoinList());
                Coin change = coinDataResult.getChange();
                if (null != change) {
                    //toList
                    List<Coin> toList = new ArrayList<>();
                    toList.add(change);
                    coinData.setTo(toList);
                }
                Coin coin = new Coin(NulsConstant.BLACK_HOLE_ADDRESS, Na.parseNuls(1), 0);
                coinData.addTo(coin);
                tx.setCoinData(coinData);
            } catch (Exception e) {

```



```

        Log.error(e);
        return
Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
    }
    Result rs =
accountLedgerService.getMaxAmountOfOnce(AddressTool.getAddress(form.getAddress()), tx,
        TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
    if (rs.isSuccess()) {
        maxAmount = ((Na) rs.getData()).getValue();
    }
}
map.put("fee", fee);
map.put("maxAmount", maxAmount);
result.setData(map);
}
return result.toRpcClientResult();
}

```

```

@GET
@Path("/alias")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[ ] ( ) ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult isAliasUsable(@ApiParam(name = "alias", value = "", required = true)
@QueryParam("alias") String alias) {
    if (StringUtils.isBlank(alias)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Map<String, Boolean> map = new HashMap<>();
    map.put("value", aliasService.isAliasUsable(alias));
    return Result.getSuccess().setData(map).toRpcClientResult();
}

```

```

@GET
@Path("/alias/address")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})

```

```

public RpcClientResult getAddressByAlias(@ApiParam(name = "alias", value = "", required =
true) @QueryParam("alias") String alias) {
    if (StringUtils.isBlank(alias)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Alias aliasObj = aliasService.getAlias(alias);
    if (null == aliasObj) {
        return new RpcClientResult(false, AccountErrorCode.ALIAS_NOT_EXIST);
    }
    Map<String, String> map = new HashMap<>();
    map.put("value", AddressTool.getStringAddressByBytes(aliasObj.getAddress()));
    return Result.getSuccess().setData(map).toRpcClientResult();
}

```

```

@GET
@Path("/balance")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("{} ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult getTotalBalance() {
    try {
        return accountService.getBalance().toRpcClientResult();
    } catch (NulsException e) {
        return Result.getFailed(AccountErrorCode.FAILED).toRpcClientResult();
    }
}

```

```

@GET
@Path("/assets/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "{} [3.3.8]", notes = "result.data: List<AssetDto>")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult getAssets(@ApiParam(name = "address", value = "", required = true)
    @PathParam("address") String address,
    @ApiParam(name = "pageNumber", value = "", required = true)
    @QueryParam("pageNumber") Integer pageNumber,
    @ApiParam(name = "pageSize", value = "", required = false)
    @QueryParam("pageSize") Integer pageSize) {

```

```

try {
    if (null == pageNumber || pageNumber == 0) {
        pageNumber = 1;
    }
    if (null == pageSize || pageSize == 0) {
        pageSize = 10;
    }
    if (pageNumber < 0 || pageSize < 0 || pageSize > 100) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Address addr = new Address(address);
    Result<Balance> balanceResult =
accountLedgerService.getBalance(addr.getAddressBytes());
    if (balanceResult.isFailed()) {
        return balanceResult.toRpcClientResult();
    }
    Balance balance = balanceResult.getData();
    List<AssetDto> dtoList = new ArrayList<>();
    dtoList.add(new AssetDto("NULS", balance));

    Result<List<ContractTokenInfo>> allTokenListResult =
contractService.getAllTokensByAccount(address);
    if (allTokenListResult.isSuccess()) {
        List<ContractTokenInfo> tokenInfoList = allTokenListResult.getData();
        if (tokenInfoList != null && tokenInfoList.size() > 0) {
            for (ContractTokenInfo tokenInfo : tokenInfoList) {
                if (tokenInfo.isLock()) {
                    continue;
                }
                dtoList.add(new AssetDto(tokenInfo));
            }
        }
    }

    Result result = Result.getSuccess();
    List<AssetDto> infoDtoList = new ArrayList<>();
    Page<AssetDto> page = new Page<>(pageNumber, pageSize, dtoList.size());
    int start = pageNumber * pageSize - pageSize;
    if (start >= page.getTotal()) {

```

```

        result.setData(page);
        return result.toRpcClientResult();
    }

```

```

    int end = start + pageSize;
    if (end > page.getTotal()) {
        end = (int) page.getTotal();
    }

```

```

    if (dtoList.size() > 0) {
        for (int i = start; i < end; i++) {
            infoDtoList.add(dtoList.get(i));
        }
    }

```

```

    page.setList(infoDtoList);

```

```

    result.setSuccess(true);
    result.setData(page);

```

```

        return result.toRpcClientResult();
    } catch (Exception e) {
        Log.error(e);
        Result result = Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION);
        return result.toRpcClientResult();
    }

```

```

}

```

```

@POST

```

```

@Path("/prikey/{address}")

```

```

@Produces(MediaType.APPLICATION_JSON)

```

```

@ApiOperation("[ ] ")

```

```

@ApiResponses(value = {

```

```

    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)

```

```

})

```

```

public RpcClientResult getPrikey(@PathParam("address") String address, @ApiParam(name =
"form", value = "", required = true)

```

```

    AccountPasswordForm form) {

```

```

    if (!AddressTool.validAddress(address)) {

```

```

        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }

```

```

}

```

```

    Result result = accountBaseService.getPrivateKey(address, form.getPassword());

```

```

        if (result.isSuccess()) {
            Map<String, String> map = new HashMap<>();
            map.put("value", (String) result.getData());
            result.setData(map);
        }
        return result.toRpcClientResult();
    }

    @POST
    @Path("/prikey")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("[]")
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult getAllPrikey(@ApiParam(name = "form", value = "")
AccountPasswordForm form) {
        String password = form.getPassword();
        if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
            return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
        }
        Result result = accountBaseService.getAllPrivateKey(form.getPassword());
        return result.toRpcClientResult();
    }

    @GET
    @Path("/validate/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("[]")
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult validateAddress(@ApiParam(name = "address", value = "", required =
true)
        @PathParam("address") String address) {
        Result result = Result.getSuccess();
        Map<String, Object> map = new HashMap<>();
        map.put("value", AddressTool.validAddress(address));
        result.setData(map);
        return result.toRpcClientResult();
    }

```

```
}
```

```
@POST
```

```
@Path("/lock/{address}")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
@ApiOperation(value = "[ ] ", notes = "Clear the cache unlock account.")
```

```
public RpcClientResult lock(@ApiParam(name = "address", value = "", required = true)
```

```
@PathParam("address") String address) {
```

```
    Account account = accountService.getAccount(address).getData();
```

```
    if (null == account) {
```

```
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
```

```
    }
```

```
    accountCacheService.removeAccount(account.getAddress());
```

```
    BlockingQueue<Runnable> queue = scheduler.getQueue();
```

```
    String addr = account.getAddress().toString();
```

```
    Runnable scheduledFuture = (Runnable) accountUnlockSchedulerMap.get(addr);
```

```
    if (queue.contains(scheduledFuture)) {
```

```
        scheduler.remove(scheduledFuture);
```

```
        accountUnlockSchedulerMap.remove(addr);
```

```
    }
```

```
    Map<String, Boolean> map = new HashMap<>();
```

```
    map.put("value", true);
```

```
    return Result.getSuccess().setData(map).toRpcClientResult();
```

```
}
```

```
@POST
```

```
@Path("/unlock/{address}")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
@ApiOperation(value = "[ ] ")
```

```
public RpcClientResult unlock(@ApiParam(name = "address", value = "", required = true)
```

```
    @PathParam("address") String address,
```

```
    @ApiParam(name = "form", value = "", required = true)
```

```
    AccountUnlockForm form) {
```

```
    Account account = accountService.getAccount(address).getData();
```

```
    if (null == account) {
```

```
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
```

```
    }
```

```
    String addr = account.getAddress().toString();
```

```
    //,
```

```
    if (accountUnlockSchedulerMap.containsKey(addr)) {
```

```
        BlockingQueue<Runnable> queue = scheduler.getQueue();
```

```

        Runnable sf = (Runnable) accountUnlockSchedulerMap.get(addr);
        if (queue.contains(sf)) {
            scheduler.remove(sf);
            accountUnlockSchedulerMap.remove(addr);
        }
    }
    String password = form.getPassword();
    Integer unlockTime = form.getUnlockTime();
    try {
        account.unlock(password);
        accountCacheService.putAccount(account);
        if (null == unlockTime || unlockTime >
AccountConstant.ACCOUNT_MAX_UNLOCK_TIME) {
            unlockTime = AccountConstant.ACCOUNT_MAX_UNLOCK_TIME;
        }
        if (unlockTime < 0) {
            unlockTime = 0;
        }
        //
        ScheduledFuture scheduledFuture = scheduler.schedule(() -> {
            accountCacheService.removeAccount(account.getAddress());
        }, unlockTime, TimeUnit.SECONDS);
        accountUnlockSchedulerMap.put(addr, scheduledFuture);
    } catch (NulsException e) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
    }
    Map<String, Boolean> map = new HashMap<>();
    map.put("value", true);
    return Result.getSuccess().setData(map).toRpcClientResult();
}

@POST
@Path("/remark/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult setRemark(@ApiParam(name = "address", value = "", required = true)
    @PathParam("address") String address,
    @ApiParam(name = "form", value = "") AccountRemarkForm

```

```

accountRemarkForm) {
    String remark = accountRemarkForm.getRemark();
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(remark)) {
        remark = null;
    }
    Result result = accountBaseService.setRemark(address, remark);
    if (result.isSuccess()) {
        Map<String, Boolean> map = new HashMap<>();
        map.put("value", (Boolean) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

@POST
@Path("/password/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult setPassword(@ApiParam(name = "address", value = "", required = true)
    @PathParam("address") String address,
    @ApiParam(name = "form", value = "", required = true)
    AccountPasswordForm form) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    String password = form.getPassword();
    if (StringUtils.isBlank(password)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }
    Result result = accountBaseService.setPassword(address, password);
    if (result.isSuccess()) {
        Map<String, Boolean> map = new HashMap<>();

```



```

        map.put("value", (Boolean) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

@POST
@Path("/offline/password/")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult setPassword(@ApiParam(name = "form", value = "", required = true)
                                   OfflineAccountPasswordForm form) {
    String address = form.getAddress();
    String priKey = form.getPriKey();
    String password = form.getPassword();

    if (StringUtils.isBlank(address) || !AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(priKey) || !ECKKey.isValidPrivteHex(priKey)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(password) || !StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }

    //
    ECKKey key = ECKKey.fromPrivate(new BigInteger(1, Hex.decode(priKey)));
    try {
        String newAddress = AccountTool.newAddress(key).getBase58();
        if (!newAddress.equals(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
    } catch (NulsException e) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }

    try {

```

```

        Account account = AccountTool.createAccount(priKey);
        account.encrypt(password);
        Map<String, String> map = new HashMap<>();
        map.put("value", Hex.encode(account.getEncryptedPriKey()));
        return Result.getSuccess().setData(map).toRpcClientResult();
    } catch (NulsException e) {
        return Result.getFailed(AccountErrorCode.FAILED).toRpcClientResult();
    }
}

@PUT
@Path("/offline/password/")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult updatePassword(@ApiParam(name = "form", value = "", required = true)
                                     OfflineAccountPasswordForm form) {
    String address = form.getAddress();
    String priKey = form.getPriKey();
    String password = form.getPassword();
    String newPassword = form.getNewPassword();

    if (StringUtils.isBlank(address) || !AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(priKey)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(password) || !StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }
    if (StringUtils.isBlank(newPassword) || !StringUtils.validPassword(newPassword)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }

    try {
        byte[] priKeyBytes = AESEncrypt.decrypt(Hex.decode(priKey), password);
        Account tempAccount = AccountTool.createAccount(Hex.encode(priKeyBytes));

```

```

        if (!address.equals(tempAccount.getAddress().getBase58())) {
            return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        Result result = getEncryptedPrivateKey(address, Hex.encode(priKeyBytes),
newPassword);
        if (result.isSuccess()) {
            Map<String, Boolean> map = new HashMap<>();
            map.put("value", (Boolean) result.getData());
            result.setData(map);
        }
        return result.toRpcClientResult();
    } catch (Exception e) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
    }
}

```

```

public Result getEncryptedPrivateKey(String address, String priKey, String password) {
    if (!ECKey.isValidPrivteHex(priKey)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
    }
    Account account;
    try {
        account = AccountTool.createAccount(priKey);
        if (!address.equals(account.getAddress().getBase58())) {
            return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
        }
        account.encrypt(password);
    } catch (NulsException e) {
        return Result.getFailed(AccountErrorCode.FAILED);
    }
    return Result.getSuccess().setData(Hex.encode(account.getEncryptedPriKey()));
}

```

@PUT

@Path("/password/{address}")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "[ ] ")

@ApiResponses(value = {

@ApiResponse(code = 200, message = "success", response = RpcClientResult.class)

})

public RpcClientResult updatePassword(@ApiParam(name = "address", value = "", required =

true)

```
        @PathParam("address") String address,
        @ApiParam(name = "form", value = "", required = true)
        AccountUpdatePasswordForm form) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    String password = form.getPassword();
    String newPassword = form.getNewPassword();
    if (StringUtils.isBlank(password)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(newPassword)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }
    if (!StringUtils.validPassword(newPassword)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }
    Result result = accountBaseService.changePassword(address, password, newPassword);
    if (result.isSuccess()) {
        Map<String, Boolean> map = new HashMap<>();
        map.put("value", (Boolean) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}
```

@PUT

@Path("/password/prikey")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "[ ] ")

@ApiResponses(value = {

@ApiResponse(code = 200, message = "success", response = RpcClientResult.class)

})

```
public RpcClientResult updatePasswordByPriKey(@ApiParam(name = "form", value = "",
required = true)
```

```

        AccountPriKeyChangePasswordForm form) {

    String prikey = form.getPriKey();
    if (!ECKey.isValidPrivteHex(prikey)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    String newPassword = form.getPassword();
    if (StringUtils.isBlank(newPassword)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!StringUtils.validPassword(newPassword)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_FORMAT_WRONG).toRpcClientResult();
    }
    Result result = accountService.importAccount(prikey, newPassword);
    if (result.isSuccess()) {
        Account account = (Account) result.getData();
        Map<String, String> map = new HashMap<>();
        map.put("value", account.getAddress().toString());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

@POST
@Path("/password/keystore")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] AccountKeyStore")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult updatePasswordByAccountKeyStore(@ApiParam(name = "form", value
= "", required = true)

```

```

        AccountKeyStoreResetPasswordForm form) {

```

```

    if (null == form || null == form.getAccountKeyStoreDto()) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    AccountKeyStoreDto accountKeyStoreDto = form.getAccountKeyStoreDto();

    String password = form.getPassword();
    if (!StringUtils.validPassword(password)) {

```

```

        return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
    }
    Result result =
accountService.updatePasswordByAccountKeyStore(accountKeyStoreDto.toAccountKeyStore(),
password);
    if (result.isSuccess()) {
        Account account = (Account) result.getData();
        Map<String, String> map = new HashMap<>();
        map.put("value", account.getAddress().toString());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

```

@POST
@Path("/export/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] AccountKeyStore ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult export(@ApiParam(name = "address", value = "", required = true)
    @PathParam("address") String address,
    @ApiParam(name = "form", value = "")
        AccountPasswordForm form, @Context HttpServletResponse
response) {
    if (StringUtils.isNotBlank(address) && !AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Result<AccountKeyStore> result = accountService.exportAccountToKeyStore(address,
form.getPassword());
    if (result.isFailed()) {
        return result.toRpcClientResult();
    }
    AccountKeyStore accountKeyStore = result.getData();
    return Result.getSuccess().setData(new
AccountKeyStoreDto(accountKeyStore)).toRpcClientResult();
}

```

/\*\*

```

*
* Export file
*/
private void backUpFile(AccountKeyStoreDto accountKeyStoreDto, HttpServletResponse
response) {
    try {
        String fileName =
accountKeyStoreDto.getAddress().concat(AccountConstant.ACCOUNTKEYSTORE_FILE_SUFFI
X);

        //1.ContentType
        response.setContentType("application/octet-stream");
        //2.
        response.addHeader("Content-Disposition", "attachment;filename=" + new
String(fileName.getBytes("utf-8")));
        response.getOutputStream().write(JSONUtils.obj2json(accountKeyStoreDto).getBytes());
        response.getOutputStream().flush();
    } catch (Exception e) {
        Log.error("Export Exception!");
    }
}

```

```

/**
*
* Export file
*/
private Result backUpFile(String path, AccountKeyStoreDto accountKeyStoreDto) {
    File backupFile = new File(path);
    //if not directory , create directory
    if (!backupFile.isDirectory()) {
        if (!backupFile.mkdirs()) {
            return Result.getFailed(KernelErrorCode.FILE_OPERATION_FAILED);
        }
        if (!backupFile.exists() && !backupFile.mkdir()) {
            return Result.getFailed(KernelErrorCode.FILE_OPERATION_FAILED);
        }
    }
    String fileName =
accountKeyStoreDto.getAddress().concat(AccountConstant.ACCOUNTKEYSTORE_FILE_SUFFI
X);
    backupFile = new File(backupFile, fileName);
    try {

```

```

        if (!backupFile.exists() && !backupFile.createNewFile()) {
            return Result.getFailed(KernelErrorCode.FILE_OPERATION_FAILED);
        }
    } catch (IOException e) {
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    }
    FileOutputStream fileOutputStream = null;
    try {
        fileOutputStream = new FileOutputStream(backupFile);
        fileOutputStream.write(JSONUtils.obj2json(accountKeyStoreDto).getBytes());
    } catch (Exception e) {
        return Result.getFailed(KernelErrorCode.PARSE_JSON_FAILED);
    } finally {
        if (fileOutputStream != null) {
            try {
                fileOutputStream.close();
            } catch (IOException e) {
                Log.error(e);
            }
        }
    }
    return Result.getSuccess().setData("The path to the backup file is " + path + File.separator +
fileName);
}

```

```

@POST
@Path("/import")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] AccountKeyStore")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult importAccount(@ApiParam(name = "form", value = "", required = true)
AccountKeyStoreImportForm form) {

    if (null == form || null == form.getAccountKeyStoreDto() || null == form.getOverwrite()) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    AccountKeyStoreDto accountKeyStoreDto = form.getAccountKeyStoreDto();
    if (!form.getOverwrite()) {
        Account account =
accountService.getAccount(accountKeyStoreDto.getAddress()).getData();

```



```

        if (null != account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_EXIST).toRpcClientResult();
        }
    }
    String password = form.getPassword();
    if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
    }
    Result result =
accountService.importAccountFormKeyStore(accountKeyStoreDto.toAccountKeyStore(),
password);
    if (result.isSuccess()) {
        Account account = (Account) result.getData();
        Map<String, String> map = new HashMap<>();
        map.put("value", account.getAddress().toString());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

```

@POST
@Path("/import/keystore")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.MULTIPART_FORM_DATA)
@ApiOperation(value = "[ ] AccountKeyStore")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult importAccountByKeystoreFile(@FormDataParam("keystore")
InputStream in,
                                                    @FormDataParam("password") String password,
                                                    @FormDataParam("overwrite") Boolean overwrite) {

    if (null == in || null == overwrite) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
    }
    Result<AccountKeyStoreDto> rs = getAccountKeyStoreDto(in);
}

```

```

    if (rs.isFailed()) {
        return rs.toRpcClientResult();
    }
    AccountKeyStoreDto accountKeyStoreDto = rs.getData();
    if (!overwrite) {
        Account account =
accountService.getAccount(accountKeyStoreDto.getAddress()).getData();
        if (null != account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_EXIST).toRpcClientResult();
        }
    }

    Result result =
accountService.importAccountFormKeyStore(accountKeyStoreDto.toAccountKeyStore(),
password);
    if (result.isSuccess()) {
        Account account = (Account) result.getData();
        Map<String, String> map = new HashMap<>();
        map.put("value", account.getAddress().toString());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

```

private Result<AccountKeyStoreDto> getAccountKeyStoreDto(InputStream in) {
    StringBuilder ks = new StringBuilder();
    InputStreamReader inputStreamReader = null;
    BufferedReader bufferedReader = null;
    String str;
    try {
        inputStreamReader = new InputStreamReader(in);
        bufferedReader = new BufferedReader(inputStreamReader);
        while ((str = bufferedReader.readLine()) != null) {
            if (!str.isEmpty()) {
                ks.append(str);
            }
        }
        AccountKeyStoreDto accountKeyStoreDto = JSONUtils.json2pojo(ks.toString(),
AccountKeyStoreDto.class);
        return Result.getSuccess().setData(accountKeyStoreDto);
    } catch (FileNotFoundException e) {

```

```

        return Result.getFailed(AccountErrorCode.ACCOUNTKEYSTORE_FILE_NOT_EXIST);
    } catch (IOException e) {
        return Result.getFailed(AccountErrorCode.ACCOUNTKEYSTORE_FILE_DAMAGED);
    } catch (Exception e) {
        return Result.getFailed(AccountErrorCode.ACCOUNTKEYSTORE_FILE_DAMAGED);
    } finally {
        if (bufferedReader != null) {
            try {
                bufferedReader.close();

            } catch (IOException e) {
                Log.error(e);
            }
        }
        if (inputStreamReader != null) {
            try {
                inputStreamReader.close();
            } catch (IOException e) {
                Log.error(e);
            }
        }
    }
}

```

```

@POST
@Path("/import/prikeys")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ,,, , ", notes = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult importAccountByPriKeys(@ApiParam(name = "form", value = "",
required = true) AccountPriKeysPasswordForm form) {

    List<String> list = form.getPriKey();
    List<String> success = new ArrayList<>();
    String password = form.getPassword();
    if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
        return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
    }
}

```

```

for (String priKey : list) {
    if (!ECKKey.isValidPrivteHex(priKey)) {
        Result result = Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
        result.setMsg(result.getMsg() + ", " + success.size() + "");
        return result.toRpcClientResult();
    }
    Result result = accountService.importAccount(priKey, password);
    if (result.isSuccess()) {
        Account account = (Account) result.getData();
        success.add(account.getAddress().toString());
    } else {
        result.setMsg(result.getMsg() + ", " + success.size() + "");
        return result.toRpcClientResult();
    }
}
Map<String, List<String>> map = new HashMap<>();
map.put("list", success);
return Result.getSuccess().setData(map).toRpcClientResult();
}

```

@POST

@Path("/import/pri")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "[ ] ", notes = "")

@ApiResponses(value = {

    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)

})

public RpcClientResult importAccountByPriKey(@ApiParam(name = "form", value = "", required = true) AccountPriKeyPasswordForm form) {

```

    if (null == form || null == form.getOverwrite()) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }

```

```

    String priKey = form.getPriKey();
    if (!ECKKey.isValidPrivteHex(priKey)) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }

```

```

    if (!form.getOverwrite()) {
        ECKKey key = null;
        try {

```

```

        key = EKey.fromPrivate(new BigInteger(1, Hex.decode(form.getPriKey())));
    } catch (Exception e) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Address address = new Address(NulsContext.DEFAULT_CHAIN_ID,
NulsContext.DEFAULT_ADDRESS_TYPE, SerializeUtils.sha256hash160(key.getPubKey()));
    Account account = accountService.getAccount(address).getData();
    if (null != account) {
        return Result.getFailed(AccountErrorCode.ACCOUNT_EXIST).toRpcClientResult();
    }
}

```

```

String password = form.getPassword();
if (StringUtils.isNotBlank(password) && !StringUtils.validPassword(password)) {
    return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
}
Result result = accountService.importAccount(priKey, password);
if (result.isSuccess()) {
    Account account = (Account) result.getData();
    Map<String, String> map = new HashMap<>();
    map.put("value", account.getAddress().toString());
    result.setData(map);
}
return result.toRpcClientResult();
}

```

```

@POST
@Path("/remove/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult removeAccount(@ApiParam(name = "address", value = "", required =
true)

        @PathParam("address") String address,
        @ApiParam(name = "", value = "JSONFormat", required = true)
        AccountPasswordForm form) {
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
}

```

```

Result result = accountService.removeAccount(address, form.getPassword());
if (result.isSuccess()) {
    Map<String, Boolean> map = new HashMap<>();
    map.put("value", (Boolean) result.getData());
    result.setData(map);
}
return result.toRpcClientResult();
}

@POST
@Path("/createMultiAccount")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ", notes = "result.data: List<String>")
@ApiResponses(value = {@ApiResponse(code = 200, message = "success", response =
RpcClientResult.class)
})
public RpcClientResult createMultiAccount(@ApiParam(name = "form", value = "", required =
true)
                                     MultiAccountCreateForm form) {
    if (null == form || null == form.getPubkeys() || form.getPubkeys().size() == 0) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (form.getM() == 0) {
        form.setM(form.getPubkeys().size());
    }
    if(form.getPubkeys().size() < form.getM()){
        return
Result.getFailed(AccountErrorCode.SIGN_COUNT_TOO_LARGE).toRpcClientResult();
    }
    Set<String> pubkeySet = new HashSet<>(form.getPubkeys());
    if(pubkeySet.size() < form.getPubkeys().size()){
        return Result.getFailed(AccountErrorCode.PUBKEY_REPEAT).toRpcClientResult();
    }
    Result result = accountService.createMultiAccount(form.getPubkeys(), form.getM());
    if (result.isFailed()) {
        return result.toRpcClientResult();
    }
    MultiSigAccountDto account = new MultiSigAccountDto((MultiSigAccount) result.getData());
    return Result.getSuccess().setData(account).toRpcClientResult();
}

```

```

@POST
@Path("multiAccount/mutilAlias")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult alias(@ApiParam(name = "form", value = "", required = true)
                             CreateMultiAliasForm form) {
    if(NulsContext.MAIN_NET_VERSION <=1){
        return Result.getFailed(KernelErrorCode.VERSION_TOO_LOW).toRpcClientResult();
    }
    if (form == null) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (!AddressTool.validAddress(form.getSignAddress()) ||
!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(form.getAlias())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (!StringUtils.validAlias(form.getAlias())) {
        return
Result.getFailed(AccountErrorCode.ALIAS_FORMAT_WRONG).toRpcClientResult();
    }
    if (!StringUtils.validAlias(form.getAlias())) {
        return
Result.getFailed(AccountErrorCode.ALIAS_FORMAT_WRONG).toRpcClientResult();
    }
    if (!aliasService.isAliasUsable(form.getAlias())) {
        return Result.getFailed(AccountErrorCode.ALIAS_EXIST).toRpcClientResult();
    }
    Result result =
aliasService.setMutilAlias(form.getAddress(),form.getSignAddress(),form.getAlias(),form.getPassword());
    if (result.isSuccess()) {
        Map<String, String> map = new HashMap<>();
        map.put("txData", (String) result.getData());
        result.setData(map);
    }
    return result.toRpcClientResult();
}

```

```
}
```

```
@POST
@Path("/importMultiAccount")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ", notes = "result.data: boolean ")
@ApiResponses(value = {@ApiResponse(code = 200, message = "success", response =
RpcClientResult.class)
})
public RpcClientResult importMultiAccount(@ApiParam(name = "form", value = "", required =
true)
                                     MultiAccountImportForm form) {
    if (null == form || null == form.getPubkeys() || StringUtils.isBlank(form.getAddress()) ||
form.getPubkeys().size() == 0) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    if (form.getM() == 0) {
        form.setM(form.getPubkeys().size());
    }
    if(form.getPubkeys().size() < form.getM()){
        return
Result.getFailed(AccountErrorCode.SIGN_COUNT_TOO_LARGE).toRpcClientResult();
    }
    Result result = accountService.saveMultiSigAccount(form.getAddress(), form.getPubkeys(),
form.getM());
    if (result.isFailed()) {
        return result.toRpcClientResult();
    }
    return result.toRpcClientResult();
}

@GET
@Path("/multiAccount/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] MultiSigAccount")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult getMultiSigAccount(@ApiParam(name = "address", value = "", required
= true)
                                     @PathParam("address") String address) throws Exception {
```



```

    if (StringUtils.isNotBlank(address) && !AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Result<MultiSigAccount> result = accountService.getMultiSigAccount(address);
    if (result.isFailed()) {
        return result.toRpcClientResult();
    }
    return Result.getSuccess().setData(new
MultiSigAccountDto(result.getData())).toRpcClientResult();
}

```

```

@DELETE
@Path("/multiAccount/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult delMultiSigAccount(@ApiParam(name = "address", value = "", required
= true)

        @PathParam("address") String address) throws Exception {
    if (StringUtils.isNotBlank(address) && !AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    Result result = accountService.removeMultiSigAccount(address);
    Map<String, Boolean> map = new HashMap<>();
    map.put("result", result.isSuccess());
    return result.setData(map).toRpcClientResult();
}

```

```

@GET
@Path("/multiAccounts")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "[ ] MultiSigAccount")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult getMultiSigAccountList() throws Exception {

```

```

    Result<List<MultiSigAccount>> result = accountService.getMultiSigAccountList();
    if (result.isFailed()) {
        return result.toRpcClientResult();
    }

```

```

    }
    List<MultiSigAccountDto> list = new ArrayList<>();
    for (MultiSigAccount account : result.getData()) {
        MultiSigAccountDto dto = new MultiSigAccountDto(account);
        list.add(dto);
    }
    return Result.getSuccess().setData(list).toRpcClientResult();
}

@GET
@Path("multiAccount/alias/fee")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[[] ")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
})
public RpcClientResult multiAliasFee(@BeanParam() MultiAliasFeeForm form) throws
Exception{
    if (!AddressTool.validAddress(form.getAddress())) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    if (StringUtils.isBlank(form.getAlias())) {
        return Result.getFailed(AccountErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Result result = aliasService.getMultiAliasFee(form.getAddress(), form.getAlias());
    AliasTransaction tx = new AliasTransaction();
    tx.setTime(TimeService.currentTimeMillis());
    Result<MultiSigAccount> sigAccountResult =
accountService.getMultiSigAccount(form.getAddress());
    MultiSigAccount multiSigAccount = sigAccountResult.getData();
    Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
    if(redeemScript == null){
        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
    }
    Alias alias = new Alias(AddressTool.getAddress(form.getAddress()), form.getAlias());
    tx.setTxData(alias);
    try {
        CoinDataResult coinDataResult =
accountLedgerService.getMutilCoinData(AddressTool.getAddress(form.getAddress()),
AccountConstant.ALIAS_NA, tx.size(),
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        if (!coinDataResult.isEnough()) {

```

```

        return
Result.getFailed(AccountErrorCode.INSUFFICIENT_BALANCE).toRpcClientResult();
    }
    CoinData coinData = new CoinData();
    coinData.setFrom(coinDataResult.getCoinList());
    Coin change = coinDataResult.getChange();
    if (null != change) {
        //toList
        List<Coin> toList = new ArrayList<>();
        toList.add(change);
        coinData.setTo(toList);
    }
    Coin coin = new Coin(NulsConstant.BLACK_HOLE_ADDRESS, Na.parseNuls(1), 0);
    coinData.addTo(coin);
    tx.setCoinData(coinData);
} catch (Exception e) {
    Log.error(e);
    return
Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
}
//m*+
int scriptSignLenth = redeemScript.getProgram().length + ((int)multiSigAccount.getM()) * 72;
Result rs =
accountLedgerService.getMultiMaxAmountOfOnce(AddressTool.getAddress(form.getAddress()),
tx, TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES,scriptSignLenth);
Map<String, Long> map = new HashMap<>();
Long fee = null;
Long maxAmount = null;
if (result.isSuccess()) {
    fee = ((Na) result.getData()).getValue();
}
if (rs.isSuccess()) {
    maxAmount = ((Na) rs.getData()).getValue();
}
map.put("fee", fee);
map.put("maxAmount", maxAmount);
result.setData(map);
return result.toRpcClientResult();
}
}

```

```
rpc\src\test\java\io\nuls\account\test\CreateAccountTest.java
```

```
*/
```

```
package io.nuls.account.test;
```

```
import io.nuls.core.tools.log.Log;
```

```
import io.nuls.core.tools.str.StringUtils;
```

```
import org.junit.Test;
```

```
import java.io.*;
```

```
import java.net.HttpURLConnection;
```

```
import java.net.URL;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
/**
```

```
 * @author: Niels Wang
```

```
*/
```

```
public class CreateAccountTest {
```

```
    @Test
```

```
    public void test() {
```

```
        for (int x = 0; x < 100; x++) {
```

```
            String param = "{\"count\": 100}";
```

```
            String url = "http://127.0.0.1:8001/api/account";
```

```
            String res = post(url, param, "utf-8");
```

```
            System.out.println(res);
```

```
        }
```

```
    }
```

```
    public String post(String url, final String param, String encoding) {
```

```
        StringBuffer sb = new StringBuffer();
```

```
        OutputStream os = null;
```

```
        InputStream is = null;
```

```
        InputStreamReader isr = null;
```

```
        BufferedReader br = null;
```

```

// UTF-8
if (StringUtils.isNull(encoding)) {
    encoding = "UTF-8";
}
try {
    URL u = new URL(url);
    HttpURLConnection connection = (HttpURLConnection) u.openConnection();
    connection.setRequestProperty("Content-Type", "application/json");
    connection.setDoOutput(true);
    connection.setDoInput(true);
    connection.setRequestMethod("POST");

    connection.connect();

    os = connection.getOutputStream();
    os.write(param.getBytes(encoding));
    os.flush();
    is = connection.getInputStream();
    isr = new InputStreamReader(is, encoding);
    br = new BufferedReader(isr);
    String line;
    while ((line = br.readLine()) != null) {
        sb.append(line);
        sb.append("");
    }
} catch (Exception ex) {
    ex.printStackTrace();
} finally {
    if (is != null) {
        try {
            is.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (os != null) {
        try {
            os.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
}

```

```

        if (isr != null) {
            try {
                isr.close();
            } catch (IOException e) {
                Log.error(e);
            }
        }
        if (br != null) {
            try {
                br.close();
            } catch (IOException e) {
                Log.error(e);
            }
        }
    }
    return sb.toString();
}
}

```

133:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-storage\src\main\java\io.nuls.account.storage\constant\AccountStorageConstant.java  
\*/

```
package io.nuls.account.storage.constant;
```

```
import io.nuls.core.tools.str.StringUtils;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
public interface AccountStorageConstant {
```

```
    /**
```

```
     *
```

```
     * The name of the account table
```

```
    */
```

```
    String DB_NAME_ACCOUNT = "account";
```

```
    String DB_NAME_MULTI_SIG_ACCOUNT = "multi_account";
```

```
    /**
```

```
     * key
```

```
     * The name of the key account table
```

```

    */
    byte[] DEFAULT_ACCOUNT_KEY = StringUtils.bytes("DEFAULT_ACCOUNT");
    /**
     *
     * The name of the account table
     */
    String DB_NAME_ACCOUNT_ALIAS = "account_alias";

}

```

134:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-storage\src\main\java\io.nuls.account.storage\po\AccountPo.java

```

*/

```

```

package io.nuls.account.storage.po;

import io.nuls.account.model.Account;
import io.nuls.kernel.model.Address;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.EncryptedData;
import io.nuls.core.tools.log.Log;
import java.math.BigInteger;

/**
 * @author: Charlie
 */
public class AccountPo {

    private transient Address addressObj;

    private String address;

    private Long createTime;

    private String alias;

    private byte[] pubKey;

    private byte[] priKey;

    private byte[] encryptedPriKey;

```

```

private byte[] extend;

private int status;

private String remark;

public AccountPo(){
}

public AccountPo(Account account){
    this.addressObj = account.getAddress();
    this.address = account.getAddress().toString();
    this.createTime = account.getCreateTime();
    this.alias = account.getAlias();
    this.pubKey = account.getPubKey();
    this.priKey = account.getPriKey();
    this.encryptedPriKey = account.getEncryptedPriKey();
    this.extend = account.getExtend();
    this.status = account.getStatus();
    this.remark = account.getRemark();
}

public Account toAccount(){
    Account account = new Account();
    account.setCreateTime(this.getCreateTime());
    try {
        account.setAddress(Address.fromHashs(this.getAddress()));
    } catch (Exception e) {
        Log.error(e);
    }
    account.setAlias(this.getAlias());
    account.setExtend(this.getExtend());
    account.setPriKey(this.getPriKey());
    account.setPubKey(this.getPubKey());
    account.setEncryptedPriKey(this.getEncryptedPriKey());
    if (this.getPriKey() != null && this.getPriKey().length > 1) {
        account.setEcKey(ECKKey.fromPrivate(new BigInteger(1, account.getPriKey())));
    } else {
        account.setEcKey(ECKKey.fromEncrypted(new EncryptedData(this.getEncryptedPriKey()),
this.getPubKey()));
    }
    account.setStatus(this.getStatus());
}

```



```
    account.setRemark(this.remark);
    return account;
}

public String getAddress() {
    return address;
}

public Address getAddressObj() {
    return addressObj;
}

public void setAddressObj(Address addressObj) {
    this.addressObj = addressObj;
}

public void setAddress(String address) {
    this.address = address;
}

public Long getCreateTime() {
    return createTime;
}

public void setCreateTime(Long createTime) {
    this.createTime = createTime;
}

public String getAlias() {
    return alias;
}

public void setAlias(String alias) {
    this.alias = alias;
}

public byte[] getPubKey() {
    return pubKey;
}

public void setPubKey(byte[] pubKey) {
    this.pubKey = pubKey;
}
```

```
}

public byte[] getPriKey() {
    return priKey;
}

public void setPriKey(byte[] priKey) {
    this.priKey = priKey;
}

public byte[] getEncryptedPriKey() {
    return encryptedPriKey;
}

public void setEncryptedPriKey(byte[] encryptedPriKey) {
    this.encryptedPriKey = encryptedPriKey;
}

public byte[] getExtend() {
    return extend;
}

public void setExtend(byte[] extend) {
    this.extend = extend;
}

public int getStatus() {
    return status;
}

public void setStatus(int status) {
    this.status = status;
}

public String getRemark() {
    return remark;
}

public void setRemark(String remark) {
    this.remark = remark;
}
}
```

```
135:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
storage\src\main\java\io.nuls.account.storage\po\AliasPo.java  
*/
```

```
package io.nuls.account.storage.po;
```

```
import io.nuls.account.model.Alias;  
import io.nuls.kernel.exception.NulsException;  
import io.nuls.kernel.model.BaseNulsData;  
import io.nuls.kernel.utils.NulsByteBuffer;  
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
```

```
import java.io.IOException;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
public class AliasPo extends BaseNulsData {
```

```
    private byte[] address;
```

```
    private String alias;
```

```
    public AliasPo() {  
    }
```

```
    public AliasPo(Alias alias) {  
        this.address = alias.getAddress();  
        this.alias = alias.getAlias().trim();  
  
    }
```

```
    public Alias toAlias() {  
        return new Alias(this.address, this.getAlias().trim());  
    }
```

```
    public String getAlias() {  
        return alias;  
    }
```

```
    public void setAlias(String alias) {
```

```

        this.alias = alias == null ? null : alias.trim();
    }

    public byte[] getAddress() {
        return address;
    }

    public void setAddress(byte[] address) {
        this.address = address;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {

    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {

    }

    @Override
    public int size() {
        return 0;
    }
}

```

136:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-storage\src\main\java\io.nuls.account.storage\service\AccountStorageService.java  
\*/

```

package io.nuls.account.storage.service;

import io.nuls.kernel.model.Address;
import io.nuls.account.storage.po.AccountPo;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 *
 * Account data storage service interface

```

```

*
* @author: Charlie
*/
public interface AccountStorageService {

    /**
     *
     * Create accounts
     * @param accountPoList
     * @param accountPoList Account collection to be created
     * @return the result of the opration
     */
    Result saveAccountList(List<AccountPo> accountPoList);

    /**
     *
     * Create account
     * @param account
     * @return
     */
    Result saveAccount(AccountPo account);

    /**
     *
     * Delete account
     * @param address Account address to be deleted
     * @return the result of the opration
     */
    Result removeAccount(Address address);

    /**
     *
     * @return the result of the opration and Result<List<Account>>
     */
    Result<List<AccountPo>> getAccountList();

    /**
     *
     * According to the account to obtain account information
     * @param address
     * @return the result of the opration
     */
}

```

```
Result<AccountPo> getAccount(Address address);
```

```
/**
```

```
*
```

```
* According to the account to obtain account information
```

```
* @param address
```

```
* @return the result of the opration
```

```
*/
```

```
Result<AccountPo> getAccount(byte[] address);
```

```
/**
```

```
*
```

```
* Update account information according to the account.
```

```
* @param account The account to be updated.
```

```
* @return the result of the opration
```

```
*/
```

```
Result updateAccount(AccountPo account);
```

```
/**
```

```
*
```

```
* Set default account
```

```
* @param account default
```

```
* @return
```

```
*/
```

```
Result saveDefaultAccount(AccountPo account);
```

```
/**
```

```
*
```

```
* get default account
```

```
* @return
```

```
*/
```

```
Result<AccountPo> getDefaultAccount();
```

```
/**
```

```
* remove default account
```

```
* @return
```

```
*/
```

```
Result removeDefaultAccount();
```

```
}
```

```
storage\src\main\java\io.nuls.account.storage\service\AliasStorageService.java
```

```
*/
```

```
package io.nuls.account.storage.service;
```

```
import io.nuls.account.storage.po.AliasPo;
```

```
import io.nuls.kernel.model.Result;
```

```
import java.util.List;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
public interface AliasStorageService {
```

```
    Result<List<AliasPo>> getAliasList();
```

```
    Result<AliasPo> getAlias(String alias);
```

```
    Result saveAlias(AliasPo aliasPo);
```

```
    Result removeAlias(String alias);
```

```
}
```

```
138:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-
```

```
storage\src\main\java\io.nuls.account.storage\service\impl\AccountStorageServiceImpl.java
```

```
*/
```

```
package io.nuls.account.storage.service.impl;
```

```
import io.nuls.account.constant.AccountErrorCode;
```

```
import io.nuls.kernel.model.Address;
```

```
import io.nuls.account.storage.constant.AccountStorageConstant;
```

```
import io.nuls.account.storage.po.AccountPo;
```

```
import io.nuls.account.storage.service.AccountStorageService;
```

```
import io.nuls.db.constant.DBErrorCode;
```

```
import io.nuls.db.service.BatchOperation;
```

```
import io.nuls.db.service.DBService;
```

```
import io.nuls.kernel.exception.NulsException;
```

```
import io.nuls.kernel.exception.NulsRuntimeException;
```

```
import io.nuls.kernel.lite.annotation.Autowired;
```

```

import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 * @author: Charlie
 */
@Service
public class AccountStorageServiceImpl implements AccountStorageService, InitializingBean {

    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    @Override
    public void afterPropertiesSet() throws NulsException {
        Result result = this.dbService.createArea(AccountStorageConstant.DB_NAME_ACCOUNT);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveAccountList(List<AccountPo> accountPoList) {
        BatchOperation batch =
dbService.createWriteBatch(AccountStorageConstant.DB_NAME_ACCOUNT);
        for (AccountPo po : accountPoList) {
            batch.putModel(po.getAddressObj().getAddressBytes(), po);
        }
        return batch.executeBatch();
    }

    @Override
    public Result saveAccount(AccountPo po) {
        return dbService.putModel(AccountStorageConstant.DB_NAME_ACCOUNT,
po.getAddressObj().getAddressBytes(), po);
    }

```



```

    }

    @Override
    public Result removeAccount(Address address) {
        if (null == address || address.getAddressBytes() == null || address.getAddressBytes().length
<= 0) {
            return Result.getFailed(AccountErrorCode.PARAMETER_ERROR);
        }
        return dbService.delete(AccountStorageConstant.DB_NAME_ACCOUNT,
address.getAddressBytes());
    }

    @Override
    public Result<List<AccountPo>> getAccountList() {
        List<AccountPo> listPo =
dbService.values(AccountStorageConstant.DB_NAME_ACCOUNT, AccountPo.class);
        return Result.getSuccess().setData(listPo);
    }

    @Override
    public Result<AccountPo> getAccount(Address address) {
        return this.getAccount(address.getAddressBytes());
    }

    @Override
    public Result<AccountPo> getAccount(byte[] address) {
        AccountPo account = dbService.getModel(AccountStorageConstant.DB_NAME_ACCOUNT,
address, AccountPo.class);
        if (null == account){
            return Result.getFailed();
        }
        return Result.getSuccess().setData(account);
    }

    @Override
    public Result updateAccount(AccountPo po) {
        if (null == po.getAddressObj()){
            po.setAddressObj(new Address(po.getAddress()));
        }
        AccountPo account = dbService.getModel(AccountStorageConstant.DB_NAME_ACCOUNT,
po.getAddressObj().getAddressBytes(), AccountPo.class);
        if (null == account){

```

```

        return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST);
    }
    return dbService.putModel(AccountStorageConstant.DB_NAME_ACCOUNT,
po.getAddressObj().getAddressBytes(), po);
}

@Override
public Result saveDefaultAccount(AccountPo po) {
    return dbService.putModel(AccountStorageConstant.DB_NAME_ACCOUNT,
AccountStorageConstant.DEFAULT_ACCOUNT_KEY, po);
}

@Override
public Result<AccountPo> getDefaultAccount() {
    return
Result.getSuccess().setData(dbService.getModel(AccountStorageConstant.DB_NAME_ACCOUNT, AccountStorageConstant.DEFAULT_ACCOUNT_KEY));
}

@Override
public Result removeDefaultAccount() {
    return dbService.delete(AccountStorageConstant.DB_NAME_ACCOUNT,
AccountStorageConstant.DEFAULT_ACCOUNT_KEY);
}
}

```

139:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-storage\src\main\java\io.nuls.account.storage\service\impl\AliasStorageServiceImpl.java  
\*/

```

package io.nuls.account.storage.service.impl;

import io.nuls.account.storage.constant.AccountStorageConstant;
import io.nuls.account.storage.po.AliasPo;
import io.nuls.account.storage.service.AliasStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;

```

```

import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Result;

import java.io.UnsupportedEncodingException;
import java.util.List;

/**
 * @author: Charlie
 */
@Service
public class AliasStorageServiceImpl implements AliasStorageService, InitializingBean {

    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    @Override
    public void afterPropertiesSet() throws NulsException {
        Result result =
this.dbService.createArea(AccountStorageConstant.DB_NAME_ACCOUNT_ALIAS);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result<List<AliasPo>> getAliasList() {
        List<AliasPo> list =
dbService.values(AccountStorageConstant.DB_NAME_ACCOUNT_ALIAS, AliasPo.class);
        return Result.getSuccess().setData(list);
    }

    @Override
    public Result<AliasPo> getAlias(String alias) {
        try {
            byte[] aliasByte = alias.getBytes(NulsConfig.DEFAULT_ENCODING);
            AliasPo aliasPo =
dbService.getModel(AccountStorageConstant.DB_NAME_ACCOUNT_ALIAS, aliasByte,

```

```

AliasPo.class);
        return Result.getSuccess().setData(aliasPo);
    } catch (UnsupportedEncodingException e) {
        Log.error(e);
        return Result.getFailed();
    }
}

@Override
public Result saveAlias(AliasPo aliasPo) {
    try {
        return dbService.putModel(AccountStorageConstant.DB_NAME_ACCOUNT_ALIAS,
aliasPo.getAlias().getBytes(NulsConfig.DEFAULT_ENCODING), aliasPo);
    } catch (UnsupportedEncodingException e) {
        Log.error(e);
        return Result.getFailed();
    }
}

```

```

@Override
public Result removeAlias(String alias) {
    try {
        return dbService.delete(AccountStorageConstant.DB_NAME_ACCOUNT_ALIAS,
alias.getBytes(NulsConfig.DEFAULT_ENCODING));
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed();
    }
}
}

```

140:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-storage\src\main\java\io.nuls.account.storage\service\impl\MultiSigAccountStorageServiceImpl.java

```

package io.nuls.account.storage.service.impl;

import io.nuls.account.storage.constant.AccountStorageConstant;
import io.nuls.account.storage.service.MultiSigAccountStorageService;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;

```

```

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 * @author: Niels Wang
 */
@Component
public class MultiSigAccountStorageServiceImpl implements MultiSigAccountStorageService,
InitializingBean {

    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    @Override
    public void afterPropertiesSet() throws NulsException {
        Result result =
this.dbService.createArea(AccountStorageConstant.DB_NAME_MULTI_SIG_ACCOUNT);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    /**
     *
     * save account
     */
    @Override
    public Result saveAccount(Address address, byte[] multiSigAccount) {
        return dbService.put(AccountStorageConstant.DB_NAME_MULTI_SIG_ACCOUNT,
address.getAddressBytes(), multiSigAccount);
    }
}

```

```

/**
 *
 * Delete account
 *
 * @param address Account address to be deleted
 * @return the result of the opration
 */
@Override
public Result removeAccount(Address address) {
    return dbService.delete(AccountStorageConstant.DB_NAME_MULTI_SIG_ACCOUNT,
address.getAddressBytes());
}

/**
 *
 *
 * @return the result of the opration and Result<List<Account>>
 */
@Override
public Result<List<byte[]>> getAccountList() {
    List<byte[]> valueList =
dbService.valueList(AccountStorageConstant.DB_NAME_MULTI_SIG_ACCOUNT);
    Result<List<byte[]>> result = new Result<>();
    result.setData(valueList);
    return result;
}

/**
 *
 * According to the account to obtain account information
 *
 * @return the result of the opration
 */
@Override
public Result<byte[]> getAccount(Address address) {
    return new
Result<byte[]>().setData(dbService.get(AccountStorageConstant.DB_NAME_MULTI_SIG_ACCO
UNT, address.getAddressBytes()));
}
}

```

```
141:F:\git\coin\nuls\nuls-1.1.3\nuls\account-module\base\account-  
storage\src\main\java\io.nuls.account.storage\service\MultiSigAccountStorageService.java  
*/
```

```
package io.nuls.account.storage.service;
```

```
import io.nuls.account.storage.po.AccountPo;
```

```
import io.nuls.kernel.model.Address;
```

```
import io.nuls.kernel.model.Result;
```

```
import java.util.List;
```

```
/**
```

```
*
```

```
* Account data storage service interface
```

```
*
```

```
* @author: Charlie
```

```
*/
```

```
public interface MultiSigAccountStorageService {
```

```
    /**
```

```
    *
```

```
    * save account
```

```
    * @return
```

```
    */
```

```
    Result saveAccount(Address address,byte[] multiSigAccount);
```

```
    /**
```

```
    *
```

```
    * Delete account
```

```
    * @param address Account address to be deleted
```

```
    * @return the result of the opration
```

```
    */
```

```
    Result removeAccount(Address address);
```

```
    /**
```

```
    *
```

```
    * @return the result of the opration and Result<List<Account>>
```

```
    */
```

```
    Result<List<byte[]>> getAccountList();
```

```
    /**
```

```

*
* According to the account to obtain account information
* @param address
* @return the result of the opration
*/

```

```
Result<byte[]> getAccount(Address address);
```

```
}
```

```
142:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\main\java\io\nuls\client\Bootstrap.java
*/
```

```
package io.nuls.client;
```

```

import io.nuls.client.rpc.RpcServerManager;
import io.nuls.client.rpc.constant.RpcConstant;
import io.nuls.client.rpc.resources.thread.ShutdownHook;
import io.nuls.client.rpc.resources.util.FileUtil;
import io.nuls.client.storage.LanguageService;
import io.nuls.client.version.WalletVersionManager;
import io.nuls.client.web.view.WebViewBootstrap;
import io.nuls.consensus.poc.cache.TxMemoryPool;
import io.nuls.core.tools.date.DateUtil;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.i18n.I18nUtils;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.module.service.ModuleService;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.manager.ConnectionManager;
import io.nuls.network.model.Node;
import io.nuls.network.service.NetworkService;

import java.io.File;
import java.io.UnsupportedEncodingException;
import java.lang.reflect.Field;
import java.net.URLDecoder;

```



```

import java.nio.charset.Charset;
import java.util.*;

import static java.nio.charset.StandardCharsets.UTF_8;

/**
 * @author: Niels Wang
 */
public class Bootstrap {

    public static void main(String[] args) {
        Thread.currentThread().setName("Nuls");
        try {
            System.setProperty("protostuff.runtime.allow_null_array_element", "true");
            System.setProperty("file.encoding", UTF_8.name());
            Field charset = Charset.class.getDeclaredField("defaultCharset");
            charset.setAccessible(true);
            charset.set(null, UTF_8);
            sysStart();
        } catch (Exception e) {
            Log.error(e);
            System.exit(-1);
        }
    }

    private static void copyWebFiles() throws UnsupportedOperationException {
        String path = Bootstrap.class.getClassLoader().getResource("").getPath() + "/temp/" +
NulsConfig.VERSION + "/conf/client-web/";
        path = URLDecoder.decode(path, "UTF-8");
        File source = new File(path);
        if (!source.exists()) {
            Log.info("source not exists:" + path);
            return;
        }
        Log.info("do the files copy!");
        File target = new
File(URLDecoder.decode(Bootstrap.class.getClassLoader().getResource("").getPath(), "UTF-8") +
"/conf/client-web/");
        FileUtil.deleteFolder(target);
        FileUtil.copyFolder(source, target);
    }
}

```

```

private static void sysStart() throws Exception {
    do {
        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();
        WalletVersionManager.start();
        initModules();
        String ip =
NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
RpcConstant.CFG_RPC_SERVER_IP, RpcConstant.DEFAULT_IP);
        int port =
NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
RpcConstant.CFG_RPC_SERVER_PORT, RpcConstant.DEFAULT_PORT);
        copyWebFiles();
        RpcServerManager.getInstance().startServer(ip, port);

        LanguageService languageService =
NulsContext.getServiceBean(LanguageService.class);
        String languageDB = (String) languageService.getLanguage().getData();
        String language = null == languageDB ? I18nUtils.getLanguage() : languageDB;
        I18nUtils.setLanguage(language);
        if (null == languageDB) {
            languageService.saveLanguage(language);
        }
    } while (false);

    // if isDaemon flag is true, don't launch the WebView
    boolean isDaemon =
NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
RpcConstant.CFG_RPC_DAEMON, false);
    if (!isDaemon) {
        TaskManager.asyncExecuteRunnable(new WebViewBootstrap());
    }

    int i = 0;
    Map<NulsDigestData, List<Node>> map = new HashMap<>();
    NulsContext context = NulsContext.getInstance();
    while (true) {
        if (context.getStop() > 0) {
            if (context.getStop() == 2) {
                Runtime.getRuntime().addShutdownHook(new ShutdownHook());
            }
        }
    }
}

```

```

        System.exit(0);
    }
    if (NulsContext.mastUpgrade) {
        //
        ConnectionManager.getInstance().shutdown();
        Log.error(">>>>> The new protocol version has taken effect, the network connection
has been disconnected please upgrade immediately *****");
    }
    try {
        Thread.sleep(1000L);
    } catch (InterruptedException e) {
        Log.error(e);
    }
    if (i > 10) {
        i = 0;
        Log.info("----- netTime :
" + (DateUtil.convertDate(new Date(TimeService.currentTimeMillis()))));
        Block bestBlock = NulsContext.getInstance().getBestBlock();
        Collection<Node> nodes =
NulsContext.getServiceBean(NetworkService.class).getAvailableNodes();
        Log.info("bestHeight:" + bestBlock.getHeader().getHeight() + " , txCount : " +
bestBlock.getHeader().getTxCount() + " , tx memory pool count : " +
TxMemoryPool.getInstance().size() + " - " + TxMemoryPool.getInstance().getOrphanPoolSize() + "
, hash : " + bestBlock.getHeader().getHash() + " , nodeCount:" + nodes.size());
        map.clear();
        for (Node node : nodes) {
            List<Node> ips = map.get(node.getBestBlockHash());
            if (null == ips) {
                ips = new ArrayList<>();
                map.put(node.getBestBlockHash(), ips);
            }
            ips.add(node);
        }
        for (NulsDigestData key : map.keySet()) {
            List<Node> nodeList = map.get(key);
            long height = nodeList.get(0).getBestBlockHeight();
            StringBuilder ids = new StringBuilder();
            for (Node node : nodeList) {
                ids.append(", " + node.getId());
            }
            Log.info("height:" + height + " , count:" + nodeList.size() + " , hash:" +
key.getDigestHex() + ids);

```

```

    }
    } else {
        i++;
    }
}
}

```

```

private static void initModules() {
    Map<String, String> bootstrapClasses = null;
    try {
        bootstrapClasses = getModuleBootstrapClass();
    } catch (Exception e) {
        Log.error(e);
    }
    if (null == bootstrapClasses || bootstrapClasses.isEmpty()) {
        return;
    }
    ModuleService.getInstance().startModules(bootstrapClasses);
}

```

```

private static Map<String, String> getModuleBootstrapClass() throws Exception {
    Map<String, String> map = new HashMap<>();
    List<String> moduleNameList = NulsConfig.MODULES_CONFIG.getSectionList();
    if (null == moduleNameList || moduleNameList.isEmpty()) {
        return map;
    }
    for (String moduleName : moduleNameList) {

        String className = null;
        try {
            className = NulsConfig.MODULES_CONFIG.getCfgValue(moduleName,
NulsConstant.MODULE_BOOTSTRAP_KEY);
        } catch (Exception e) {
            continue;
        }
        map.put(moduleName, className);
    }
    return map;
}
}

```

```
143:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\cmd\CommandHandler.java  
*/
```

```
package io.nuls.client.cmd;
```

```
import com.fasterxml.jackson.core.JsonParser;  
import io.nuls.account.rpc.cmd.*;  
import io.nuls.account.ledger.rpc.cmd.CreateMultiTransferProcess;  
import io.nuls.account.ledger.rpc.cmd.GetAccountTxListProcessor;  
import io.nuls.account.ledger.rpc.cmd.SignMultiTransactionProcess;  
import io.nuls.account.ledger.rpc.cmd.TransferProcessor;  
import io.nuls.client.constant.CommandConstant;  
import io.nuls.client.rpc.constant.RpcConstant;  
import io.nuls.consensus.poc.rpc.cmd.*;  
import io.nuls.contract.rpc.cmd.*;  
import io.nuls.core.tools.cfg.ConfigLoader;  
import io.nuls.core.tools.json.JSONUtils;  
import io.nuls.core.tools.log.Log;  
import io.nuls.core.tools.str.StringUtils;  
import io.nuls.kernel.cfg.NulsConfig;  
import io.nuls.kernel.constant.KernelErrorCode;  
import io.nuls.kernel.constant.NulsConstant;  
import io.nuls.kernel.exception.NulsException;  
import io.nuls.kernel.exception.NulsRuntimeException;  
import io.nuls.kernel.i18n.I18nUtils;  
import io.nuls.kernel.processor.CommandProcessor;  
import io.nuls.kernel.utils.RestFulUtils;  
import io.nuls.ledger.rpc.cmd.GetTxProcessor;  
import io.nuls.network.rpc.cmd.GetNetInfoProcessor;  
import io.nuls.network.rpc.cmd.GetNetNodesProcessor;  
import io.nuls.protocol.rpc.cmd.GetBestBlockHeaderProcessor;  
import io.nuls.protocol.rpc.cmd.GetBlockHeaderProcessor;  
import io.nuls.protocol.rpc.cmd.GetBlockProcessor;  
import io.nuls.utxo.accounts.rpc.cmd.GetUtxoAccountsProcessor;  
import jline.console.ConsoleReader;  
import jline.console.completer.ArgumentCompleter;  
import jline.console.completer.Completer;  
import jline.console.completer.StringsCompleter;  
  
import java.io.IOException;  
import java.io.UnsupportedEncodingException;
```

```

import java.net.URLDecoder;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class CommandHandler {

    public static final Map<String, CommandProcessor> PROCESSOR_MAP = new TreeMap<>();

    public static ConsoleReader CONSOLE_READER;

    private static final Pattern CMD_PATTERN = Pattern.compile("\"[^\"]+\"|'[^']*'");

    public CommandHandler() {

    }

    /**
     *
     */
    private void init() {
        /**
         * ledger
         */
        register(new GetTxProcessor());

        /**
         * block
         */
        register(new GetBlockHeaderProcessor());
        register(new GetBlockProcessor());
        register(new GetBestBlockHeaderProcessor());

        /**
         * account
         */
        register(new BackupAccountProcessor());
    }

```

```

register(new CreateProcessor());
register(new GetAccountProcessor());
register(new GetAccountsProcessor());
//    register(new GetAssetProcessor());//
register(new GetBalanceProcessor());
//    register(new GetWalletBalanceProcessor());//
register(new GetPrivateKeyProcessor());
register(new ImportByKeyStoreProcessor());
register(new ImportByPrivateKeyProcessor());
register(new ImportForcedByPrivateKeyProcessor());
register(new RemoveAccountProcessor());
register(new ResetPasswordProcessor());
register(new SetAliasProcessor());
register(new SetPasswordProcessor());

/**
 * Multi-signature account
 */
register(new CreateMultiSigAccountProcessor());
register(new ImportMultiSigAccountProcessor());
register(new GetMultiSigAccountListProcessor());
register(new GetMultiSigAccountProcessor());
register(new RemoveMultiSigAccountProcessor());
register(new GetMultiSigAccountCountProcessor());
register(new CreateMultiSigAccountProcessor());

register(new CreateMultiTransferProcess());
register(new SignMultiTransactionProcess());
register(new CreateMultiAliasProcess());
register(new CreateMultiAgentProcessor());
register(new CreateMultiDepositProcessor());
register(new CreateMultiWithdrawProcessor());
register(new CreateMultiStopAgentProcessor());

/**
 * accountLedger
 */
register(new TransferProcessor());
register(new GetAccountTxListProcessor());
//    register(new GetUTXOProcessor());//

/**

```

```

* consensus
*/
register(new CreateAgentProcessor());
register(new GetConsensusProcessor());
register(new DepositProcessor());
register(new WithdrawProcessor());
register(new StopAgentProcessor());
register(new GetAgentProcessor());
register(new GetAgentsProcessor());
register(new GetDepositedAgentsProcessor());
register(new GetDepositedsProcessor());
register(new GetDepositedInfoProcessor());

/**
* network
*/
register(new GetNetInfoProcessor());
register(new GetNetNodesProcessor());

/**
* system
*/
register(new ExitProcessor());
register(new HelpProcessor());
register(new VersionProcessor());
register(new UpgradeProcessor());
/**
* utxoAccounts
*/
register(new GetUtxoAccountsProcessor());

/**
* contract
*/
register(new GetContractTxProcessor());
register(new GetContractResultProcessor());
register(new GetContractInfoProcessor());
register(new GetContractBalanceProcessor());
register(new GetContractTxListProcessor());
register(new GetContractAddressValidProcessor());
register(new GetWalletContractsProcessor());
register(new GetTokenBalanceProcessor());

```



```

register(new CreateContractProcessor());
register(new CallContractProcessor());
register(new ViewContractProcessor());
register(new TransferToContractProcessor());
register(new TokenTransferProcessor());
register(new DeleteContractProcessor());
register(new GetContractConstructorProcessor());
JSONUtils.getInstance().configure(JsonParser.Feature.ALLOW_SINGLE_QUOTES, true);
sdkInit();
}

private void sdkInit() {
    String port = null;
    try {
        NulsConfig.MODULES_CONFIG =
ConfigLoader.loadIni(NulsConstant.MODULES_CONFIG_FILE);
        port = NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
RpcConstant.CFG_RPC_SERVER_PORT);
    } catch (Exception e) {
        Log.error("CommandHandler start failed", e);
        throw new NulsRuntimeException(KernelErrorCode.FAILED);
    }
    if (StringUtils.isBlank(port)) {
        RestFulUtils.getInstance().setServerUri("http://" + RpcConstant.DEFAULT_IP + ":" +
RpcConstant.DEFAULT_PORT + RpcConstant.PREFIX);
    } else {
        String ip = null;
        try {
            ip = NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
"server.ip").trim();
            if ("0.0.0.0".equals(ip)) {
                ip = RpcConstant.DEFAULT_IP;
            }
        } catch (Exception e) {
            ip = RpcConstant.DEFAULT_IP;
        }
        RestFulUtils.getInstance().setServerUri("http://" + ip + ":" + port + RpcConstant.PREFIX);
    }
}

public static void main(String[] args) {
    /**

```

```

    * windows, falseWindows APIJava IO
    * If the operating system is windows, it may cause the console to read part of the loop, can
be set to false,
    * bypass the native Windows API, use the Java IO stream output directly
    */
if (System.getProperties().getProperty("os.name").toUpperCase().indexOf("WINDOWS") != -
1) {
    System.setProperty("jline.WindowsTerminal.directConsole", "false");
}
CommandHandler instance = new CommandHandler();
instance.init();
try {
    I18nUtils.setLanguage("en");
} catch (NulsException e) {
    e.printStackTrace();
}
try {
    CONSOLE_READER = new ConsoleReader();
    List<Completer> completers = new ArrayList<Completer>();
    completers.add(new StringsCompleter(PROCESSOR_MAP.keySet()));
    CONSOLE_READER.addCompleter(new ArgumentCompleter(completers));
    String line;
    do {
        line = CONSOLE_READER.readLine(CommandConstant.COMMAND_PS1);
        if (StringUtils.isBlank(line)) {
            continue;
        }
        String[] cmdArgs = parseArgs(line);
        System.out.print(instance.processCommand(cmdArgs) + "\n");
    } while (line != null);
} catch (IOException e) {

} finally {
    try {
        if (!CONSOLE_READER.delete()) {
            CONSOLE_READER.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

```

}

private static String[] parseArgs(String line) throws UnsupportedOperationException {
    if(StringUtils.isBlank(line)) {
        return new String[0];
    }
    Matcher matcher = CMD_PATTERN.matcher(line);
    String result = line;
    while (matcher.find()) {
        String group = matcher.group();
        String subGroup = group.substring(1, group.length() - 1);
        String encoder = URLEncoder.encode(subGroup, StandardCharsets.UTF_8.toString());
        result = result.replace(group, encoder);
    }

    String[] args = result.split("\\s+");
    for(int i = 0, length = args.length; i < length; i++) {
        args[i] = URLDecoder.decode(args[i], StandardCharsets.UTF_8.toString());
    }
    return args;
}

private String processCommand(String[] args) {
    int length = args.length;
    if (length == 0) {
        return CommandConstant.COMMAND_ERROR;
    }
    String command = args[0];
    CommandProcessor processor = PROCESSOR_MAP.get(command);
    if (processor == null) {
        return command + " not a nuls command!";
    }
    if (length == 2 && CommandConstant.NEED_HELP.equals(args[1])) {
        return processor.getHelp();
    }
    try {
        boolean result = processor.argsValidate(args);
        if (!result) {
            return "args incorrect:\n" + processor.getHelp();
        }
        return processor.execute(args).toString();
    } catch (Exception e) {

```

```

        return CommandConstant.EXCEPTION + ": " + e.getMessage();
    }
}

private void register(CommandProcessor processor) {
    PROCESSOR_MAP.put(processor.getCommand(), processor);
}
}

```

144:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\cmd\ExitProcessor.java  
\*/

```
package io.nuls.client.cmd;
```

```
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
```

```
/**
```

```
 * @author: Charlie
```

```
*/
```

```
public class ExitProcessor implements CommandProcessor {
```

```
    @Override
```

```
    public String getCommand() {
        return "exit";
    }

```

```
    @Override
```

```
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription());
        return bulider.toString();
    }

```

```
    @Override
```

```
    public String getCommandDescription() {
        return "exit --exit the nuls command";
    }

```

```
    @Override
```

```

    public boolean argsValidate(String[] args) {
        if(args.length > 1) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        System.exit(1);
        return CommandResult.getSuccess("");
    }
}

145:F:\git\coin\nuls\nuls-1.1.3\nuls\client-
module\client\src\main\java\io\nuls\client\cmd\HelpProcessor.java
*/

package io.nuls.client.cmd;

import io.nuls.client.constant.CommandConstant;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.RestFulUtils;

/**
 * @author: Charlie
 */
public class HelpProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "help";
    }

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())

```

```

        .newLine("\t[-a] show all commands and options of command - optional");
    return bulider.toString();
}

@Override
public String getCommandDescription() {
    return "help [-a] --print all commands";
}

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if(length > 2) {
        return false;
    }
    if(length == 2 && !CommandConstant.NEED_ALL.equals(args[1])) {
        return false;
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    int length = args.length;
    StringBuilder str = new StringBuilder();
    str.append("all commands:");
    for (CommandProcessor processor : CommandHandler.PROCESSOR_MAP.values()) {
        str.append("\n");
        if(length == 2 && CommandConstant.NEED_ALL.equals(args[1])) {
            str.append(processor.getHelp());
        } else {
            str.append(processor.getCommandDescription());
        }
    }
    return CommandResult.getSuccess(str.toString());
}
}

```

146:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\cmd\UpgradeProcessor.java

\*/

```

package io.nuls.client.cmd;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.RestFulUtils;

import java.io.IOException;
import java.util.Map;

/**
 * @author: Charlie
 */
public class UpgradeProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
    public String getCommand() {
        return "upgrade";
    }

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription()).newLine("\t<version> the version you want to
upgrade -required");
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "upgrade <version> --upgrade to the newest version and restart the client where
download complete";
    }

    @Override
    public boolean argsValidate(String[] args) {
        if (args.length == 2 && args[1] != null && args[1].trim().length() > 0) {

```

```

        return true;
    }
    return false;
}

```

@Override

```

public CommandResult execute(String[] args) {
    RpcClientResult result = restFul.post("/client/upgrade/" + args[1], "");
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    //todo
    int count = 0;
    while (true) {
        result = restFul.get("/client/upgrade", null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        int percentage = (int) ((Map) result.getData()).get("percentage");
        //todo
        print(percentage + "%", count);
        if (percentage < 10) {
            count = 2;
        } else {
            count = 3;
        }

        if (percentage == 100) {
            break;
        }
        try {
            Thread.sleep(500L);
        } catch (InterruptedException e) {
            Log.error(e.getMessage());
        }
    }
    result = restFul.post("/client/restart", "");

    return CommandResult.getResult(result);
}

private void print(String s, int backCount) {

```



```

        try {
            for (int i = 0; i < backCount; i++) {
//todo        CommandHandler.CONSOLE_READER.backspace();
//        CommandHandler.CONSOLE_READER.delete();
            }
            CommandHandler.CONSOLE_READER.clearScreen();
            CommandHandler.CONSOLE_READER.print(s);
            CommandHandler.CONSOLE_READER.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}

```

147:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\main\java\io\nuls\client\cmd\VersionProcessor.java  
\*/

```
package io.nuls.client.cmd;
```

```

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.RestFulUtils;

```

```
import java.util.Map;
```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
public class VersionProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```
    @Override
```

```
    public String getCommand() {
```

```
        return "version";
```

```
    }
```

```
    @Override
```

```

public String getHelp() {
    CommandBuilder bulider = new CommandBuilder();
    bulider.newLine(getCommandDescription());
    return bulider.toString();
}

```

```

@Override
public String getCommandDescription() {
    return "version --show the version of local&network";
}

```

```

@Override
public boolean argsValidate(String[] args) {
    if (args.length != 1) {
        return false;
    }
    return true;
}

```

```

@Override
public CommandResult execute(String[] args) {
    RpcClientResult result = restFul.get("/client/version", null);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    Map<String, Object> map = ((Map)result.getData());
    if (null != map) {
        String infromation = (String)map.get("infromation");
        map.put("infromation", infromation.replaceAll("\r\n", ""));
    }
    result.setData(map);

    return CommandResult.getResult(result);
}
}

```

148:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\main\java\io\nuls\client\constant\CommandConstant.java

package io.nuls.client.constant;

/\*\*

\* Created by Niels on 2017/10/30.

```

*
*/
public interface CommandConstant {

    String COMMAND_PS1 = "nuls>>> ";
    String COMMAND_ERROR = "command error! ";
    String EXCEPTION = "Exception";

    String CMD_EXIT = "exit";
    String CMD_HELP = "help";
    String NEED_HELP = "-h";
    String NEED_ALL = "-a";

    String CMD_SYS = "sys";
    String CMD_ACCT = "account";

    String DB_LANGUAGE = "language";

}

```

149:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\main\java\io\nuls\client\rpc\config\NulsResourceConfig.java

```

*/
package io.nuls.client.rpc.config;

import com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider;
import io.nuls.client.rpc.filter.RpcServerFilter;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.lite.core.SpringLiteContext;
import org.glassfish.jersey.media.multipart.MultiPartFeature;
import org.glassfish.jersey.server.ResourceConfig;

import javax.ws.rs.Path;
import java.util.Collection;

/**
 * @author Niels
 */
public class NulsResourceConfig extends ResourceConfig {

    public NulsResourceConfig() {
        register(io.swagger.jaxrs.listing.ApiListingResource.class);
    }

```

```

register(io.swagger.jaxrs.listing.AcceptHeaderApiListingResource.class);
register(NulsSwaggerSerializers.class);
register(MultiPartFeature.class);
register(RpcServerFilter.class);
register(JacksonJsonProvider.class);

```

```

Collection<Object> list = SpringLiteContext.getAllBeanList();
for (Object object : list) {
    if (object.getClass().getAnnotation(Path.class) != null) {
        Log.debug("register restFul resource:{", object.getClass());
        register(object);
    }
}
}
}
}

```

150:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\config\NulsSwaggerSerializers.java  
\*/

```

package io.nuls.client.rpc.config;

```

```

import io.swagger.jaxrs.listing.SwaggerSerializers;
import io.swagger.models.Path;
import io.swagger.models.Swagger;

```

```

import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.MultivaluedMap;
import java.io.IOException;
import java.io.OutputStream;
import java.lang.annotation.Annotation;
import java.lang.reflect.Type;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

```

```

/**

```

```

 * @author: Niels Wang

```

```

 */

```

```

public class NulsSwaggerSerializers extends SwaggerSerializers {

```

@Override

```
public void writeTo(Swagger data, Class<?> type, Type genericType, Annotation[] annotations,
MediaType mediaType, MultivaluedMap<String, Object> headers, OutputStream out) throws
IOException {
    Map<String, Path> map = data.getPaths();
    Set<String> keyset = new HashSet<>(map.keySet());
    for (String key : keyset) {
        if(key.startsWith("/api")){
            continue;
        }
        Path path = map.get(key);
        map.remove(key);
        map.put("/api" + key, path);
    }
    super.writeTo(data, type, genericType, annotations, mediaType, headers, out);
}
}
```

151:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\constant\RpcConstant.java  
\*/

package io.nuls.client.rpc.constant;

/\*\*

\* @author: Niels Wang

\*/

public interface RpcConstant {

String PACKAGES = "io.nuls.rpc.resource.impl";

int DEFAULT\_PORT = 8001;

String DEFAULT\_IP = "127.0.0.1";

String PREFIX = "/api";

String CFG\_RPC\_SECTION = "client";

String CFG\_RPC\_SERVER\_IP = "server.ip";

String CFG\_RPC\_SERVER\_PORT = "server.port" ;

String CFG\_RPC\_REQUEST\_WHITE\_SHEET="request.white.sheet";

String CFG\_RPC\_DAEMON = "daemon";

String WHITE\_SHEET\_SPLIT = ",";

```
}
```

```
152:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\filter\HttpContextHelper.java
```

```
*/
```

```
package io.nuls.client.rpc.filter;
```

```
import org.glassfish.grizzly.http.server.Request;
```

```
/**
```

```
 * @author Niels
```

```
*/
```

```
public class HttpContextHelper {
```

```
    private static final ThreadLocal<Request> LOCAL = new ThreadLocal<>();
```

```
    public static void put(Request request) {  
        LOCAL.set(request);  
    }
```

```
    public static Request getRequest() {  
        return LOCAL.get();  
    }
```

```
    public static void removeRequest() {  
        LOCAL.remove();  
    }
```

```
}
```

```
153:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\filter\RpcServerFilter.java
```

```
*/
```

```
package io.nuls.client.rpc.filter;
```

```
import io.nuls.client.rpc.constant.RpcConstant;
```

```
import io.nuls.core.tools.log.Log;
```

```
import io.nuls.core.tools.str.StringUtils;
```

```
import io.nuls.kernel.cfg.NulsConfig;
```

```
import io.nuls.kernel.constant.KernelErrorCode;
```

```
import io.nuls.kernel.exception.NulsException;
```

```
import io.nuls.kernel.exception.NulsRuntimeException;
```

```

import io.nuls.kernel.model.ErrorData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.RpcClientResult;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.ext.ExceptionMapper;
import java.io.IOException;

/**
 * @author Niels
 */
public class RpcServerFilter implements ContainerRequestFilter, ContainerResponseFilter,
ExceptionMapper<Exception> {

    @Context
    private HttpServletRequest request;
    @Context
    private HttpServletResponse response;
    private String[] ipArray;
    private boolean all = false;

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {
        if (!whiteSheetVerifier(request)) {
            throw new NulsRuntimeException(KernelErrorCode.REQUEST_DENIED);
        }
        requestContext.setProperty("start", System.currentTimeMillis());
    }

    @Override
    public void filter(ContainerRequestContext requestContext, ContainerResponseContext
responseContext) {
        //      Log.info("url:{},IP:{},useTime:{}, params:{},result:{},",

```

```

requestContext.getUriInfo().getRequestUri().getPath() + "?" +
requestContext.getUriInfo().getRequestUri().getQuery(),
grizzlyRequestProvider.get().getRemoteAddr()
//      , (System.currentTimeMillis() -
Long.parseLong(requestContext.getProperty("start").toString())), null,
responseContext.getEntity());

    //todo
    response.setHeader("Access-control-Allow-Origin", request.getHeader("Origin"));
    response.setHeader("Access-Control-Allow-Methods",
"GET,POST,OPTIONS,PUT,DELETE");
    response.setHeader("Access-Control-Allow-Headers", request.getHeader("Access-Control-
Request-Headers"));

}

@Override
public Response toResponse(Exception e) {
//    System.out.println("-----" + request.getRequestURI());
    Log.error("RequestURI is " + request.getRequestURI(), e);
    RpcClientResult result;
    if (e instanceof NulsException) {
        NulsException exception = (NulsException) e;
        result = new RpcClientResult(false, exception.getErrorCode());
    } else if (e instanceof NulsRuntimeException) {
        NulsRuntimeException exception = (NulsRuntimeException) e;
        result = new RpcClientResult(false, new ErrorData(exception.getCode(),
exception.getMessage()));
    } else {
        result = Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
    }

    return Response.ok(result, MediaType.APPLICATION_JSON).build();
}

private boolean whiteSheetVerifier(HttpServletRequest request) {
    if (all) {
        return true;
    }
    if (ipArray == null) {
        String ips = null;
        try {
            ips =

```



```

NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
RpcConstant.CFG_RPC_REQUEST_WHITE_SHEET);
    } catch (Exception e) {
        Log.error(e);
    }
    if (StringUtils.isBlank(ips)) {
        return false;
    }
    this.ipArray = ips.split(RpcConstant.WHITE_SHEET_SPLIT);
    for (String ip : ipArray) {
        if ("0.0.0.0".equals(ip)) {
            this.all = true;
            return true;
        }
    }
}
String reallp = request.getRemoteAddr();
for (String ip : ipArray) {
    if (ip.equals(reallp)) {
        return true;
    }
}
return false;
}
}

```

154:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\resources\ClientResource.java  
\*/

```

package io.nuls.client.rpc.resources;

import io.nuls.client.rpc.RpcServerManager;
import io.nuls.client.rpc.resources.dto.ProtocolContainerDTO;
import io.nuls.client.rpc.resources.dto.UpgradeProcessDTO;
import io.nuls.client.rpc.resources.dto.VersionDto;
import io.nuls.client.rpc.resources.thread.UpgradeThread;
import io.nuls.client.version.SyncVersionRunner;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;

```

```

import io.nuls.core.tools.str.VersionUtils;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.manager.ConnectionManager;
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.base.version.ProtocolContainer;
import io.nuls.protocol.storage.po.ProtocolTempInfoPo;
import io.nuls.protocol.storage.service.VersionManagerStorageService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author: Niels Wang
 */
@Path("/client")
@Api(value = "/client", description = "Client")
@Component
public class ClientResource {

    @Autowired
    private VersionManagerStorageService versionManagerStorageService;

    @GET
    @Path("/version")
    @Produces(MediaType.APPLICATION_JSON)

```

```

@ApiOperation(value = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = VersionDto.class)
})
public RpcClientResult getVersion() {
    VersionDto rpcVersion = new VersionDto();
    rpcVersion.setMyVersion(NulsConfig.VERSION);
    SyncVersionRunner syncer = SyncVersionRunner.getInstance();
    rpcVersion.setNewestVersion(syncer.getNewestVersion());
    if (StringUtils.isBlank(rpcVersion.getNewestVersion())) {
        rpcVersion.setNewestVersion(NulsConfig.VERSION);
    }
    rpcVersion.setInformation(syncer.getInformation());
    boolean upgradable = VersionUtils.higherThan(rpcVersion.getNewestVersion(),
NulsConfig.VERSION);
    URL url = ClientResource.class.getClassLoader().getResource("libs");
    upgradable = upgradable && url != null;
    rpcVersion.setUpgradable(upgradable);
    rpcVersion.setNetworkVersion(NulsContext.MAIN_NET_VERSION);
    return Result.getSuccess().setData(rpcVersion).toRpcClientResult();
}

```

```

@GET
@Path("/upgrade")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response =
UpgradeProcessDTO.class)
})
public RpcClientResult getUpgradeProcess() {
    UpgradeProcessDTO dto = UpgradeThread.getInstance().getProcess();
    return Result.getSuccess().setData(dto).toRpcClientResult();
}

```

```

@POST
@Path("/upgrade/{version}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = Boolean.class)
})

```

```

public RpcClientResult startUpdate(@PathParam("version") String version) {
    AssertUtil.canNotEmpty(version);
    SyncVersionRunner syncor = SyncVersionRunner.getInstance();
    String newestVersion = syncor.getNewestVersion();
    if(!VersionUtils.higherThan(newestVersion,NulsConfig.VERSION )){
        Result result = Result.getFailed(KernelErrorCode.NONEWVER);
        return result.toRpcClientResult();
    }
    if (!version.equals(newestVersion)) {
        Result result = Result.getFailed(KernelErrorCode.VERSION_NOT_NEWEST);
        return result.toRpcClientResult();
    }
    URL url = ClientResource.class.getClassLoader().getResource("libs");
    if (null == url) {
        return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND).toRpcClientResult();
    }

    UpgradeThread thread = UpgradeThread.getInstance();
    if (thread.isUpgrading()) {
        return Result.getFailed(KernelErrorCode.UPGRADING).toRpcClientResult();
    }
    boolean result = thread.start();
    if (result) {
        TaskManager.createAndRunThread((short) 1, "upgrade", thread);
        Map<String, Boolean> map = new HashMap<>();
        map.put("value", true);
        return Result.getSuccess().setData(map).toRpcClientResult();
    }
    return Result.getFailed(KernelErrorCode.FAILED).toRpcClientResult();
}

```

@POST

@Path("/upgrade/stop")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "")

@ApiResponses(value = {

    @ApiResponse(code = 200, message = "success", response = Boolean.class)

})

```

public RpcClientResult stopUpdate() {

```

```

    UpgradeThread thread = UpgradeThread.getInstance();

```

```

    if (!thread.isUpgrading()) {

```

```

        return Result.getFailed(KernelErrorCode.NOT_UPGRADING).toRpcClientResult();
    }
}

```

```

    }
    if (thread.getProcess().getPercentage() == 100) {
        return Result.getFailed(KernelErrorCode.UPGRADING).toRpcClientResult();
    }
    boolean result = thread.stop();
    if (result) {
        Map<String, Boolean> map = new HashMap<>();
        map.put("value", true);
        return Result.getSuccess().setData(map).toRpcClientResult();
    }
    return Result.getFailed(KernelErrorCode.FAILED).toRpcClientResult();
}

```

@POST

@Path("/restart")

@Produces(MediaType.APPLICATION\_JSON)

@ApiOperation(value = "")

@ApiResponses(value = {

@ApiResponse(code = 200, message = "success", response = Boolean.class)

})

public RpcClientResult restartSystem() {

URL url = ClientResource.class.getClassLoader().getResource("libs");

if (url == null) {

return Result.getFailed(KernelErrorCode.FAILED).toRpcClientResult();

}

Thread t = new Thread(new Runnable() {

@Override

public void run() {

try {

Thread.sleep(1000L);

RpcServerManager.getInstance().shutdown();

ConnectionManager.getInstance().shutdown();

} catch (Exception e) {

Log.error(e);

}

NulsContext.getInstance().exit(2);

}

});

t.start();

Map<String, Boolean> map = new HashMap<>();

map.put("value", true);

return Result.getSuccess().setData(map).toRpcClientResult();

```

}

@POST
@Path("/stop")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = Boolean.class)
})
public RpcClientResult stopSystem() {
    NulsContext.getInstance().exit(1);
    Map<String, Boolean> map = new HashMap<>();
    map.put("value", true);
    return Result.getSuccess().setData(map).toRpcClientResult();
}

@GET
@Path("/protocol/info")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response =
ProtocolContainerDTO.class)
})
public RpcClientResult getProtocolInfo() {
    BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
    List<ProtocolContainerDTO> list = new ArrayList<>();
    ProtocolContainer protocolContainer =
NulsVersionManager.getProtocolContainer(NulsContext.CURRENT_PROTOCOL_VERSION);
    ProtocolContainerDTO pcDTO = new ProtocolContainerDTO(protocolContainer);
    if(pcDTO.getStatus() == ProtocolContainer.DELAY_LOCK){
        pcDTO.setEffectiveHeight(blockHeader.getHeight() + pcDTO.getCountdownDelay() + 1);
    }
    list.add(pcDTO);
    Map<String, ProtocolTempInfoPo> protocolTempMap =
versionManagerStorageService.getProtocolTempMap();
    for (ProtocolTempInfoPo protocolTempInfoPo : protocolTempMap.values()){
        ProtocolContainerDTO protocolContainerDTO = new
ProtocolContainerDTO(protocolTempInfoPo);
        if(protocolContainerDTO.getStatus() == ProtocolContainer.DELAY_LOCK){
            protocolContainerDTO.setEffectiveHeight(blockHeader.getHeight() +
protocolContainerDTO.getCountdownDelay() + 1);

```

```

    }
    list.add(protocolContainerDTO);
}
Map<String, List<ProtocolContainerDTO>> map = new HashMap<>();
map.put("list", list);
return Result.getSuccess().setData(map).toRpcClientResult();

}
}

```

155:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\resources\dto\ProtocolContainerDTO.java

```
*/
```

```
package io.nuls.client.rpc.resources.dto;
```

```
import io.nuls.protocol.base.version.ProtocolContainer;
import io.nuls.protocol.storage.po.ProtocolTempInfoPo;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
* @author: Charlie
```

```
* @date: 2018/9/15
```

```
*/
```

```
@ApiModel(value = "protocolContainerJSON")
```

```
public class ProtocolContainerDTO {
```

```
/**
```

```
*
```

```
*/
```

```
@ApiModelProperty(name = "version", value = "")
```

```
private Integer version;
```

```
****/
```

```
@ApiModelProperty(name = "percent", value = "")
```

```
private int percent;
```

```
/**
```

```
* (status = 0status1)
```

```
*/
```

```
@ApiModelProperty(name = "currentPercent", value = "")
```

```
private int currentPercent;
```

/\*\*\*/

@ApiModelProperty(name = "roundIndex", value = "")  
private long roundIndex;

/\*\*\*/

@ApiModelProperty(name = "delay", value = "")  
private long delay;

/\*\*

\*

\*/

@ApiModelProperty(name = "currentDelay", value = "")  
private long currentDelay;

/\*\*

\*

\*/

@ApiModelProperty(name = "countdownDelay", value = "")  
private long countdownDelay;

/\*\*

\*

\*/

@ApiModelProperty(name = "effectiveHeight", value = "")  
private Long effectiveHeight;

/\*\*

\*

\*/

@ApiModelProperty(name = "status", value = "")  
private int status;

public ProtocolContainerDTO(){}

public ProtocolContainerDTO(ProtocolContainer protocolContainer){  
 this.version = protocolContainer.getVersion();  
 this.percent = protocolContainer.getPercent();  
 this.currentPercent = protocolContainer.getPrePercent();  
 this.roundIndex = protocolContainer.getRoundIndex();  
 this.delay = protocolContainer.getDelay();  
 this.currentDelay = protocolContainer.getCurrentDelay();  
 this.countdownDelay = delay - currentDelay;  
}



```
    this.effectiveHeight = protocolContainer.getEffectiveHeight();
    this.status = protocolContainer.getStatus();
}
```

```
public ProtocolContainerDTO(ProtocolTemplInfoPo templInfoPo){
    this.version = templInfoPo.getVersion();
    this.percent = templInfoPo.getPercent();
    this.currentPercent = templInfoPo.getPrePercent();
    this.roundIndex = templInfoPo.getRoundIndex();
    this.delay = templInfoPo.getDelay();
    this.currentDelay = templInfoPo.getCurrentDelay();
    this.countdownDelay = delay - currentDelay;
    this.effectiveHeight = templInfoPo.getEffectiveHeight();
    this.status = templInfoPo.getStatus();
}
```

```
public Integer getVersion() {
    return version;
}
```

```
public void setVersion(Integer version) {
    this.version = version;
}
```

```
public int getPercent() {
    return percent;
}
```

```
public void setPercent(int percent) {
    this.percent = percent;
}
```

```
public long getRoundIndex() {
    return roundIndex;
}
```

```
public void setRoundIndex(long roundIndex) {
    this.roundIndex = roundIndex;
}
```

```
public long getDelay() {
    return delay;
}
```

```
}

public void setDelay(long delay) {
    this.delay = delay;
}

public long getCurrentDelay() {
    return currentDelay;
}

public void setCurrentDelay(long currentDelay) {
    this.currentDelay = currentDelay;
}

public long getCountdownDelay() {
    return countdownDelay;
}

public void setCountdownDelay(long countdownDelay) {
    this.countdownDelay = countdownDelay;
}

public int getCurrentPercent() {
    return currentPercent;
}

public void setCurrentPercent(int currentPercent) {
    this.currentPercent = currentPercent;
}

public Long getEffectiveHeight() {
    return effectiveHeight;
}

public void setEffectiveHeight(Long effectiveHeight) {
    this.effectiveHeight = effectiveHeight;
}

public int getStatus() {
    return status;
}
```

```
    public void setStatus(int status) {  
        this.status = status;  
    }  
}
```

156:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\resources\dto\UpgradeProcessDTO.java  
\*/

```
package io.nuls.client.rpc.resources.dto;
```

```
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
@ApiModel(value = "upgradeProcessJSON")
```

```
public class UpgradeProcessDTO {
```

```
    /**
```

```
     * 0
```

```
     * 1
```

```
     * 2
```

```
     * 3
```

```
     * 4
```

```
    */
```

```
@ApiModelProperty(name = "status", value = "0,1,2,3,4")
```

```
private int status = 0;
```

```
/**
```

```
 *
```

```
 */
```

```
@ApiModelProperty(name = "percentage", value = "0-100")
```

```
private int percentage = 0;
```

```
@ApiModelProperty(name = "message", value = "")
```

```
private String message;
```

```
@ApiModelProperty(name = "time", value = "")
```

```
private long time;
```

```
public int getStatus() {
```

```

        return status;
    }

    public void setStatus(int status) {
        this.status = status;
    }

    public int getPercentage() {
        return percentage;
    }

    public void setPercentage(int percentage) {
        this.percentage = percentage;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public void setTime(long time) {
        this.time = time;
    }

    public long getTime() {
        return time;
    }
}

```

```

157:F:\git\coin\nuls\nuls-1.1.3\nuls\client-
module\client\src\main\java\io\nuls\client\rpc\resources\dto\VersionDto.java
*/

```

```

package io.nuls.client.rpc.resources.dto;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "versionJSON")

```

```
public class VersionDto {

    @ApiModelProperty(name = "myVersion", value = "")
    private String myVersion;

    @ApiModelProperty(name = "newestVersion", value = "")
    private String newestVersion;

    @ApiModelProperty(name = "upgradeable", value = "")
    private boolean upgradeable;

    @ApiModelProperty(name = "infomation", value = "")
    private String infomation;

    @ApiModelProperty(name = "networkVersion", value = "")
    private Integer networkVersion;

    public String getMyVersion() {
        return myVersion;
    }

    public void setMyVersion(String myVersion) {
        this.myVersion = myVersion;
    }

    public String getNewestVersion() {
        return newestVersion;
    }

    public void setNewestVersion(String newestVersion) {
        this.newestVersion = newestVersion;
    }

    public boolean isUpgradeable() {
        return upgradeable;
    }

    public void setUpgradeable(boolean upgradeable) {
        this.upgradeable = upgradeable;
    }

    public String getInfomation() {
```

```

        return infromation;
    }

    public void setInfromation(String infromation) {
        this.infromation = infromation;
    }

    public Integer getNetworkVersion() {
        return networkVersion;
    }

    public void setNetworkVersion(Integer networkVersion) {
        this.networkVersion = networkVersion;
    }
}

158:F:\git\coin\nuls\nuls-1.1.3\nuls\client-
module\client\src\main\java\io\nuls\client\rpc\resources\model\JarSig.java
*/

package io.nuls.client.rpc.resources.model;

/**
 * @author: Niels Wang
 */
public class JarSig {
    private String fileName;
    private String sig;

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public String getSig() {
        return sig;
    }

    public void setSig(String sig) {

```

```
        this.sig = sig;
    }
}
```

159:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\resources\model\VersionFile.java  
\*/

```
package io.nuls.client.rpc.resources.model;
```

```
import java.util.ArrayList;
import java.util.List;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class VersionFile {
    private String version;
    private String binSig;
    private String confSig;
    private String exeSig;
    private String information;
    private List<JarSig> jarSigList = new ArrayList<>();
```

```
    public String getBinSig() {
        return binSig;
    }
```

```
    public void setBinSig(String binSig) {
        this.binSig = binSig;
    }
```

```
    public String getConfSig() {
        return confSig;
    }
```

```
    public void setConfSig(String confSig) {
        this.confSig = confSig;
    }
```

```
    public String getExeSig() {
        return exeSig;
```

```

    }

    public void setExeSig(String exeSig) {
        this.exeSig = exeSig;
    }

    public List<JarSig> getJarSigList() {
        return jarSigList;
    }

    public void setJarSigList(List<JarSig> jarSigList) {
        this.jarSigList = jarSigList;
    }

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }

    public synchronized void addJarSig(JarSig jarSig) {
        this.jarSigList.add(jarSig);
    }

    public String getInformation() {
        return information;
    }

    public void setInformation(String information) {
        this.information = information;
    }
}

```

160:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\resources\SystemResource.java  
\*/

```
package io.nuls.client.rpc.resources;
```

```
import io.nuls.client.storage.LanguageService;
```



```
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.i18n.I18nUtils;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.RpcClientResult;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
```

```
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.util.HashMap;
import java.util.Map;
```

```
/**
 * @author: Niels Wang
 */
@Path("/sys")
@Api(value = "/sys", description = "System")
@Component
public class SystemResource {

    @Autowired
    private LanguageService languageService;

    @PUT
    @Path("/lang/{language}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    public RpcClientResult setLanguage(@PathParam("language") String language) {
        AssertUtil.canNotEmpty(language);
        boolean b = I18nUtils.hasLanguage(language);
        if(!b){
            return Result.getFailed(KernelErrorCode.DATA_ERROR).toRpcClientResult();
        }
        try {
```

```

        l18nUtils.setLanguage(language);
        languageService.saveLanguage(language);
    } catch (NulsException e) {
        Log.error(e);
        return Result.getFailed(e.getErrorCode()).toRpcClientResult();
    }
    Map<String, Boolean> map = new HashMap<>();
    map.put("value", true);
    return Result.getSuccess().setData(map).toRpcClientResult();
}
}

```

161:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\resources\thread\ShutdownHook.java  
\*/

```
package io.nuls.client.rpc.resources.thread;
```

```
import io.nuls.client.rpc.resources.util.FileUtil;
import io.nuls.client.version.SyncVersionRunner;
import io.nuls.core.tools.log.Log;
```

```
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
```

```
/**
 * @author: Niels Wang
 */
```

```
public class ShutdownHook extends Thread {
```

```
    @Override
```

```
    public void run() {
        String root = this.getClass().getClassLoader().getResource("").getPath();
        try {
            root = URLDecoder.decode(root, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            Log.error(e);
        }
        String version = SyncVersionRunner.getInstance().getNewestVersion();
    }
}

```

```

String newDirPath = root + "/temp/" + version;
File tempDir = new File(newDirPath);
if (tempDir.exists()) {
    Log.error(1 + "");
    FileUtil.deleteFolder(root + "/bin");
    Log.error(2 + "");
    FileUtil.deleteFolder(root + "/conf");
    Log.error(3 + "");
    FileUtil.deleteFolder(root + "/libs");
    Log.error(4 + "");
    FileUtil.decompress(newDirPath + "/bin.zip", newDirPath);
    FileUtil.copyFolder(new File(newDirPath + "/bin"), new File(root + "/bin"));
    Log.error(5 + "");
    FileUtil.decompress(newDirPath + "/conf.zip", newDirPath);
    String os = System.getProperty("os.name").toUpperCase();
    FileUtil.copyFolder(new File(newDirPath + "/conf"), new File(root + "/conf"));
    Log.error(6 + "");
    FileUtil.copyFolder(new File(newDirPath + "/libs"), new File(root + "/libs"));

}
String os = System.getProperty("os.name").toUpperCase();
if (os.startsWith("WINDOWS")) {
    try {
        Runtime.getRuntime().exec("NULS-Wallet.exe");
    } catch (IOException e) {
        Log.error(e);
    }
} else if (os.startsWith("MAC")) {
    try {
        Runtime.getRuntime().exec("open -a NULSWallet");
    } catch (IOException e) {
        Log.error(e);
    }
} else {
    try {
        Runtime.getRuntime().exec("sh start.sh", null, new File(root + "/bin"));
    } catch (IOException e) {
        Log.error(e);
    }
}
}

```

```
}
```

```
162:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\rpc\resources\thread\UpgradeThread.java  
*/
```

```
package io.nuls.client.rpc.resources.thread;
```

```
import io.nuls.client.rpc.resources.dto.UpgradeProcessDTO;  
import io.nuls.client.rpc.resources.model.JarSig;  
import io.nuls.client.rpc.resources.model.VersionFile;  
import io.nuls.client.rpc.resources.util.FileUtil;  
import io.nuls.client.version.SyncVersionRunner;  
import io.nuls.client.version.constant.VersionConstant;  
import io.nuls.core.tools.crypto.Hex;  
import io.nuls.core.tools.crypto.Sha256Hash;  
import io.nuls.core.tools.io.HttpDownloadUtils;  
import io.nuls.core.tools.json.JSONUtils;  
import io.nuls.core.tools.log.Log;  
import io.nuls.core.tools.str.VersionUtils;  
import io.nuls.kernel.cfg.NulsConfig;  
import io.nuls.kernel.func.TimeService;
```

```
import java.io.*;  
import java.net.URLDecoder;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class UpgradeThread implements Runnable {
```

```
    private static final UpgradeThread INSTANCE = new UpgradeThread();
```

```
    private UpgradeThread() {  
    }
```

```
    public static UpgradeThread getInstance() {  
        return INSTANCE;  
    }
```

```

private Lock lock = new ReentrantLock();

private boolean upgrading = false;
private UpgradeProcessDTO process;
private String version;

@Override
public void run() {
    if (!upgrading) {
        return;
    }
    try {
        SyncVersionRunner syncor = SyncVersionRunner.getInstance();
        this.version = syncor.getNewestVersion();
        String versionFileHash = syncor.getVersionFileHash();
        if (!VersionUtils.higherThan(version, NulsConfig.VERSION)) {
            setFailedMessage("The version is wrong!");
            return;
        }
        process.setPercentage(1);
        VersionFile versionJson = getVersionJson(syncor, versionFileHash);
        if (null == versionJson) {
            return;
        }
        process.setPercentage(5);
        process.setStatus(VersionConstant.DOWNLOADING);
        String root = this.getClass().getClassLoader().getResource("").getPath();
        root = URLDecoder.decode(root, "UTF-8");
        String newDirPath = root + "/temp/" + version;
        File newDir = new File(newDirPath);
        if (!newDir.exists()) {
            newDir.mkdirs();
        }
        String urlRoot = syncor.getRootUrl() + "/" + version;
        boolean result = this.download(urlRoot + "/bin.zip", newDirPath + "/bin.zip",
versionJson.getBinSig());
        if (!result) {
            deleteTemp(root + "/temp/");
            return;
        }
        process.setPercentage(12);
        result = this.download(urlRoot + "/conf.zip", newDirPath + "/conf.zip",

```

```

versionJson.getConfSig());
    if (!result) {
        deleteTemp(root + "/temp/");
        return;
    }
    process.setPercentage(15);
    File libsDir = new File(newDirPath + "/libs");
    if (libsDir.exists()) {
        FileUtil.deleteFolder(libsDir);
    }
    libsDir.mkdirs();
    process.setPercentage(20);
    int count = 0;
    int size = versionJson.getJarSigList().size();
    for (JarSig jarSig : versionJson.getJarSigList()) {
        File file = new File(root + "/libs/" + jarSig.getFileName());
        if (file.exists() && this.verifySig(file, Hex.decode(jarSig.getSig()))) {
            FileUtil.copyFile(file, new File(newDirPath + "/libs/" + jarSig.getFileName()));
            result = true;
        } else {
            result = this.download(urlRoot + "/libs/" + jarSig.getFileName(), newDirPath + "/libs/" +
jarSig.getFileName(), jarSig.getSig());
        }
        if (!result) {
            deleteTemp(root + "/temp/");
            return;
        }
        count++;
        process.setPercentage(20 + (count * 70) / (size));
    }
    process.setStatus(VersionConstant.INSTALLING);
    String oldDirPath = root + "/temp/old";
    result = this.copy(root + "/bin", oldDirPath + "/bin");
    if (!result) {
        deleteTemp(root + "/temp/");
        return;
    }
    process.setPercentage(92);
    result = this.copy(root + "/conf", oldDirPath + "/conf");
    if (!result) {
        deleteTemp(root + "/temp/");
        return;
    }

```

```

    }
    process.setPercentage(95);
    result = this.copy(root + "/libs", oldDirPath + "/libs");
    if (!result) {
        deleteTemp(root + "/temp/");
        return;
    }
    process.setPercentage(100);
    process.setStatus(VersionConstant.WAITING_RESTART);
} catch (Exception e) {
    Log.error(e);
    setFailedMessage(e.getMessage());
    upgrading = false;
}
}

```

```

private void rollbackAndDeleteTemp(String root) throws UnsupportedOperationException {
    File rootBin = new File(root + "/bin");
    FileUtil.deleteFolder(rootBin);
    File rootConf = new File(root + "/conf");
    FileUtil.deleteFolder(rootConf);
    File rootLibs = new File(root + "/libs");
    FileUtil.deleteFolder(rootLibs);

    FileUtil.copyFolder(new File(root + "/temp/old/bin"), rootBin);
    FileUtil.copyFolder(new File(root + "/temp/old/conf"), rootConf);
    FileUtil.copyFolder(new File(root + "/temp/old/libs"), rootLibs);

    FileUtil.deleteFolder(new File(root + "/temp/"));
}

```

```

private void deleteTemp(String tempPath) {
    FileUtil.deleteFolder(new File(tempPath));
}

```

```

private VersionFile getVersionJson(SyncVersionRunner syncor, String versionFileHash) {
    if (!upgrading) {
        setFailedMessage("The upgrade has stopped");
        return null;
    }
    String jsonStr = null;

```

```

try {
    byte[] bytes = HttpDownloadUtils.download(syncor.getRootUrl() + "/" + version +
"/version.json");
    String hash = Hex.encode(Sha256Hash.hash(bytes));
    if (!hash.equals(versionFileHash)) {
        setFailedMessage("Signature verification is incorrect:" + version + "/version.json");
        return null;
    }
    jsonStr = new String(bytes, NulsConfig.DEFAULT_ENCODING);
} catch (IOException e) {
    Log.error(e);
    setFailedMessage("Download version json failed!");
    return null;
}
try {
    return JSONUtils.json2pojo(jsonStr, VersionFile.class);
} catch (Exception e) {
    Log.error(e);
    setFailedMessage("Parse version json failed!");
    return null;
}
}

```

```

private boolean download(String url, String filePath, String signature) throws IOException {
    if (!upgrading) {
        setFailedMessage("The upgrade has stopped");
        return false;
    }
    byte[] bytes = HttpDownloadUtils.download(url);
    if (!verifySig(bytes, Hex.decode(signature))) {
        setFailedMessage("Signature verification is incorrect:" + url);
        return false;
    }
    boolean result = FileUtil.writeFile(bytes, filePath);
    if (!result) {
        setFailedMessage("Write file failed:" + filePath);
    }
    return result;
}

```

```

private boolean copy(String filePath, String targetPath) {
    if (!upgrading) {

```



```

        setFailedMessage("The upgrade has stopped");
        return false;
    }
    File source = new File(filePath);
    File target = new File(targetPath);
    if (source.isFile()) {
        FileUtil.copyFile(source, target);
    } else {
        FileUtil.copyFolder(source, target);
    }
    return true;
}

```

```

public boolean start() {
    lock.lock();
    try {
        if (!upgrading) {
            this.process = new UpgradeProcessDTO();
            this.upgrading = true;
            return true;
        }
    } finally {
        lock.unlock();
    }
    return false;
}

```

```

public boolean stop() {
    lock.lock();
    try {
        if (upgrading) {
            upgrading = false;
        }
        String root = UpgradeThread.class.getClassLoader().getResource("").getPath();
        deleteTemp(root + "/temp/");
        return true;
    } finally {
        lock.unlock();
    }
}

```

```

public boolean isUpgrading() {

```

```

        lock.lock();
        try {
            return upgrading;
        } finally {
            lock.unlock();
        }
    }

    public UpgradeProcessDTO getProcess() {
        if (!isUpgrading()) {
            return new UpgradeProcessDTO();
        }
        return process;
    }

    private void setFailedMessage(String message) {
        process.setStatus(VersionConstant.FAILED);
        process.setMessage(message);
        process.setTime(TimeService.currentTimeMillis());
    }

```

```

    private boolean verifySig(byte[] bytes, byte[] sig) {
        byte[] hash = Sha256Hash.hash(bytes);
        return VersionConstant.EC_KEY.verify(hash, sig);
    }

```

```

    private boolean verifySig(File file, byte[] sig) throws IOException {
        InputStream input = new FileInputStream(file);
        byte[] bytes;
        try {
            bytes = new byte[input.available()];
            input.read(bytes);
        } finally {
            input.close();
        }
        byte[] hash = Sha256Hash.hash(bytes);
        return VersionConstant.EC_KEY.verify(hash, sig);
    }
}

```

163:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\main\java\io\nuls\client\rpc\resources\util\FileUtil.java

```
*/
```

```
package io.nuls.client.rpc.resources.util;
```

```
import io.nuls.core.tools.log.Log;
```

```
import java.io.*;
```

```
import java.net.URLDecoder;
```

```
import java.nio.channels.FileChannel;
```

```
import java.util.zip.ZipEntry;
```

```
import java.util.zip.ZipInputStream;
```

```
import java.util.zip.ZipOutputStream;
```

```
/**
```

```
 * @author: Niels Wang
```

```
*/
```

```
public final class FileUtil {
```

```
    public static File compress(File source, File target) {
```

```
        if (target.exists()) {
```

```
            target.delete();
```

```
        }
```

```
        FileOutputStream fos = null;
```

```
        ZipOutputStream zos = null;
```

```
        try {
```

```
            fos = new FileOutputStream(target);
```

```
            zos = new ZipOutputStream(new BufferedOutputStream(fos));
```

```
            addEntry("/", source, zos);
```

```
        } catch (IOException e) {
```

```
            Log.error(e);
```

```
        } finally {
```

```
            close(zos, fos);
```

```
        }
```

```
        return target;
```

```
    }
```

```
    private static void addEntry(String dir, File source, ZipOutputStream zos) {
```

```
        String entry = dir + source.getName();
```

```
        if (source.isDirectory()) {
```

```
            for (File file : source.listFiles()) {
```

```
                addEntry(entry + "/", file, zos);
```

```
            }
```

```

    } else {
        FileInputStream fis = null;
        BufferedInputStream bis = null;
        try {
            fis = new FileInputStream(source);
            byte[] buffer = new byte[fis.available()];
            if (buffer.length == 0) {
                return;
            }
            bis = new BufferedInputStream(fis, buffer.length);
            int size;
            zos.putNextEntry(new ZipEntry(entry));
            while ((size = bis.read(buffer, 0, buffer.length)) != -1) {
                zos.write(buffer, 0, size);
            }
            zos.closeEntry();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            close(bis, fis);
        }
    }
}

```

```

public static void decompress(String zipPath, String targetPath) {
    File source = null;
    try {
        source = new File(URLDecoder.decode(zipPath, "UTF-8"));
    } catch (UnsupportedEncodingException e) {
        Log.error(e);
    }
    if (null == source || !source.exists()) {
        return;
    }
    ZipInputStream zis = null;
    BufferedOutputStream bos = null;
    try {
        zis = new ZipInputStream(new FileInputStream(source));
        ZipEntry entry;
        while ((entry = zis.getNextEntry()) != null && !entry.isDirectory()) {
            File target = new File(URLDecoder.decode(targetPath, "UTF-8"), entry.getName());
            if (!target.getParentFile().exists()) {

```

```

        target.getParentFile().mkdirs();
    }
    bos = new BufferedOutputStream(new FileOutputStream(target));
    int size;
    byte[] buffer = new byte[zis.available()];
    while ((size = zis.read(buffer, 0, buffer.length)) != -1) {
        bos.write(buffer, 0, size);
    }
    bos.flush();
}
zis.closeEntry();
} catch (IOException e) {
    Log.error(e);
} finally {
    close(zis, bos);
}
}

```

```

public static void copyFile(File source, File target) {
    FileChannel in = null;
    FileChannel out = null;
    FileInputStream inStream = null;
    FileOutputStream outStream = null;
    try {
        inStream = new FileInputStream(source);
        outStream = new FileOutputStream(target);
        in = inStream.getChannel();
        out = outStream.getChannel();
        in.transferTo(0, in.size(), out);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (inStream != null) {
                inStream.close();
            }
            if (in != null) {
                in.close();
            }
            if (outStream != null) {
                outStream.close();
            }
        }
    }
}

```

```

        if (out != null) {
            out.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

```

private static void close(Closeable... closeables) {
    if (closeables == null) {
        return;
    }
    try {
        for (Closeable closeable : closeables) {
            if (closeable != null) {
                closeable.close();
            }
        }
    } catch (IOException e) {
        Log.error(e);
    }
}

```

```

public static void copyFolder(File source, File target) {
    String[] filePath = source.list();

    if (null == filePath || filePath.length == 0) {
        return;
    }

    if (!target.exists()) {
        target.mkdirs();
    }

    String sourcePath = source.getPath();
    String path = target.getPath();
    try {
        sourcePath = URLDecoder.decode(sourcePath, "UTF-8");
        path = URLDecoder.decode(path, "UTF-8");
    } catch (Exception e) {
        Log.error(e);
    }
}

```

```

    }
    for (int i = 0; i < filePath.length; i++) {
        if ((new File(sourcePath + "/" + filePath[i])).isDirectory()) {
            copyFolder(new File(sourcePath + "/" + filePath[i]), new File(path + "/" + filePath[i]));
        }

        if (new File(sourcePath + "/" + filePath[i]).isFile()) {
            copyFile(new File(sourcePath + "/" + filePath[i]), new File(path + "/" + filePath[i]));
        }
    }
}

```

```

public static void writeText(String text, String path) throws IOException {
    FileWriter writer = null;
    try {
        File file = new File(path);
        if (file.exists()) {
            file.delete();
        }
        writer = new FileWriter(file);
        writer.write(text);
        writer.flush();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        writer.close();
    }
}

```

```

public static boolean writeFile(byte[] bytes, String filePath) throws IOException {
    File file = new File(URLDecoder.decode(filePath, "UTF-8"));
    if (file.exists()) {
        file.delete();
    }
}

```

```

OutputStream output = new FileOutputStream(file);
try {
    output.write(bytes);
    output.flush();
} catch (IOException e) {
}

```

```

        Log.error(e);
        return false;
    } finally {
        output.close();
    }
    return true;
}

```

```

public static boolean deleteFolder(File folder) {
    if (!folder.exists()) {
        return true;
    }
    File[] files = folder.listFiles();
    for (File file : files) {
        if (file.isFile()) {
            try {
                boolean b = file.delete();
                if (!b) {
                    Log.info("delete " + file.getName() + " result:" + b);
                    mkNullToFile(file);
                }
            } catch (Exception e) {
                Log.error(e);
            }
        } else {
            deleteFolder(file);
        }
    }
    try {
        boolean b = folder.delete();
    } catch (Exception e) {
        Log.error(e);
    }
    return true;
}

```

```

private static void mkNullToFile(File file) {
    byte[] bytes = new byte[0];
    OutputStream outputStream = null;
    try {

        outputStream = new FileOutputStream(file);
    }
}

```



```

        outputStream.write(bytes);
        outputStream.flush();
    } catch (FileNotFoundException e) {
        Log.error(e);
    } catch (IOException e) {
        Log.error(e);
    } finally {
        try {
            if (null != outputStream) {
                outputStream.close();
            }
        } catch (Exception e) {
            Log.error(e);
        }
    }
}

}

public static void deleteFolder(String path) {
    deleteFolder(new File(path));
}
}

```

164:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\main\java\io\nuls\client\rpc\RpcServerManager.java  
\*/

```

package io.nuls.client.rpc;

import io.nuls.core.tools.log.Log;
import org.glassfish.grizzly.http.server.*;
import org.glassfish.grizzly.nio.transport.TCPNIOTransport;
import org.glassfish.grizzly.servlet.WebappContext;
import org.glassfish.grizzly.strategies.WorkerThreadIOStrategy;
import org.glassfish.grizzly.threadpool.ThreadPoolConfig;
import org.glassfish.grizzly.utils.Charsets;
import org.glassfish.jersey.internal.guava.ThreadFactoryBuilder;
import org.glassfish.jersey.servlet.ServletContainer;

import javax.servlet.ServletRegistration;
import javax.ws.rs.core.UriBuilder;
import java.io.IOException;

```

```

import java.net.URI;
import java.util.Map;

/**
 * @author: Niels Wang
 */
public class RpcServerManager {

    private static final RpcServerManager INSTANCE = new RpcServerManager();

    private HttpServer httpServer;

    private RpcServerManager() {
    }

    public static RpcServerManager getInstance() {
        return INSTANCE;
    }

    public void startServer(String ip, int port) {
        URI serverURI = UriBuilder.fromUri("http://" + ip).port(port).build();
        // Create test web application context.
        WebappContext webappContext = new WebappContext("NULS-RPC-SERVER", "/api");

        ServletRegistration servletRegistration = webappContext.addServlet("jersey-servlet",
ServletContainer.class);
        servletRegistration.setInitParameter("javax.ws.rs.Application",
"io.nuls.client.rpc.config.NulsResourceConfig");
        servletRegistration.addMapping("/api/*");

        httpServer = new HttpServer();
        NetworkListener listener = new NetworkListener("grizzly2", ip, port);
        TCPNIOTransport transport = listener.getTransport();
        ThreadPoolConfig workerPool = ThreadPoolConfig.defaultConfig()
            .setCorePoolSize(4)
            .setMaxPoolSize(4)
            .setQueueLimit(1000)
            .setThreadFactory((new ThreadFactoryBuilder()).setNameFormat("grizzly-http-server-
%d").build());
        transport.configureBlocking(false);
        transport.setSelectorRunnersCount(2);
        transport.setWorkerThreadPoolConfig(workerPool);
    }
}

```

```

transport.setIOStrategy(WorkerThreadIOStrategy.getInstance());
transport.setTcpNoDelay(true);
listener.setSecure(false);
httpServer.addListener(listener);

ServerConfiguration config = httpServer.getServerConfiguration();
config.setDefaultQueryEncoding(Charsets.UTF8_CHARSET);

webappContext.deploy(httpServer);

try {
    ClassLoader loader = this.getClass().getClassLoader();

    addSwaggerUi(loader);
    addClientUi(loader);

    httpServer.start();
    Log.info("http restFul server is started!url is " + serverURI.toString());
} catch (IOException e) {
    Log.error(e);
    httpServer.shutdownNow();
}
}

private void addClientUi(ClassLoader loader) {
    CLStaticHttpHandler docsHandler = new CLStaticHttpHandler(loader, "client-web/");
    docsHandler.setFileCacheEnabled(true);
    ServerConfiguration cfg = httpServer.getServerConfiguration();
    cfg.addHttpHandler(docsHandler, "/");
}

private void addSwaggerUi(ClassLoader loader) {
    CLStaticHttpHandler docsHandler = new CLStaticHttpHandler(loader, "swagger-ui/");
    docsHandler.setFileCacheEnabled(false);
    ServerConfiguration cfg = httpServer.getServerConfiguration();
    cfg.addHttpHandler(docsHandler, "/docs/");
}

public void shutdown() {
    Map<HttpHandler, HttpHandlerRegistration[]> mapping =
httpServer.getServerConfiguration().getHttpHandlersWithMapping();
    for (HttpHandler handler : mapping.keySet()) {

```

```

        handler.destroy();
    }
    httpServer.shutdown();
}

public boolean isStarted() {
    if (null == this.httpServer) {
        return false;
    }
    return this.httpServer.isStarted();
}
}

```

165:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\main\java\io\nuls\client\storage\impl\LanguageServiceImpl.java  
 \*/

```
package io.nuls.client.storage.impl;
```

```

import io.nuls.client.constant.CommandConstant;
import io.nuls.client.storage.LanguageService;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Result;

```

```
/**
```

```
 * @author: Charlie
```

```
 */
```

```
@Service
```

```
public class LanguageServiceImpl implements LanguageService, InitializingBean {
```

```
/**
```

```
 *
```

```
 * Universal data storage services.
```

```
 */
```

```
@Autowired
```

```
private DBService dbService;
```

```
@Override
```

```

public void afterPropertiesSet() throws NulsException {
    Result result = this.dbService.createArea(CommandConstant.DB_LANGUAGE);
    if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
        throw new NulsRuntimeException(result.getErrorCode());
    }
}

```

```

@Override
public Result saveLanguage(String language) {
    return dbService.putModel(CommandConstant.DB_LANGUAGE,
CommandConstant.DB_LANGUAGE.getBytes(), language);
}

```

```

@Override
public Result getLanguage() {
    return
Result.getSuccess().setData(dbService.getModel(CommandConstant.DB_LANGUAGE,
CommandConstant.DB_LANGUAGE.getBytes()));
}
}

```

166:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\storage\LanguageService.java  
\*/

```
package io.nuls.client.storage;
```

```
import io.nuls.kernel.model.Result;
```

```

/**
 *
 * @author: Charlie
 */

```

```

public interface LanguageService {

    Result saveLanguage(String language);

    Result getLanguage();

}

```

167:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\version\constant\VersionConstant.java

```
*/
```

```
package io.nuls.client.version.constant;
```

```
import io.nuls.core.tools.crypto.ECKey;
```

```
import io.nuls.core.tools.crypto.Hex;
```

```
/**
```

```
 * @author: Niels Wang
```

```
*/
```

```
public interface VersionConstant {
```

```
    String PUBLIC_KEY =
```

```
"043f48de189fe5c01c7cd746cfdc404ab1957a287ef46fe23cb23e7ff6108f188eaaa89ce8e248854a  
809c187e9d207c881b126f0874183c0c81efe4293e6b1db7";
```

```
    ECKey EC_KEY = ECKey.fromPublicOnly(Hex.decode(PUBLIC_KEY));
```

```
/**
```

```
 * 0,1,2,3,4
```

```
*/
```

```
int UN_START = 0;
```

```
int DOWNLOADING = 1;
```

```
int INSTALLING = 2;
```

```
int WAITING_RESTART = 3;
```

```
int FAILED = 4;
```

```
}
```

```
168:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\version\SyncVersionRunner.java
```

```
*/
```

```
package io.nuls.client.version;
```

```
import io.nuls.client.rpc.resources.thread.UpgradeThread;
```

```
import io.nuls.client.rpc.resources.util.FileUtil;
```

```
import io.nuls.client.version.constant.VersionConstant;
```

```
import io.nuls.core.tools.crypto.Hex;
```

```
import io.nuls.core.tools.crypto.Sha256Hash;
```

```
import io.nuls.core.tools.io.HttpDownloadUtils;
```

```
import io.nuls.core.tools.json.JSONUtils;
```

```

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;

import java.io.File;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.util.Map;

/**
 * @author: Niels Wang
 */
public class SyncVersionRunner implements Runnable {
    /**
     *
     * The latest version of the current software obtained from the network.
     */
    private static String NEWEST_VERSION;
    private static String VERSION_FILE_HASH;
    private static String INFORMATION;

    private String rootUrl = NulsConfig.MODULES_CONFIG.getCfgValue("client", "version.root.url",
"https://raw.githubusercontent.com/nuls-io/nuls-wallet-release/master/test/release/");
    private String versionJsonUrl = rootUrl + "version.json";

    private boolean first = true;

    private static final SyncVersionRunner INSTANCE = new SyncVersionRunner();

    private SyncVersionRunner() {

    }

    public static SyncVersionRunner getInstance() {
        return INSTANCE;
    }

    @Override
    public void run() {

```

```

try {
    syncNewestVersionInfo();
    if (!first) {
        checkLocalTempFiles();
    }
    first = false;
} catch (Exception e) {
    Log.error(e);
}
}

```

```

private void checkLocalTempFiles() {
    if (!UpgradeThread.getInstance().isUpgrading()) {
        String root = SyncVersionRunner.class.getClassLoader().getResource("").getPath();
        try {
            root = URLDecoder.decode(root, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            Log.error(e);
        }
        File file = new File(root + "/temp");
        if (file.exists()) {
            FileUtil.deleteFolder(file);
        }
    }
}

```

```

private void syncNewestVersionInfo() throws NulsException, UnsupportedEncodingException {
    String jsonStr = null;
    try {
        jsonStr = new String(HttpDownloadUtils.download(this.versionJsonUrl),
NulsConfig.DEFAULT_ENCODING);
    } catch (IOException e) {
        throw new NulsException(KernelErrorCode.DOWNLOAD_VERSION_FAILED);
    }
    Map<String, Object> map = null;
    try {
        map = JSONUtils.json2map(jsonStr);
    } catch (Exception e) {
        throw new NulsException(KernelErrorCode.PARSE_JSON_FAILED);
    }
    String version = (String) map.get("version");
    String versionFileHash = (String) map.get("versionFileHash");
}

```



```

        String signature = (String) map.get("signature");
        boolean result = VersionConstant.EC_KEY.verify(Sha256Hash.hash((version + "& " +
versionFileHash).getBytes("UTF-8")), Hex.decode(signature));
        if (!result) {
            return;
        }
        NEWEST_VERSION = version;
        VERSION_FILE_HASH = versionFileHash;
        INFORMATION = (String) map.get("information");
    }

    public String getNewestVersion() {
        if (null == NEWEST_VERSION) {
            try {
                syncNewestVersionInfo();
            } catch (Exception e) {
                Log.error(e);
            }
        }
        return NEWEST_VERSION;
    }

    public String getVersionFileHash() {
        return VERSION_FILE_HASH;
    }

    public String getInformation() {
        return INFORMATION;
    }

    public String getRootUrl() {
        return rootUrl;
    }

    public String getVersionJsonUrl() {
        return versionJsonUrl;
    }
}

```

169:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\version\WalletVersionManager.java  
\*/

```
package io.nuls.client.version;
```

```
import io.nuls.kernel.thread.manager.NulsThreadFactory;
```

```
import io.nuls.kernel.thread.manager.TaskManager;
```

```
import java.util.concurrent.ScheduledThreadPoolExecutor;
```

```
import java.util.concurrent.TimeUnit;
```

```
/**
```

```
 * @author: Niels Wang
```

```
 */
```

```
public class WalletVersionManager {
```

```
    public static void start() {
```

```
        ScheduledThreadPoolExecutor executor = TaskManager.createScheduledThreadPool(1,  
new NulsThreadFactory((short) 1, "version-manager"));
```

```
        executor.scheduleAtFixedRate(SyncVersionRunner.getInstance(), 0, 10,  
TimeUnit.MINUTES);
```

```
    }
```

```
}
```

```
170:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\main\java\io\nuls\client\web\view\WebViewBootstrap.java
```

```
*/
```

```
package io.nuls.client.web.view;
```

```
import io.nuls.client.rpc.constant.RpcConstant;
```

```
import io.nuls.kernel.cfg.NulsConfig;
```

```
import javafx.application.Application;
```

```
import javafx.application.Platform;
```

```
import javafx.scene.image.Image;
```

```
import javafx.stage.Stage;
```

```
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```
import java.lang.reflect.Method;
```

```

import java.net.URL;

/**
 *
 * @author In
 */
public class WebViewBootstrap extends Application implements Runnable, ActionListener {

    private static final Logger log = LoggerFactory.getLogger(WebViewBootstrap.class);

    private final static String TRAY_ICON = "/image/tray.png";
    private static final String APP_ICON = "/image/logo.png";
    private static final String APP_TITLE = "NULS";

    private boolean hideTip;

    private TrayIcon trayIcon;
    private Stage stage;

    @Override
    public void run() {
        startWebView(null);
    }

    // /**
    // *
    // */
    public void startWebView(String[] args) {
        String os = System.getProperty("os.name").toUpperCase();
        if (!os.startsWith("WINDOWS") && !os.startsWith("MAC OS")) {
            return;
        }
        launch(args);
    }

    // /**
    // *
    // */
    @Override
    public void start(final Stage stage) throws Exception {

```

```

this.stage = stage;

//
stage.setTitle(APP_TITLE);

stage.setResizable(false);

//
stage.getIcons().add(new Image(getClass().getResourceAsStream(TRAY_ICON)));
if (isMac()) {
    java.awt.Image dockIcon = new
ImageIcon(getClass().getResource(APP_ICON)).getImage();
    try {
        Class<?> cls = Class.forName("com.apple.eawt.Application");
        Object application =
cls.newInstance().getClass().getMethod("getApplication").invoke(null);
        application.getClass().getMethod("setDockIconImage",
java.awt.Image.class).invoke(application, dockIcon);
    } catch (Exception e) {
    }

} else {
    //
    stage.setTitle(APP_TITLE);
    //
    stage.getIcons().add(new Image(getClass().getResourceAsStream(APP_ICON)));
}

//
initSystemTray();

openBrowse();

}

// /**
// *
// */
@Override
public void stop() throws Exception {
    System.exit(0);
}

```

```

private boolean isMac() {
    String osName = System.getProperty("os.name").toLowerCase();
    return osName.indexOf("mac") != -1;
}

/*
 *
 */
private void initSystemTray() {
    //
    if (SystemTray.isSupported()) {
        //
        URL resource = this.getClass().getResource(APP_ICON);
        trayIcon = new TrayIcon(new ImageIcon(resource).getImage(), "NULS", createMenu());
        //
        trayIcon.setActionCommand("db_click_tray");
        //
        trayIcon.addActionListener(this);
        //
        trayIcon.setImageAutoSize(true);

        SystemTray sysTray = SystemTray.getSystemTray();
        try {
            sysTray.add(trayIcon);
        } catch (AWTException e) {
            log.error(e.getMessage(), e);
        }
    }
}

/*
 *
 */
private PopupMenu createMenu() {
    PopupMenu popupMenu = new PopupMenu(); //

    //.
    MenuItem itemShow = new MenuItem("Show Wallet");
    itemShow.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

```

```

        openBrowse();
    }
});

popupMenu.add(itemShow);
popupMenu.addSeparator();

//
MenuItem itemExit = new MenuItem("Exit");
popupMenu.add(itemExit);

//
itemExit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        exit();
    }
});

return popupMenu;
}

// /**
//  *
//  */
@Override
public void actionPerformed(ActionEvent e) {

    String command = e.getActionCommand();

    if ("db_click_tray".equals(command) && !stage.isIconified()) {
        //
        //false
        if (stage.isShowing()) {
            hide();
        } else {
            openBrowse();
        }
    }
}
}

```

```

    // /**
    //  *
    //  */
    public void openBrowse() {
        String ip =
NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
RpcConstant.CFG_RPC_SERVER_IP, RpcConstant.DEFAULT_IP);
        int port = NulsConfig.MODULES_CONFIG.getCfgValue(RpcConstant.CFG_RPC_SECTION,
RpcConstant.CFG_RPC_SERVER_PORT, RpcConstant.DEFAULT_PORT);
        if("0.0.0.0".equals(ip)){
            ip = RpcConstant.DEFAULT_IP;
        }
        String url = "http://" + ip + ":" + port;
        openURL(url);
    }

    public static void openURL(String url) {
        try {
            browse(url);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void browse(String url) throws Exception {
        //
        String osName = System.getProperty("os.name", "");
        if (osName.startsWith("Mac OS")) {
            //
            Class fileMgr = Class.forName("com.apple.eio.FileManager");
            Method openURL = fileMgr.getDeclaredMethod("openURL",
                new Class[]{String.class});
            openURL.invoke(null, new Object[]{url});
        } else if (osName.startsWith("Windows")) {
            // windows
            Runtime.getRuntime().exec(
                "rundll32 url.dll,FileProtocolHandler " + url);
        } else {
            // Unix or Linux
            String[] browsers = {"firefox", "opera", "konqueror", "epiphany",
                "mozilla", "netscape"};
            String browser = null;

```

```

for (int count = 0; count < browsers.length && browser == null; count++) {
    // browser
    // ==0
    if (Runtime.getRuntime()
        .exec(new String[]{"which", browsers[count]})
        .waitFor() == 0) {
        browser = browsers[count];
    }
}
if (browser == null) {
    throw new Exception("Could not find web browser");
} else {
    //
    Runtime.getRuntime().exec(new String[]{browser, url});
}
}
}

```

```

// /**
//  *
//  */
public void hide() {
    Platform.runLater(new Runnable() {
        @Override
        public void run() {
            stage.hide();
        }
    });
}

// /**
//  *
//  */
public void exit() {
    SystemTray.getSystemTray().remove(trayIcon);
    Platform.exit();
}
}

```

171:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\test\java\io\nuls\client\rpc\resources\thread\UpgradeThreadTest.java



```

*/

package io.nuls.client.rpc.resources.thread;

import io.nuls.client.rpc.resources.util.FileUtil;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * @author: Niels Wang
 */
public class UpgradeThreadTest {

    @Test
    public void download() {
//      UpgradeThread service = UpgradeThread.getInstance();
//      service.start();
//      service.run();
//      FileUtil.deleteFolder("C:\\Users\\Administrator\\Desktop\\release\\NULS-Wallet-0.9.10.5-
windows-x64\\conf");
//      FileUtil.deleteFolder("C:\\Users\\Administrator\\Desktop\\release\\NULS-Wallet-0.9.10.5-
windows-x64\\libs");
//      System.out.println("success");

        System.out.print("abcdefg");
        System.out.print("\b");
        System.out.print("\b");
        System.out.print("\b");

    }
}

```

```

172:F:\git\coin\nuls\nuls-1.1.3\nuls\client-
module\client\src\test\java\io\nuls\client\rpc\resources\util\FileUtilTest.java
*/

```

```

package io.nuls.client.rpc.resources.util;

import org.junit.Test;

import static org.junit.Assert.*;

```

```

/**
 * @author: Niels Wang
 * @date: 2018/7/1
 */
public class FileUtilTest {

    @Test
    public void deleteFolder() {
        FileUtil.deleteFolder("C:\\Users\\Administrator\\Desktop\\release\\NULS-Wallet-0.9.10-
windows-x64-for-test\\conf\\client-web");
    }
}

```

173:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\test\java\io\nuls\client\rpc\RpcServerManagerTest.java

```

*/

```

```

package io.nuls.client.rpc;

import io.nuls.kernel.MicroKernelBootstrap;
import org.junit.Test;

import static org.junit.Assert.*;

```

```

/**
 * @author: Niels Wang
 */
public class RpcServerManagerTest {

    public void startServer() {
        MicroKernelBootstrap bootstrap = MicroKernelBootstrap.getInstance();
        bootstrap.init();
        bootstrap.start();

        RpcServerManager.getInstance().startServer("127.0.0.1", 8080);
        assertTrue(true);
    }
}

```

174:F:\git\coin\nuls\nuls-1.1.3\nuls\client-

module\client\src\test\java\io\nuls\license\FileScanUtils.java

\*/

package io.nuls.license;

import io.nuls.core.tools.io.StringFileLoader;

import io.nuls.core.tools.log.Log;

import java.io.File;

import java.io.FileWriter;

/\*\*

\* @author: Niels Wang

\*/

public class FileScanUtils {

public static void main(String[] args) {

String dirPath = "C:\\workspace\\nuls";

File root = new File(dirPath);

scanFiles(root);

}

private static void scanFiles(File root) {

File[] files = root.listFiles();

for (File file : files) {

if (file.isDirectory()) {

scanFiles(file);

} else if (file.getName().endsWith(".java")) {

executeJavaFile(file);

}

}

}

private static void executeJavaFile(File javaFile) {

try {

String content = StringFileLoader.readRealPath(javaFile.getPath(), true);

if (content.startsWith("package io.nuls")) {

String newContent = getMitLicense() + "\n" + content;

FileWriter writer;

writer = new FileWriter(javaFile.getPath());

writer.write(newContent);

writer.flush();

```

        writer.close();

    }
} catch (Exception e) {
    Log.error(e);
}
}

private static String getMitLicense() {
    StringBuilder str = new StringBuilder();
    str.append("/");
    str.append("\n");
    str.append(" * MIT License");
    str.append("\n");
    str.append(" *");
    str.append("\n");
    str.append(" * Copyright (c) 2017-2018 nuls.io");
    str.append("\n");
    str.append(" *");
    str.append("\n");
    str.append(" * Permission is hereby granted, free of charge, to any person obtaining a copy");
    str.append("\n");
    str.append(" * of this software and associated documentation files (the \"Software\"), to deal");
    str.append("\n");
    str.append(" * in the Software without restriction, including without limitation the rights");
    str.append("\n");
    str.append(" * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell");
    str.append("\n");
    str.append(" * copies of the Software, and to permit persons to whom the Software is");
    str.append("\n");
    str.append(" * furnished to do so, subject to the following conditions:");
    str.append("\n");
    str.append(" *");
    str.append("\n");
    str.append(" * The above copyright notice and this permission notice shall be included in all");
    str.append("\n");
    str.append(" * copies or substantial portions of the Software.");
    str.append("\n");
    str.append(" *");
    str.append("\n");
    str.append(" * THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY
    KIND, EXPRESS OR");

```

```

        str.append("\n");
        str.append(" * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,");
        str.append("\n");
        str.append(" * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
NO EVENT SHALL THE");
        str.append("\n");
        str.append(" * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER");
        str.append("\n");
        str.append(" * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM,");
        str.append("\n");
        str.append(" * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE");
        str.append("\n");
        str.append(" * SOFTWARE.");
        str.append("\n");
        str.append(" *");
        str.append("\n");
        str.append(" */");
        return str.toString();
    }
}

```

175:F:\git\coin\nuls\nuls-1.1.3\nuls\client-  
module\client\src\test\java\io\nuls\test\network\TestNetwork.java  
\*/

```
package io.nuls.test.network;
```

```

import io.nuls.db.module.impl.LevelDbModuleBootstrap;
import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.message.bus.module.MessageBusModuleBootstrap;
import io.nuls.network.module.impl.NettyNetworkModuleBootstrap;
import io.nuls.protocol.base.module.BaseProtocolsModuleBootstrap;
import org.junit.Before;
import org.junit.Test;

```

```
public class TestNetwork {
```

```
    @Before
```

```

public void init() {
    try {
        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();

        LevelDbModuleBootstrap dbModuleBootstrap = new LevelDbModuleBootstrap();
        dbModuleBootstrap.init();
        dbModuleBootstrap.start();

        BaseProtocolsModuleBootstrap protocolsModuleBootstrap = new
BaseProtocolsModuleBootstrap();
        protocolsModuleBootstrap.init();
        protocolsModuleBootstrap.start();

        MessageBusModuleBootstrap messageBusModuleBootstrap = new
MessageBusModuleBootstrap();
        messageBusModuleBootstrap.init();
        messageBusModuleBootstrap.start();

        NettyNetworkModuleBootstrap networkModuleBootstrap = new
NettyNetworkModuleBootstrap();
        networkModuleBootstrap.init();
        networkModuleBootstrap.start();
    } catch (Exception e) {
        e.printStackTrace();
    }

}

@Test
public void testNetworkModule() {
    while (true) {
        try {
            Thread.sleep(1000000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
}

```

```
176:F:\git\coin\nuls\nuls-1.1.3\nuls\client-module\client\src\test\java\io\nuls\transfer\TestMain.java
*/
package io.nuls.transfer;

import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.RestFulUtils;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author: Charlie
 * @date: 2018/7/8
 */
public class TestMain {

    public static RestFulUtils restFul;

    public static void main(String[] args) {
        get();
    }

    static void get() {
        RestFulUtils.getInstance().setServerUri("http://127.0.0.1:8001/api");
        restFul = RestFulUtils.getInstance();
        //RpcClientResult result =
restFul.get("/contract/result/0020ce2b820d15ecbe1c8c22526611336fd425522340c6246f0364dd1
e784da8ed0b", null);
        //RpcClientResult result =
restFul.get("/contract/balance/token/NseCWjDsXZUz1VDNskCPA1qw4ETE9xjd/Nse7ke7HXfe3uv
fY1tyDHvWRao9uUKAn", null);
        RpcClientResult result = restFul.get("/contract/NseCWjDsXZUz1VDNskCPA1qw4ETE9xjd",
null);
        if (result.isFailed()) {
            System.out.println("query fail");
        }
        Map<String, Object> map = ((Map) result.getData());
        System.out.println(map);
    }

    static void post() {
```

[illegible]



080808001d5f2c4d00000000000000000000000002a000000696f2f6e756c732f766f74652f636f6e74726163742f6d6f64656c2f566f74654974656d2e636c6173738d935b4f135110c7ffa7b72debb640b92988566e96b6b8a278835a51bca155133124fac45236b0ba74b1bbe5c5f811fc02be2b2f3c68a235d1c4f822267e26a3ceec6eb12c25314de79c3367e6cceffce7eccdf9fbf02984249460a536d88e21c9bf3bcbcb20e1a28c4b989631c3e6120a3c2b48b82c43c6541c451eafb0998de36a1cd7783a27e1ba40c85811e8283dd53635d5d42aab6ac9aaacce084865abe2e8154720d5b4b9e0540d773b56302a8653140867c617052273d68a2ed05e322afafddafab25e7da42d9b3a275b65cd5cd4aa06af7d67c459336c81b19261a9959a69ab9b96a3ab5cb0aa951d759dce32d545f2cd3bfa3a158baeeace3c61a632e3fb41a3b6b7d99509ee31984ca9738d9b74efc9dfbd8b6c37c5f464f687f04131fd794d33ed40c083e5a77ad999197f2220acbd52f95b02f14dff26028905472b3fbba76db84250db68774db3d73cf548ca792a54d5ed9a4924821671c7f2188872c1aa55cbfa4d83154c34d439c50515f4e1b0822e742be841af821bb82930f23fea4ab8a5600c2725dc56308153027dc1eb5fab19e68a5e25a55f182bd36905f3b8c3e6ae80924ffbcf643a3d446f69e8a582490cd3830aea4057e14e989a6d73ab9afbe03a49a7e4def6ed39a421424cdbc8d02bd4ed89566ddae7f2d16702f18dae1d188f13f46da5e8830bd18f647567a42c8d7108569bec115aa9340a1aa3d94f10efddb07eb231d719c60059c50bc0510cd228700cc7f7257f40e85d20b9ad65729ac8bce4a2cb46d1d9dc4784fe95965d6f827292ee09bd5e947f02cff83e5c7808c32d40c24190ce962023186d05120e82f4504eef01202c2817a6e7e79ff59d62c2342e65736f118d6ce77690ccd611c9d1ff0da2e1eddc3744efb15c3be8e6214fff3a62afd1be0539effa298aafb1834e1ec817ae43da423bcf7e408a6c2112de76bbd3ef5245c9f64322c6c3443648dd2992cc4c9cf65876899790c138910e620e59b221e45c7ddaa4a4f885a312f231d12dcff295e84bf2aff4ca9767c2a34e645de23853143a8e0fac79acec26ce36762f1506fef1255d86516ada18553bd9a4e484cfd545712ae4e5339a6398648e7acf843da86888750680e7f1e128c36b9dbf687740e7735f305f2e34f389452ea0dbe64aabddde804aa43ac8d4d1197c19f9a69791d97d1967dca8b37f01504b0708d5267fac55030000b2060000504b03041400080808001d5f2c4d00000000000000000000000002c000000696f2f6e756c732f766f74652f636f6e74726163742f6d6f64656c2f566f7465456e746974792e636c6173739557eb73135514ff6d5ebb09a12d6d29ade5912242480b014585b660a18044cab358015f2cc9926c9b2625bbe129d6f71bdf84047c70fca17466546c2a8338e9f70c62ffe2b7ef38ba39eb337bb4db69b5267da7bcf9efb3bf7fef677ce3ddbfe fecf4bf00d884cf235883c93016a0c0439187291e4ef350e2c1e0c1e4a1cce03332ce46700ee723b8c0c3451e9ee1e112ce2fc0b36c9dc334af4ef3ea34af4ef3ea34af4ecb782e824e4c2a789ee717787851c14b0a5e56f08a8257d9f19a82d715bca1e04d7e7a8b87cb32de96e0d333125a46c6d5336a32af16b2c99162213b202168ea665e93d05ab3346a96746b3190d18cb48450ba5838a56725c447f462b250ce1bc93345534b92db2ca969333959cc68f9e418f9862d2485860c5335cb860429458714cf16b49284e54ebc136a642692db339992661814152d69e962b6a05f500b6922d521484daa662eb943cfa60aa696d54acc5a37b549c379a1b2a9e79323ba61d2527854cf16e8e812c56f762d0fde917f8af61dd8c6fc07f5826e6e93e08faf1d2325860922a179442f68fbc9327b5d211f5a4d0ad9856f3636a49e7e7aa3360e674633e72ed2a90fce7f985b29a99a20cb5c6d77ae4c8108b6d71f71a535328f48848627b5db49346c570108be3b301bc894c9becb4922d1bb61526df7035f309da79deb90f1b3381bdf3ae19fb5d5222b58e12b5c9ed9fe5fc1f296519aa9bdb4aceeccda777d4e7f2fc949dcf01377cdea7f2b6ace368f5365035a58440b627104fd96f7e40dc92588dd60dee09bf4a15dd139f1bcc9b37d3e687eb6e56a72de4ecbbd56cb8c15d716f2c6f1dd24e97d5bce1aaac0327c7b534897a9cee7fb1beb95497ac4ba29a12169210e9897dea94a535f5487abb9c6ae4c4850bd13b94f30453cca228570991d162b994d676eb9c9ae6996bb49e0f89620bf9aa388636d1409f446d1c7c3

3aac8fe27e3c1045121ba2d8887ba3b80f9ba27807ef4a583dbf6b2ae3bd28549c94f1be8c0f647c2  
8e38a8c8fa2c8818458e2be573bca7a3ec3190a5ed433fdb1283ec6273c7c4a6fd017b3da6e7f6c25  
3dacec8b719fb51e16d083e8b6fd310b27daa808bf6a79ac5e6a811711b8b6617248b82f66f54611f  
15914e3d8457dd2adbf28b8e1bc6a58b7a1b66d584eca4f537dafa9dbc44ec51d95b3dbc0d2b96a9  
41a97577dd9149c5b47e5a04e4d69056a85ebbcfad82c573505032ebc5d9d73e097d2ad9c7379b8  
f1327ae893bf86fe4a50e0e34224cbc7b568cd7dd5992ad29aa91ead994ad29aa92aad998a15121e  
b4eccd68279bea9ac601f2246996680e266e41ba614106690c594e055b698c0a00b6e1219a250c6  
1fbace01fe0fbde15dcee19bc03c3d5e06d84f6313ad17b13be99a32396b79362baac1d3a04aaba0  
35bac031fbc13bb3c88f8dd44967a12d98d87bd88f8dd446214d3d3800827820fdee34924e026b2c  
a9348ca9b48c04d244e316b1b10e9ab1279047b3d8804dd44fa3c898c609f1791a09bc8068ad9d8  
800897221fbc1f0768cd4d24e426b2c98388a8d383e43b84c3d54d0ed6120a31217f0da1cd14bba5  
012151fba3ce29ecd9ecdc8623643d8a310fcde4eb2eaa839e9a3d86a3b334ebbe09d9add910c56c  
6f4091af2d1f7c0cc73d88286ecd767a12791c4f78254f7113d94331a90644b86ff0c14fe2290f22613  
791114f224fe3841791b09bc8418a39d48008372e3e983e90622fe92bc2f8696e4ff47e8360e07aef6  
d34252a88f4d2efd708faaff7fe8a05fb386f7df4fb257bb827dd463b4fe4f35510bd8ae66bf4d5b3fc02  
e017003ff9fc152c74007e1b10108000f902b580800d080a40907cc10a9a1c40d0062802a0904fa9a  
0d9012836202c0061f2852b687100611b10ba8d369ec817fa118b7cb88666b67f831cb88680ffba73  
1dd6d3c70254d03295742755e632aaa80495c65e2a8e3ca5f53225f65bcad01f94a33f49ddad567fb  
374b5d5272b8d0ca9be0c7f41239f0fa7c8dd832eb949fa1b8b6464978424d74f476488d395b36f92  
6456537f43246161c24a402bb31d6c59d19d13d2b39b646f63f709db1f10fe80db1f147e12b9bdce2f  
3b962210a4f2e23a4458f849dc8e3a3fe9da9cb0345d2241ac74cf68b91a611a4fa10d596aae39fa6  
ce9a4e438693241ff46e7710593f80e859a0abe51d5b00d5f1096ef342ba7a369d190f42f16c347da9  
14c64a36afb24696ccec71373ae1ead7fa403c79d0fd155a2c41773ea67741ebb85aed6bb2ae8166  
958daba4cd8a47d77eb726107d85e21ec206362c2962be8695d296c85fd770b3bccf62a6187c86eb  
954c13d15ac763709a3a6494c394d62c242e5ff03504b07083101cfbe8e06000002110000504b030  
41400080808001d5f2c4d00000000000000000000000002c000000696f2f6e756c732f766f74652f63  
6f6e74726163742f6d6f64656c2f566f7465436f6e6669672e636c6173738d56dd731355143f9bfd48  
1ab669132895a6d00410d234103f5094222154a4bca87c10215846db2b44b37d99a6c181d8607c  
717ff007c601c47c719f5056774c61647671c9f9cd177ff1b47fc9dbb9b4db2a48cd3d97bcf39f79e7b  
7ee777cebde95ffffef21b111da1d5388dd2ec000dd0591ee67898e7e11c0fa5382dd079de72214a1  
7a374294e719a8dd13b3c9779b81ca37763b4c8e295185de5f95a8c9662f41e8bd7a37443a281a6  
6b34dccb56cd94489a97286ad6ab9e366c35175ab66badb66d9b4cd8a8b0d4b12256ac6879e3e  
e3b4ea6c9c9368f0aee39a33467dc1a95ab73f92483b6ed52df784446a6e7e7e72512265c6a9e2cc  
a1925537cfb76acb66e3b2b16cc3922a3915c35e341a16ebbe517157ada644b992e514eb2dbb59e  
4e38b15a7ee368c8a5bace12cbbb8c8219dfa6d6b651ab01167698e23c584b4c4a25a59352b6b12  
c9b949001f2cbb46656dc158f783e82ba65bee248f4d485f6ff618959c401fc7d6d36d5ee2cd2e2509  
25cc929213d113b5903d89531642e421ea9c7f4a6845c9897486ace6622fb7c3d81c3269e6072dc3  
066323b9d21de3ae51b48dfa4af1c2f21d1c37cdb94b0e33fdd49260da70d13ee06dd568ae7a55d2  
1a6613d861739db2dbb0ea2b12edc84d76f97b56f8c7cb4eab5131cf58cce850a7248779af4e197a  
5fa7e768974e633ca4695ca7ddb447a7091a8fd24d9d6ed1219d0c5a96e8c0ffabb544a36118a75a  
965d351b28debda0998f6574aa50950700db56c8f86d7d2c83540a99706b7bdb6f83a442a6b7fbfd  
15844d16323d4dce2749f7759aa1b7519330b5a2b228df583bab20a16675ad78ca76b82f075c2002

[illegible]

d804950c2e90e71cd9ad52b117ca3687ca36a1a342ae73d8409558024b44279155f3e24f5c52713  
32d4ab935a9047f2c5151df0413408542dee37ac0392359d426aeccee846c0b7ac0342b200f3eafcd  
be60980f58dce0282d305f3960560a4346cca698e855bcc68492cf652427d7c4dfb2f3644432b411d  
90921c9f1271891fb98716457cabf22f7e9338c15b5a301c37cb110da8ab427681f6132ded9a5139d  
2d0fc7e276e0f5c63f504b0708daefa656a40300003f070000504b03041400080808001d5f2c4d000  
000000000000000000002f000000696f2f6e756c732f766f74652f636f6e74726163742f6576656e7  
42f566f7465496e69744576656e742e636c6173738d54df6fd35614fe6e7e39316ed286b6749442b  
741499c503306039aae7464b095854d5a512578aa9b9ad660ec2e71fa82f627f00ff0bef5a50f208d2  
06dd2b4178ab4bf69daf6dd6b27a5294855e473cffd7cee3ddff9ce71fefef7f73f015c4643471197724  
8e373692ecbed150d5fe8b88a6b3aae4b731573d29bd350d3a1e35216f372fd529a852c6e64b128d  
daf34dcd45017c86c07a1b3b42e30dc78646fdb9667fb1b5623f0376a02ba7c570ffc87ee8640a9e1  
0696dff1da9644ad66e0872dbb195a4f8275c7b356fa913c9799777d375c10a895062f3dea25e5158  
1549da840a1e1facef79d276b4eeb9ebde611293682a6edadd82d57ee6330156eba6d81ca073238  
db8e1faa0c4be4764beec834b7e1842bb100c552f9b0044371404f05934147d7c1f9a9637b2435f6a  
e0e3fac3d729a61adfc404004b29843af543176c8eccba1dd7c7cd7de5235b2d302d94dbbbd19099  
32c959798a4e5b43b1e830537d930580e5bae4fa6a307ca8950d9d3e5a0d36a3ab75d25e4013d66  
65b881097c6d60146306c671c2c02ddc16281f59540ddf18f81467357c6bc04445e0c4208b9b1dd7  
5b775a54ff69347c73d3069670479aef04f2d5e9fdb99b9b66613f1b9885c5111d544ae0648f589f53  
7bfdb1a5982881384659b6b0eed96df6e1f8014d144849f207bb2e3073b40e537b7b6bcbf1393b17  
4a87b53e2c7f5c786d20be37101f8cc7c7fcd8b207f24f8636ff8779090ede19a213e818f684f72779  
f78926bc17c0d61565e2161565f21f9521d98a42d22457b8cd6400e43c8f3778ac874740c53380d2  
84fa611ca938912f4cf302a4a637195efd2e66f48bce85f9e51e088bad08802e20b056bf8e43d87938  
387c7de7b9803151f7e1317b86a567e453ab55bd943deec2255e1f30bd2c9ddca5f48df95b4f6302a  
972a9f2e32cf51d8815e5538a364ea3d8cc88558b20b6d0705e9bd8596da412ab9cb1c49c56b9c3c  
c0dc1a77136434451d16a8c4be6aab7dd556710e33643a853ace134ba0a404c96979f10f4e69286  
7c4a8be28ebe7b71197f44c7515b810b11e3215e3ac64313f7c667233e22a61f2cc4978757e729f5f  
5e7138c7e198e195e715aff1e8c298d771a6a97227149bd3488d2c8affa874827c48853e623f2124b  
559d52249ed47d577a0f407f4fbaf71ac68747bfcf2c542b7472a5f1ca6dfc5c86037abef74b3d4efe6  
4515f5d9ff504b07089ec970336a030000d8060000504b03041400080808001d5f2c4d0000000000  
0000000000000002e000000696f2f6e756c732f766f74652f636f6e74726163742f6576656e742f4164  
644974656d4576656e742e636c617373a556df531b5514fe6e76970d2124107e432914d4860d74e  
b8ffe3020a520556a68556a6cab952ec90e2c0d09261b661cc719df7cf7d1e72a2f3ce88cd051671c  
9f70c6ffc3ffc251cfb9771393409d3a0e70efb9e79c7beff79defdc1d7efbf3c79f01bc828d0806b1d80  
e134b3cbccce3320f3738f08689372358c1cd08dec2cd0e64d85ac12a2f5779b96ae25604312c867  
19be7b77978278c77c3580be34e18efb1236be27d137705daf64abebbb9217e8ca6c3b7b8e5d708a  
9b76a654dc9ca598e7bb3b1ceb6063a954f4dda22f9068c85cf3cb9ecc3538a5523fa6ea7b053be35  
57c0ab5af799b45c7af965d81ab2de1b98c57b28bd542c5661c768eee283b39dfde29e5dd829d656  
c74eeec3ca399f38a9e3f2fb09c6c85fa2feb00602baaa9ac80be449708c4335ed1bd55ddd970cb77  
9c8d82cb044b39a79075ca1eaf03a7ee6f79c4cf7a0a60778f6a635fcfe719ef322f08727ff3419fec6  
0edb9ff149eb9724cb57dd3f5b381d089e4d449a9396125503b16d875c17b9b76d4250f077995863  
31b754f9f70fe17b5dd8fab4e818eee6b2cd5ed8d6d3747eadd1710a5e64e0c425228874077aef94e  
eed1aab32b4b4eaf86006f39952d25ba969c5aa14bca6ea55aa064418bb05f52d404226ba56a39e7

[illegible]

[illegible]

6a927d0e442cd5a4565aaa59adb2d46ab5c65257abb5acabb1f7da07ec54ce21231d93b53296ac5  
f63a96bd575449fc3ee113b9349338a150ebb79e1a7ca6e0b27f1246bdee2bdd22a59eb62ce1cb16  
4d6d968a9ebd53a824b94e97d1dc954bba5d6abcb092e3de9c0b420bd416d642ef582206378b62d  
75836ab5d426d9cc8bffb648d6eeb613399ed997ca79a28339b6538e926cf276db529bd516859af1  
a2d142faf762b1d5c2713cc25814743b745da9b526db475ab74db1a8aad29af7644d96daecd1a0db  
a1305baa3192edebed4d67723430d549bc3bee70b3446da7a576a9dd96daa34805611a649354c5  
a9c8ad193251c4b581b11dc66f96ba5172511aefe6a9ed07b5c5ede745d0a0df9ecfefe1cdebc43feb  
254c6156a22f93a14464e852fa00bd3a6813d9157a9ab7959e4f74a7b336b3bc57d55aea2629f3aa  
cc506610b9369bd9916bc2eea1283eaf5619ea660b9f965a5eb6d50192b631e2de1b72683789d10  
9a3c46198ce788f80c05271417f1b18c2846a3714f3dea1a8ff713c61a94e25cd8a4bc01aa20eb194  
b9f7a6f68bb866c20b1e3a7c397702ad21350f52d0c6b65b9849b602e314ae74ad998eb8dc8a53c6  
6a67872975784de1a2e1775261c2bbb426c05d6bdc56a5382af780a11972638774d585fb72f4f746  
3091eee98d67ecade90b74df6c30678c79ad6da37559dda2ecef4bca87c9d4e8ee319a605a337dcc  
fdebb334777af4bcd605e2bdbdb620353646035e336acabd1d968f9077f276417933972e24624a74  
b4a03475c2373ab68c468517d1a1edb13f6be7a473ae888e5c9308985cddea74e99563792329a3c  
8d5ba572fe3dbf02fc6106798e01ccb955610ee23d4149cdced9cd4e2f6fad191fdb69c13a440a101f  
145756311c832ce02cd48b466bc1646f4bb3dc79ce838cd115517f7f6d1e4a563187be11cb9530ad  
78e8ae744bec6c672dc67f7c87740f579acd674a905fd892e3bb14fa79b012dee143a9c3f0ea63c7b  
839d83e1353bbd5c94476b46da44264866d7b3d34df676db5bf445c11b945bd6c76f73866b84009d  
8c15cabb6ec29f22e2c9f2d1c17b17bbc595c2b9b5d1f15b79ef1b68166175817aab1cbeec85cedfe  
9945079748cfa0a9065e2c2376549dd01af89a7d6a7db931d24fbb9e781c8f0f497758eac2adfeeb5  
9bd99bccba10889d24ba8037b99cca76c8abe5a439c3aaee7198a1f57c3cb9ba3b2d680ae628c9d  
6a8a797f5ccdd6b53edb21773d08604bfb0da518c2269a101f8a425e75c92ef45b885e37d43c6dd2  
8e17b0f527ca63973b7de095c517b0aaa760045bb4ea1f849f8065ffd75c17e04f2303e05c3771cbe  
e272b31fc1daba27117a9c1b8bd1cbe70c04f89ccdc366a00c33f9ac423daab10411ece7ca54e710  
649005f49b18ab90e3fb6414bd831a1419e833700068966fc65b719b6361d152aa0e012a143b0b2  
b76c6b3a28456d49d2d8c2c8e1678a3528efa7151a32fecab3d89108d3fe62e0da08c6e4d2a9f9c4  
7b9b35491c71471e8247ca751c9ccde7c473fa636fa07308d92d31b03e140d89f47381ca8cb63463  
8b0208f999cf2e5316b00b32952d5681437989566eccc835834806a4e451a83e160a57947284e55  
79cc09076395e6a23c2e0e1be1e069cc2dc28ec3a63afece735464e4312f1c28ce637e38d08f4bf38  
88a29617f38701a35c5d83180da5d617fdd82b0710a75fd58100e3ca6b32c616fc3743ee7a102971  
00bf319f228435e83bda863ec628c613d0e6121eec0e5b80b8b7084abc7d0804770251ec5520c60  
19be8a467c8b732f62055e41135ec54afc04abf05bac5106d6eaf47530038f329507713b82f814a5ff  
08ef85295929a4946f7f8c3f61cee4ed10df8af4db9fe27d84c8127c17efa7161fad3a863fc361f869db  
11daf50166f71867ff1c1f8481bfa0228b3a9c993b3923006940a8b45959faef3984099266f50e7622a  
8117397fefb21ce1af8b0818f18f8288d78931053f84b7ccc41917a2fcf16880e5407eff6201463de63  
1e682e1314549f427d2c8f85e597bb98a866064f6351119ec234c19327be9802828225470b3357b8  
330d8dfef0ff2c2687fda771a58277dc52392e8f65f760267f1adde5870acbcfbf576ae5c15f651cbfc01  
acd825a7c74ea18979f79df0d2dec41a035a3109d7b3ced6632e36620136717e0b47db70137630d  
83b9986dd0cf01e5ab0170f73f6b388e334094352ba84296038bc8a1c605a8e48faf8f6579a169a98  
5049908fbab631911fe48e42829c993b392309aa4669c8cd4e8c19685693e4b58a996856b39b958f  
1b3e8ebf76ebf9e75458ca8c246babf35859deece4606865af3a4fe8f3582d91d5115c103b236f8d0f

8da8eb35ac6bbd90c7d54e61eb0d4751113be3bbbf70c25a2f5381d135dcc65aad170a081637848  
a1b4a2a4ba41499a907511b0e54969cc635625275a315b6f2b8361ccce3ba7b10f435848e2378b8  
8465fd4265e8284a8acb5b784cf18911357f2f5df30f71adb5d10c9bcf089968c26d340beb61d32100  
ed781ed71f25fb907a2fd5fb9e29b8b24e5c91e9b051e08bf51a36c2171b86e3e6032409f03208f132  
98cd6ba19ed74103af822626a693a848130d87d047b903f8280bf41e16fc832cf94758ec278986a75  
9d6cf53e2dbccfd4b9c7b8585fc4362e475eef831f1f14bee790377b1ec3fa4caf061351d1f51517c4c  
2dc611d58a8fab1df884eaa456c1df6182e06912d727a8dfc267b0187f438d253cef0692cb61d2cc23  
c4f23d384a6b8f106fc75cc2497a84932c20966f9fc4bd446cbd6a7411dbc03305c57e8e5ed3d80dd  
0a3975909a419da7713ee631c4c8de708773a52777a52f77b520f504a30be0ce5d39ad55b58ac29  
e6c166159e4c64bf8d72a25c28e76d54927f945af026fcd61b283b877a030fcd74d0ff309d71786805  
8d93c27b78240f8d04fc864162dae8f0c612e33e54eaa2f118e786c28a87ec4dceccce627b065e895b  
d555f88dbf2d82e85240243eec4f21dcea69dcefc9e8de3c4b03c7b9de173145605826313d73e84  
d3d3ecd1be4385a48297bf0774cc9e798c647c9b74f3051272995d749aea5b7f4d5239987dd94cde  
1af24aa98bb6aa94b6e8641223987d99a45a6337825cd5a52666669d6e7d6e35e28a36e280f8d1fc  
a67e02f3e1113fea0f77a2045b5bb1f7beebe17a19826714762094922581bf39fc28d32f2df57107e8  
f087f9234e2248232acaf1b8b4f0c89522bca357d4ec297e8e19779273f854bf115dec35f6304cf927  
2bf8e6bf12ca3f60d92f53749d2cf13622f9094bfc38eec4516de4b84fef758682f0f89de212f7a87bce  
8ed75a317c3660d73277ad310d2d19bc9e89d83459496abd26058a2f619fcaddbf09d71a3b6e2dd0  
3b0a1d1179350cc8b0d76204ee3f703fafc2aed7a8dc8f8116fdded7b19c8430e8c30acf87159e0fb35  
c1fae60814981fabccbc599b9d3f5aa14be2198e0f26709b822edcbf75d5f36bd4b5f4631b0464589c  
bc08f857d83fe55b107017e4aff7e469b7f41ff7ec5dee8d7b806bfc106fc6e888f9b3c1f37793e2e747  
d5cc5fd4238833e3a330fb83e56c03fc4c773984bea701bf61314fe1c4bcbcf1c456e1bb3fffc6dcccdef  
11a973123a03b1657e0267d6bc6f45a61f2c48888bc45fbd4615d977a12ac24ab2dc2615c0cdca1c  
d25cecf722b2df6b2ef6e3f3788c1159c8083854bd89042cb1196c2e9c9907dce662a646b56e2eaad  
df0f0f7a16655413414c97fe5b9c8be8ea362fe06ea66b17b7a5c474f0c0ee956d4825f956ae3228e9  
4675c401ba2f4dbdff39629e29f9378d2d57a3f4d94f4ae7cf7f5b253b7f33797f3232038f8a9345562a  
32661922ac71c358597d65434a969434c5be999b6d24352958ba4312b22af7d3df57b504b0708c5f  
0e424930f0000e0210000504b03041400080808001d5f2c4d00000000000000000000000002b0000  
00696f2f6e756c732f766f74652f636f6e74726163742f66756e632f566f74655374617475732e636c6  
173735d90cd4a033114854f3a6d476bffd4950fe0d66c7c81a158094847483a2e4b0c515a86043a4  
91fce451fc087126fc6420777f7fb72c23dc9f7cfd711c0236e735ce7b861984b55a8b5dcbcb15426dc  
44a28062618260bef9aa05da8741d6d4677f02f5b95ea2979c6303df9a4c4ea39d91ec3f8645f8bb5  
6c93d9d92d5eca3fd76718491ff7c62eb7b56598553e58197488cdc34e1f34c3fdd67317eb861fe88  
41befc25e9bc03fa233fc1ca67229ce6bed3e79f9beb3260ce925e8d18e01ee90a14f3448cd3024ce  
3b7c417cd9e111f15587c7c4930e4fdbcf98b5f3fc17504b070820205379de00000052010000504b0  
3041400080808001d5f2c4d00000000000000000000000002e000000696f2f6e756c732f766f74652f  
636f6e74726163742f66756e632f566f7465496e746572666163652e636c6173738d50cd4e023118f  
c8a2bcbce20f08fe9d3dadd1d88b378889311a256b4cc4709053ed16525c5aec764978350f3e800f6  
5fcb610c440a2bb8769a733fda6f3f9f5fe0100e770e8c3be0f07048adc0866050116460336663461a  
a4fdbd648d56f2c33dd65ea38929aaa2c49e9585b41b956d6306ee950c722a11de4ae959576d220  
e049252d81d3b0f5a7e74aab9ec4cb9fd1956b3069d8eab6f2bdcf99ea38ca0b1d51362216c39195  
5a1108de326126d3f3133cff7fbacadcf928d22cc1a0b5dcef1e9c5999d07b364259d0967dc56c6670



c0d392a0399f371f95c6aff4328e8d48d3c6823892a96d2ed419692cf3027f02759764e6b965e9f435  
470bb5adbcdc75d1d699e1e24626e8a8e5c63b6585e9312ecef25904c2d585f432c5e92f3d81ea4f  
ba879781e0b648804001f2afe41158030f00711d8a0e7d28390ca0ec7063869bb0e5701b2ae8afa2  
bb003bb8af411dd704761db3f70d504b0708dcf90d46580100009e020000504b01020a000a00000  
800001d5f2c4d000000000000000000000000030004000000000000000000000000000696f2ff  
eca0000504b01020a000a00000800001d5f2c4d000000000000000000000000080000000000000  
000000000000025000000696f2f6e756c732f504b01020a000a00000800001d5f2c4d000000000000  
0000000000000d000000000000000000000000000004b000000696f2f6e756c732f766f74652f504b01  
020a000a00000800001d5f2c4d0000000000000000000000000160000000000000000000000000  
76000000696f2f6e756c732f766f74652f636f6e74726163742f504b010214001400080808001d5f2c  
4d372f2223be0400002f0b000028000000000000000000000000000000aa000000696f2f6e756c732f76  
6f74652f636f6e74726163742f566f7465436f6e74726163742e636c617373504b01020a000a00000  
800001d5f2c4d00be050000696f2f6  
e756c732f766f74652f636f6e74726163742f6d6f64656c2f504b010214001400080808001d5f2c4dd  
5267fac55030000b20600002a000000000000000000000000000000f8050000696f2f6e756c732f766f74  
652f636f6e74726163742f6d6f64656c2f566f74654974656d2e636c617373504b010214001400080  
808001d5f2c4d3101cfbe8e060000021100002c000000000000000000000000000000a5090000696f2f6  
e756c732f766f74652f636f6e74726163742f6d6f64656c2f566f7465456e746974792e636c6173735  
04b010214001400080808001d5f2c4d2286dd2e79050000f50b00002c000000000000000000000000  
00008d100000696f2f6e756c732f766f74652f636f6e74726163742f6d6f64656c2f566f7465436f6e66  
69672e636c617373504b01020a000a00000800001d5f2c4d000000000000000000000000000001c00000  
00000000000000000000000060160000696f2f6e756c732f766f74652f636f6e74726163742f6576656e  
742f504b010214001400080808001d5f2c4ddaefa656a40300003f0700002b00000000000000000000  
000000009a160000696f2f6e756c732f766f74652f636f6e74726163742f6576656e742f566f746545  
76656e742e636c617373504b010214001400080808001d5f2c4d9ec970336a030000d80600002f0  
0000000000000000000000000971a0000696f2f6e756c732f766f74652f636f6e74726163742f6576  
656e742f566f7465496e69744576656e742e636c617373504b010214001400080808001d5f2c4d33  
72bf1b7f040000d60900002e0000000000000000000000000000005e1e0000696f2f6e756c732f766f746  
52f636f6e74726163742f6576656e742f4164644974656d4576656e742e636c617373504b0102140  
01400080808001d5f2c4d3c1bc221fc040000c50b000031000000000000000000000000000000392300  
00696f2f6e756c732f766f74652f636f6e74726163742f6576656e742f566f74654372656174654576  
656e742e636c617373504b01020a000a00000800001d5f2c4d000000000000000000000000000001b000  
00000000000000000000000094280000696f2f6e756c732f766f74652f636f6e74726163742f66756e  
632f504b010214001400080808001d5f2c4dc5f0e424930f0000e021000029000000000000000000  
00000000cd280000696f2f6e756c732f766f74652f636f6e74726163742f66756e632f42617365566f7  
4652e636c617373504b010214001400080808001d5f2c4d20205379de000000520100002b000000  
000000000000000000000000b7380000696f2f6e756c732f766f74652f636f6e74726163742f66756e632f  
566f74655374617475732e636c617373504b010214001400080808001d5f2c4ddcf90d4658010000  
9e0200002e000000000000000000000000000000ee390000696f2f6e756c732f766f74652f636f6e74726  
163742f66756e632f566f7465496e746572666163652e636c617373504b05060000000012001200  
a6050000a23b00000000");

//parameters.put("contractCode",

"504b0304140008080800038b2d4d000000000000000000000000090004004d4554412d494e462f  
feca00000300504b070800000000020000000000000504b0304140008080800038b2d4d000000  
000000000000000000140000004d4554412d494e462f4d414e49464553542e4d46f34dccb4c4b2  
d2ed10d4b2d2acecccfb35230d433e0e5722e4a4d2c494dd175aa040958e819c41b9a982868f817  
2526e7a42a38e71715e417259600d56bf272f1720100504b07089e7c76534400000045000000504  
b03040a0000080000fb8a2d4d000000000000000000000001b00000074657374636f6e74726163  
742f6d756c74797472616e736665722f504b0304140008080800fb8a2d4d00000000000000000000  
000003200000074657374636f6e74726163742f6d756c74797472616e736665722f546573744d75  
6c74795472616e736665722e636c6173739d56eb7313d715ffad257997454e8cea00f210aa262d91  
658cb00c989a94c498180cb6f143d8d86d4256d25a5a4bda95b52b83d31749daa4e933e9234ddf8f  
90a68fb40d2d96dd64265f3ac34cfb0ff463ff857eea97cc747acedd95ac48824e6a8ff69e7b9ebf73ef  
3967f7efff79e73d00c7f08e8a53585590df85fb505050e4d5e48725a3a442c62a3fd6149479b51538  
2cacc85857710dd75584b0a1e219d8323eabe073bcff7c00fcf7daab3758f005667d51c18d1af75905  
cf0a5a1a1c8082e75cfa6f7fbda1e079155fc29715bcc0be5f64435ba5685f51f09282af32f36b8ce2eb  
0abec19b6ff2e65b0a5ee6cd2b0abeade03b2abe8beff1e35505dfe7f535053f50f043267fa4e0c7327  
e22e3a712fca656d425842657b5752d5ed0cc6c7cde291b66f694844e7ba398b20a12bac62cd3763  
4d359d00a15d256327ada286a055b8234e123d82109b256284c982b1649af96b40d2d5520455fb4  
6f81628c5919dadcd3f6998fa74a598d2cb49571c9ab4d25a61412b1bbcf7987e276790e323938e6e  
3b69cb74ca5ada89172b05678348d35ed1cbf12489a69893f43884363c57311da3a82f18b6418e46  
4dd37234c720e412fa260d2b6e560a76bceed0cee4e35a5d273ee342e6ac1f354cc3394d1965756  
75a9c4e4fb4afddf9ec228579ef8876137db67e2a94f7043948696490d6f924c98293dba5653265dd  
b6af0e4a38d81ed4a8ab41fe0322e906a344033d44072ae4e356d9339170337a6f9f6e1645cdc9c5  
cf18d909d3d1b37476ff974dcb79b4d76b7b707e4a62904b8696aba626616f7b635733e16a268466  
17514394b2ebcb950cb99279474be7a7b49257479d5aa9a49b19090f445b3170592aae825e96a0c  
e5b95725a1f37d8706f4b751d61fb20a6312e614f8b34884f81caa523990ce2319c0e62146724ec6b  
8e79a6621432acade367748f9181c8e8e2e0f1939783f83966242088b3ece6f087297b19bf08e2977  
83d88246eca7843c2817bdda584c8b49d71b2b3e746af5f99c91d1fb296af8f2eae2e94ccf3e6c5e16  
2e5d8b525ce7225885fe1cd207e8ddf1050af884722c25a3fbe9c3766edc1d9c152e60967eee2b52  
bf913d3f91396b554494c26bc001b99e4c2a29d9f4a0f9fcb2f3f93b2d712e3d6586eede2f8b5a9150  
9fb3d3491b4e8bf484a8fe8c592b37144c66f83f81ddea2a0aa2d6e662412c4ef393d1d64a8aa5ef13  
316455de761c41a7fc04dd6f8236b6435bba0af38cc7e1bb798fd276a4e6697ca465aa8ff19b7a843  
555595d0ada6e848cf14ac74febc66e74622326e07b1c986556cb161da32cc94660bc36dbcced072  
ba91cd398c21a8f2c0a1c9582cb1fc2fec5862b73b777f29b5aaa71d090fb6bd97316f2361a05d91b6  
b0bc1aa2c6501cabd602fbd bba9eb2bdeef02e3e7a9719b8336ef6d7a65c6b1bf6b4e3937b710597  
e86ac2d10b7735566af5cb6a779912d48ebd6de15d760c1ea872595fab1865eecee872fb76eeb4bd  
660eachbb6fa90f1ca97b0ff73c52d9ab1d31be2f502bd5736a7f03b59aba1b74515684cb14af3cee24  
66e4a8ce7826b5f1cd4e6bf546eaf5dac2c7e8e57f0a121e851f1d3c6df895cbb386788f13dd41fc108  
f9dfa7e8c7e6785de6ee23d8171da9fa35d3f7f63f0ef96503b4fcf4ec18863023c838410177091d649  
4f3a257834fde8c94e36e0137ac3b12d48b150c7267cb1907f1381587768139def425eda82125bbe  
0d5f15bb422a3daad82db4829be8da097d90be5b804142ff085444d18d3eec450247318413f455b6  
0369d88324e112663c18715a59168851a0b79bf2196e300ed48d67ebc6039eb12fe46f363dd560ea  
ab9bce61bed9b43bf45693e9636d4d93f5a8c73c53751bf757d15dc59ee6e0630d1ed4ba87cb750f

05f2db41ebd177115a0a7d640b3d93fd3d78601bfbaad8ef9d7c284c275e8fe09efe4e9cfbc05f6ce7  
c8fb79f450348eb7d7f5e9c55371080b581458af5064b774b6c9ce4feb950f44eead4526de01e24d1  
deec1830dbc8f126f7aa007910f81d02d8b4b84638610ced17e9e8a23498571995a6051208eb958e  
a88a73dc407318225a23a48fb109689f235d5f1a7f1192fa37f924ca175a6ff0eba07ee20180edc81e  
c7f137e5fe8a16d3cdc7f98100f84fd5e2ee100e5322287e570e7ffcec42732e9a5b8a078fbf0240ee0  
293c84ab543e4f531d68228b3937be9705534f8a2c987a8a743b04f5b4c882298d787e41a5a8b903  
824a13af93bc3e2cb295911115dfb1e771a929f1b3b5229266c90907793f56cbe2e3228b4f5471c84  
d610b8fd465d15a866d647d4216aba2bf4970580806aa38d224880bc1d11d4168b0519c10e2a16d  
1c6b0d765cc84eb4061b768d5a839d14824f36041be1e76d74ed545b82ba0d58a1fbc9e22472342  
30cacd2fff3c8e325eab73750c47b30f10f58f8174af837d61a7af47defee3af813c5ebd139af477b1b2  
6611721e9af0dc28611a80acd0ac2586fe8c4de7a45f412aad3a21373c2c6f82f504b07087c257e1c  
05070000b30e0000504b01021400140008080800038b2d4d00000000020000000000000090004  
000000000000000000000000000000004d4554412d494e462ffeca0000504b0102140014000808080  
0038b2d4d9e7c7653440000004500000014000000000000000000000000000003d0000004d4554412  
d494e462f4d414e49464553542e4d46504b01020a000a0000080000fb8a2d4d0000000000000000  
000000001b000000000000000000000000000000c300000074657374636f6e74726163742f6d756c747  
97472616e736665722f504b01021400140008080800fb8a2d4d7c257e1c05070000b30e00003200  
000000000000000000000000fc00000074657374636f6e74726163742f6d756c74797472616e7366  
65722f546573744d756c74795472616e736665722e636c617373504b05060000000004000400260  
10000610800000000");

```
    RpcClientResult result = restFul.post("/contract/constructor", parameters);  
    if (result.isFailed()) {  
        System.out.println("query fail");  
    }  
    Map<String, Object> map = ((Map) result.getData());  
    System.out.println(map);  
}
```

```
static void main0() {  
    RestFulUtils.getInstance().setServerUri("http://127.0.0.1:8001/api");  
    restFul = RestFulUtils.getInstance();  
    Map<String, Object> parameters = new HashMap<>();  
    parameters.put("password", null);  
    parameters.put("count", 100);  
    RpcClientResult result = restFul.post("/account", parameters);  
    if (result.isFailed()) {  
        System.out.println("create fail");  
    }  
    Map<String, Object> map = ((Map) result.getData());  
    List<String> list = (List<String>) map.get("list");  
    TransferTest transferTest = new TransferTest("Nse4R5LhHSpRQqqocDmBnavi3B2HysCM",
```

```
list);
    for (int i = 0; i < 1; i++){
        Thread thread = new Thread(transferTest,"Charlie-" + i);
        thread.start();
    }
}
}
```

```
177:F:\git\coin\nuls\nuls-1.1.3\nuls\client-
module\client\src\test\java\io\nuls\transfer\TransferTest.java
*/
```

```
package io.nuls.transfer;
```

```
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.RestFulUtils;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
/**
 * @author: Charlie
 * @date: 2018/7/8
 */
```

```
public class TransferTest implements Runnable {
    private String from;
    private List<String> to;
    private RestFulUtils restFul = TestMain.restFul;

    public TransferTest(String from,List<String> to){
        this.from = from;
        this.to = to;
    }
}
```

```
@Override
```

```
public void run() {
    int count = 0;
    while (true) {
        try {
            Thread.sleep(100);
```

```

    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    for (String toAdd : to) {
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("address", from);
        parameters.put("toAddress", toAdd);
        parameters.put("password", null);
        parameters.put("amount", 300000000);
        parameters.put("remark", Thread.currentThread().getName());
        RpcClientResult result = restFul.post("/accountledger/transfer", parameters);
        if (result.isFailed()) {
            System.out.println(Thread.currentThread().getName() + " - transfer fail : " +
(++count));
        } else {
            System.out.println(Thread.currentThread().getName() + " - transfer success : " +
(++count));
        }
    }
}
}

```

```

public static void main(String[] args) {
    String a =
"02230020a2075608eb8b0468413e0537d909196dc16ad1567f5b957237e68ce9deba21e4000020
4aa9d1010000ffffffff230020407969ea49a0f74c5874c725296a6043c30391293fc56f8327201841
f80ef65b000080ca3961240000ffffffff0217042301dbdb74f285112040dc80c56fe086c16e192d4e
9700204aa9d10100007f2dd4d3660117042301dbdb74f285112040dc80c56fe086c16e192d4e9700
80ca3961240000000000000000";
    String b =
"02230020a2075608eb8b0468413e0537d909196dc16ad1567f5b957237e68ce9deba21e4000020
4aa9d1010000ffffffff230020407969ea49a0f74c5874c725296a6043c30391293fc56f8327201841
f80ef65b000080ca3961240000ffffffff0217042301dbdb74f285112040dc80c56fe086c16e192d4e
9700204aa9d1010000fa33d6d3660117042301dbdb74f285112040dc80c56fe086c16e192d4e9700
80ca3961240000000000000000";
    System.out.println(a.equals(b));
}
}

```

178:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-  
module\ledger\src\main\java\io\nuls\ledger\constant\LedgerConstant.java  
package io.nuls.ledger.constant;

```
import io.nuls.kernel.constant.NulsConstant;
```

```
/**
```

```
 * @desription:
```

```
 * @author: PierreLuo
```

```
 */
```

```
public interface LedgerConstant extends NulsConstant {
```

```
    short MODULE_ID_LEDGER = 8;
```

```
}
```

```
179:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-  
module\ledger\src\main\java\io\nuls\ledger\constant\LedgerErrorCode.java
```

```
*/
```

```
package io.nuls.ledger.constant;
```

```
import io.nuls.kernel.constant.ErrorCode;
```

```
import io.nuls.kernel.constant.KernelErrorCode;
```

```
import io.nuls.kernel.constant.TransactionErrorCode;
```

```
public interface LedgerErrorCode extends TransactionErrorCode, KernelErrorCode {
```

```
//    ErrorCode LEDGER_DOUBLE_SPENT = ErrorCode.init("80000");
```

```
    ErrorCode LEDGER_P2PKH_SCRIPT_ERROR = ErrorCode.init("80001");
```

```
}
```

```
180:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-  
module\ledger\src\main\java\io\nuls\ledger\module\AbstractLedgerModule.java
```

```
package io.nuls.ledger.module;
```

```
import io.nuls.kernel.constant.NulsConstant;
```

```
import io.nuls.kernel.module.BaseModuleBootstrap;
```

```
import io.nuls.ledger.constant.LedgerConstant;
```

```
/**
```

```
 * @desription:
```

```
 * @author: PierreLuo
```

```

*/
public abstract class AbstractLedgerModule extends BaseModuleBootstrap {
    public AbstractLedgerModule() {
        super(LedgerConstant.MODULE_ID_LEDGER);
    }
}

```

181:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-  
module\ledger\src\main\java\io\nuls\ledger\service\LedgerService.java  
package io.nuls.ledger.service;

```

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.*;
import io.nuls.kernel.validate.ValidateResult;

```

```

import java.util.List;
import java.util.Map;
import java.util.Set;

```

```

/**
 * Created by ln on 2018/5/4.
 */
public interface LedgerService {

```

```

/**
 * Save transactions, automatically handle transactional coin data
 *
 * coindata
 * @param tx
 * @return boolean
 */
Result saveTx(Transaction tx) throws NulsException;

```

```

/**
 * Roll back transactions while rolling back coindata data
 *
 * coindata
 * @param tx
 * @return boolean
 */
Result rollbackTx(Transaction tx) throws NulsException;

```

```
/**
```

```
 * get a transaction
```

```
 *
```

```
 *
```

```
 * @param hash
```

```
 * @return Transaction
```

```
 */
```

```
Transaction getTx(NulsDigestData hash);
```

```
Transaction getTx(byte[] txHashBytes);
```

```
/**
```

```
 * Verify that a coindata is valid, the first verification owner is legal (whether it can be used), the second verification amount is correct (output can not be greater than the input)
```

```
 * Check whether every from one in the coinData exists in txList database, or if not, is to continue to check the from of the existence of the deal and if it exists, represents a double spend, does not exist, is the orphan transactions, finally throw an exception
```

```
 *
```

```
 * coindata
```

```
 * coinDatafromtxListfrom
```

```
 * @param transaction
```

```
 * @param temporaryToMap
```

```
 * @param temporaryFromSet
```

```
 * @return ValidateResult
```

```
 */
```

```
public ValidateResult verifyCoinData(Transaction transaction, Map<String, Coin>  
temporaryToMap, Set<String> temporaryFromSet);
```

```
/**
```

```
 * Verify that a coindata is valid, the first verification owner is legal (whether it can be used), the second verification amount is correct (output can not be greater than the input)
```

```
 * Check whether every from one in the coinData exists in txList database, or if not, is to continue to check the from of the existence of the deal and if it exists, represents a double spend, does not exist, is the orphan transactions, finally throw an exception
```

```
 *
```

```
 * coindata
```

```
 * coinDatafromtxListfrom
```

```
 * @param transaction
```

```
 * @param temporaryToMap
```

```
 * @param temporaryFromSet
```

```
 * @param bestHeight
```

```
 * @return ValidateResult
```



```

*/
public ValidateResult verifyCoinData(Transaction transaction, Map<String, Coin>
temporaryToMap, Set<String> temporaryFromSet, Long bestHeight);

/**
 * Verify that the from is repeated, and if repeated, it represents a double spend and throws an
exception.
 *
 * from
 * @param block
 * @return ValidateResult<List<Transaction>>
 */
ValidateResult<List<Transaction>> verifyDoubleSpend(Block block);

/**
 * Verify that the from is repeated, and if repeated, it represents a double spend and throws an
exception.
 *
 * from
 * @param txList
 * @return ValidateResult<List<Transaction>>
 */
ValidateResult<List<Transaction>> verifyDoubleSpend(List<Transaction> txList);

/**
 *
 *
 * Deprecated method
 *
 * Unlock the coindata of a transaction. When certain business scenarios require a certain
amount of funds to be locked, an action is unlocked at some point in the future, the method is
called, and the lock state changes to the spent state.
 * The specific operation is to determine whether the from data in the coindata is -1, if it is not
then return failure, if it is deleted, and then write the new to the input pool does not spend
 *
 * coindata
 * coindatafrom-1to
 * @param tx
 * @return boolean
 */
@Deprecated
Result unlockTxCoinData(Transaction tx, long newockTime) throws NulsException;

```

```

/**
 *
 * Deprecated method
 *
 * rollback unlockTxCoinData
 *
 * unlockTxCoinData
 * @param tx
 * @return boolean
 */
@Deprecated
Result rollbackUnlockTxCoinData(Transaction tx) throws NulsException;

/**
 * Get the entire network of UTXO
 *
 * UTXO
 * @return long
 */
long getWholeUTXO();

/**
 * get UTXO by key
 *
 * keyUTXO
 * @param owner
 * @return Coin
 */
Coin getUtxo(byte[] owner);

/**
 * get UTXO by key
 *
 * keyUTXO
 * @param address
 * @return Coin
 */
List<Coin> getAllUtxo(byte[] address);
}

```

182:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\ledger\src\main\java\io\nuls\ledger\util\LedgerUtil.java

```

package io.nuls.ledger.util;

import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.VarInt;

import java.util.Arrays;
import java.util.Base64;

/**
 * @desription:
 * @author: PierreLuo
 */
public class LedgerUtil {

    private final static int TX_HASH_LENGTH = NulsDigestData.HASH_LENGTH;

    public static byte[] getTxHashBytes(byte[] fromBytes) {
        if(fromBytes == null || fromBytes.length < TX_HASH_LENGTH) {
            return null;
        }
        byte[] txBytes = new byte[TX_HASH_LENGTH];
        System.arraycopy(fromBytes, 0, txBytes, 0, TX_HASH_LENGTH);
        return txBytes;
    }

    public static String getTxHash(byte[] fromBytes) {
        byte[] txBytes = getTxHashBytes(fromBytes);
        if(txBytes != null) {
            return Hex.encode(txBytes);
        }
        return null;
    }

    public static byte[] getIndexBytes(byte[] fromBytes) {
        if(fromBytes == null || fromBytes.length < TX_HASH_LENGTH) {
            return null;
        }
        int length = fromBytes.length - TX_HASH_LENGTH;
        byte[] indexBytes = new byte[length];

```

```

        System.arraycopy(fromBytes, TX_HASH_LENGTH, indexBytes, 0, length);
        return indexBytes;
    }

    public static Integer getIndex(byte[] fromBytes) {
        byte[] indexBytes = getIndexBytes(fromBytes);
        if(indexBytes != null) {
            VarInt varInt = new VarInt(indexBytes, 0);
            return Math.toIntExact(varInt.value);
        }
        return null;
    }

    public static String asString(byte[] bytes) {
        AssertUtil.canNotEmpty(bytes);
        return Base64.getEncoder().encodeToString(bytes);
    }

    public static byte[] asBytes(String string) {
        AssertUtil.canNotEmpty(string);
        return Base64.getDecoder().decode(string);
    }
}

```

```

183:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-
base\src\main\java\io\nuls\ledger\module\impl\UtxoLedgerModuleBootstrap.java
package io.nuls.ledger.module.impl;

```

```

import io.nuls.ledger.module.AbstractLedgerModule;

```

```

/**
 * @desription:
 * @author: PierreLuo
 */
public class UtxoLedgerModuleBootstrap extends AbstractLedgerModule {

    @Override
    public void init() {

```

```

    }

    @Override
    public void start() {

    }

    @Override
    public void shutdown() {

    }

    @Override
    public void destroy() {

    }

    @Override
    public String getInfo() {
        return null;
    }
}

```

184:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-base\src\main\java\io\nuls\ledger\service\impl\UtxoLedgerServiceImpl.java  
package io.nuls.ledger.service.impl;

```

import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.calc.LongUtils;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.map.MapUtil;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.db.model.Entry;
import io.nuls.db.service.BatchOperation;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;

```

```

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.*;
import io.nuls.kernel.script.P2PHKSignature;
import io.nuls.kernel.script.Script;
import io.nuls.kernel.script.TransactionSignature;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.kernel.utils.VarInt;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.constant.LedgerErrorCode;
import io.nuls.ledger.service.LedgerService;
import io.nuls.ledger.storage.service.UtxoLedgerTransactionStorageService;
import io.nuls.ledger.storage.service.UtxoLedgerUtxoStorageService;
import io.nuls.ledger.util.LedgerUtil;
import org.spongycastle.util.Arrays;

import java.io.IOException;
import java.util.*;

/**
 * @author: PierreLuo
 */
@Service
public class UtxoLedgerServiceImpl implements LedgerService {

    private final static String CLASS_NAME = UtxoLedgerServiceImpl.class.getName();

    @Autowired
    private UtxoLedgerUtxoStorageService utxoLedgerUtxoStorageService;
    @Autowired
    private UtxoLedgerTransactionStorageService utxoLedgerTransactionStorageService;
    @Autowired
    private ContractService contractService;

    @Override
    public Result saveTx(Transaction tx) throws NulsException {
        if (tx == null) {
            return Result.getFailed(LedgerErrorCode.NULL_PARAMETER);
        }
    }

```

```

try {
    // CoinData
    Result result = saveCoinData(tx);
    if (result.isFailed()) {
        Result rollbackResult = rollbackCoinData(tx);
        if (rollbackResult.isFailed()) {
            throw new NulsException(rollbackResult.getErrorCode());
        }
        return result;
    }
    //
    result = utxoLedgerTransactionStorageService.saveTx(tx);
    if (result.isFailed()) {
        Result rollbackResult = rollbackTx(tx);
        if (rollbackResult.isFailed()) {
            throw new NulsException(rollbackResult.getErrorCode());
        }
    }
    return result;
} catch (IOException e) {
    Log.error(e);
    Result rollbackResult = rollbackTx(tx);
    if (rollbackResult.isFailed()) {
        throw new NulsException(rollbackResult.getErrorCode());
    }
    return Result.getFailed(KernelErrorCode.IO_ERROR);
}
}

```

```

private Result saveCoinData(Transaction tx) throws IOException {
    CoinData coinData = tx.getCoinData();
    //TestLog+
    //    Log.info("=====" + tx.getClass().getSimpleName() + "hash-
    "+tx.getHash().getDigestHex());
    //TestLog-
    if (coinData != null) {
        BatchOperation batch = utxoLedgerUtxoStorageService.createWriteBatch();
        // utxo - from
        List<Coin> froms = coinData.getFrom();
        for (Coin from : froms) {
            //TestLog+
            //    Coin preFrom = utxoLedgerUtxoStorageService.getUtxo(from.());

```

```

//          if (preFrom != null) {
//              Log.info("height: +" + tx.getBlockHeight() + ", "+txHash-" + tx.getHash() + ", " +
Hex.encode(from.()));
//          }
//          Log.info("delete utxo:" + Hex.encode(from.()));
//          //TestLog-
//          batch.delete(from.getOwner());
//      }
//      // utxo - to
//      byte[] txHashBytes = tx.getHash().serialize();
//      List<Coin> tos = coinData.getTo();
//      for (int i = 0, length = tos.size(); i < length; i++) {
//          try {
//              byte[] owner = Arrays.concatenate(txHashBytes, new VarInt(i).encode());
//              Log.info("129 save utxo::" + Hex.encode(owner));
//              batch.put(owner, tos.get(i).serialize());
//          } catch (IOException e) {
//              Log.error(e);
//              return Result.getFailed(KernelErrorCode.IO_ERROR);
//          }
//      }
//      //
//      Result batchResult = batch.executeBatch();
//      if (batchResult.isFailed()) {
//          return batchResult;
//      }
//      }
//      return Result.getSuccess();
//  }

```

@Override

```

public Result rollbackTx(Transaction tx) throws NulsException {
    if (tx == null) {
        return Result.getFailed(LedgerErrorCode.NULL_PARAMETER);
    }
    try {
        // CoinData
        Result result = rollbackCoinData(tx);
        if (result.isFailed()) {
            Result recoveryResult = saveCoinData(tx);
            if (recoveryResult.isFailed()) {
                throw new NulsException(recoveryResult.getErrorCode());
            }
        }
    }
}

```



```

    }
    return result;
}
//
result = utxoLedgerTransactionStorageService.deleteTx(tx);
if (result.isFailed()) {
    Result recoveryResult = saveTx(tx);
    if (recoveryResult.isFailed()) {
        throw new NulsException(recoveryResult.getErrorCode());
    }
}
return result;
} catch (IOException e) {
    Log.error(e);
    Result rollbackResult = saveTx(tx);
    if (rollbackResult.isFailed()) {
        throw new NulsException(rollbackResult.getErrorCode());
    }
    return Result.getFailed(KernelErrorCode.IO_ERROR);
}
}

```

```

private Result rollbackCoinData(Transaction tx) throws IOException, NulsException {
    byte[] txHashBytes = tx.getHash().serialize();
    BatchOperation batch = utxoLedgerUtxoStorageService.createWriteBatch();
    CoinData coinData = tx.getCoinData();
    if (coinData != null) {
        // utxo - from
        List<Coin> froms = coinData.getFrom();
        Coin recovery;
        for (Coin from : froms) {
            try {
                NulsByteBuffer byteBuffer = new NulsByteBuffer(from.getOwner());

                NulsDigestData fromTxHash = byteBuffer.readHash();

                int fromIndex = (int) byteBuffer.readVarInt();

                Transaction fromTx = utxoLedgerTransactionStorageService.getTx(fromTxHash);
                recovery = fromTx.getCoinData().getTo().get(fromIndex);
                recovery.setFrom(from.getFrom());
                Log.info("rollback save utxo:::" + Hex.encode(from.()));
            }
        }
    }
}

```

```

        batch.put(from.getOwner(), recovery.serialize());
    } catch (IOException e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    }
}
// utxo - to
List<Coin> tos = coinData.getTo();
for (int i = 0, length = tos.size(); i < length; i++) {
    byte[] owner = Arrays.concatenate(txHashBytes, new VarInt(i).encode());
//    Log.info("" + Hex.encode(owner));
    batch.delete(owner);
}
//
Result batchResult = batch.executeBatch();
if (batchResult.isFailed()) {
    return batchResult;
}
}
return Result.getSuccess();
}

```

@Override

```

public Transaction getTx(NulsDigestData hash) {
    if (hash == null) {
        return null;
    }
    return utxoLedgerTransactionStorageService.getTx(hash);
}

```

@Override

```

public Transaction getTx(byte[] txHashBytes) {
    if (txHashBytes == null) {
        return null;
    }
    NulsDigestData digestData = new NulsDigestData();
    try {
        digestData.parse(txHashBytes, 0);
    } catch (Exception e) {
        return null;
    }
    return getTx(digestData);
}

```

```

}

/**
 * txListtoListUTXO
 * coinData txList txList to coinData from
 *
 *
 * @return ValidateResult
 */
@Override
public ValidateResult verifyCoinData(Transaction transaction, Map<String, Coin>
temporaryToMap, Set<String> temporaryFromSet) {
    return verifyCoinData(transaction, temporaryToMap, temporaryFromSet, null);
}

/**
 * txListtoListUTXO
 * coinData txList txList to coinData from
 *
 * bestHeight is used when switch chain.
 *
 * @return ValidateResult
 */
@Override
public ValidateResult verifyCoinData(Transaction transaction, Map<String, Coin>
temporaryToMap, Set<String> temporaryFromSet, Long bestHeight) {

    if (transaction == null || transaction.getCoinData() == null) {
        return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.NULL_PARAMETER);
    }

    try {
        /**

        */
        CoinData coinData = transaction.getCoinData();
        List<Coin> froms = coinData.getFrom();
        int fromSize = froms.size();
        TransactionSignature transactionSignature = new TransactionSignature();
        //
        if (transaction.needVerifySignature() && fromSize > 0) {

```

```

        try {
            transactionSignature.parse(transaction.getTransactionSignature(),0);
        } catch (NulsException e){
            return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.LEDGER_P2PKH_SCRIPT_ERROR);
        }
    }
    // Set
    if (temporaryFromSet == null) {
        temporaryFromSet = new HashSet<>();
    }

    Na fromTotal = Na.ZERO;
    byte[] fromBytes;
    // txListutxo
    Coin fromOfFromCoin = null;
    byte[] fromAddressBytes = null;
    /**
     * fromAddressBytes
     */
    byte[] realAddressBytes = null;
    for (int i = 0; i < froms.size(); i++) {
        Coin from = froms.get(i);
        fromBytes = from.getOwner();
        // , coinDatafromUTXOUTXO
        fromOfFromCoin = utxoLedgerUtxoStorageService.getUtxo(fromBytes);

        // txListUTXO
        if (temporaryToMap != null && fromOfFromCoin == null) {
            fromOfFromCoin = temporaryToMap.get(asString(fromBytes));
        }
        if (null == fromOfFromCoin) {
            // txListto(txList)
            if (null !=
utxoLedgerTransactionStorageService.getTxBytes(LedgerUtil.getTxHashBytes(fromBytes))) {
                return ValidateResult.getFailedResult(CLASS_NAME,
TransactionErrorCode.TRANSACTION_REPEATED);
            } else {
                return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.ORPHAN_TX);
            }
        } else {

```

```

        fromAddressBytes = fromOfFromCoin.getOwner();
        realAddressBytes = fromOfFromCoin.getAddress();
        // pierre add ()fromAdress()
        if (transaction.getType() != ContractConstant.TX_TYPE_CONTRACT_TRANSFER) {
            boolean isContractAddress =
contractService.isContractAddress(realAddressBytes);
            if (isContractAddress) {
                return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.DATA_ERROR);
            }
        }

        // hash160hash160utxo
        boolean signitureValidFlag = false;
        if(transaction.needVerifySignature()){
            if(transactionSignature != null){
                if(fromAddressBytes != null && fromAddressBytes.length !=
Address.ADDRESS_LENGTH && transactionSignature.getScripts() != null
                && transactionSignature.getScripts().size()>0){
                    Script scriptPubkey = new Script(fromAddressBytes);
                    for (Script scriptSig:transactionSignature.getScripts()) {
                        signitureValidFlag =
scriptSig.correctlyNulsSpends(transaction,0,scriptPubkey);
                        if(signitureValidFlag) {
                            break;
                        }
                    }
                }
            }
            else {
                if(transactionSignature.getP2PHKSignatures() != null &&
transactionSignature.getP2PHKSignatures().size() != 0){
                    for (P2PHKSignature
signature:transactionSignature.getP2PHKSignatures()) {
                        signitureValidFlag =
AddressTool.checkPublicKeyHash(realAddressBytes,signature.getSignerHash160());
                        if(signitureValidFlag) {
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

        if(!signatureValidFlag){
            Log.warn("public key hash160 check error.");
            return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.INVALID_INPUT);
        }
    }
    if (java.util.Arrays.equals(realAddressBytes,
NulsConstant.BLACK_HOLE_ADDRESS)) {
        return ValidateResult.getFailedResult(CLASS_NAME,
KernelErrorCode.ADDRESS_IS_BLOCK_HOLE);
    }
    if (NulsContext.DEFAULT_CHAIN_ID !=
SerializeUtils.bytes2Short(realAddressBytes)) {
        return ValidateResult.getFailedResult(CLASS_NAME,
KernelErrorCode.ADDRESS_IS_NOT_BELONGS_TO_CHAIN);
    }
}
//
if (!transaction.isUnlockTx()) {
    //
    if (bestHeight == null) {
        if (!fromOfFromCoin.usable()) {
            return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.UTXO_UNUSABLE);
        }
    } else {
        if (!fromOfFromCoin.usable(bestHeight)) {
            return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.UTXO_UNUSABLE);
        }
    }
} else {
    //
    if (fromOfFromCoin.getLockTime() != -1) {
        return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.UTXO_STATUS_CHANGE);
    }
}

// fromUtxotxListfromUtxo
if (temporaryFromSet != null && !temporaryFromSet.add(asString(fromBytes))) {
    if (i > 0) {

```

```

        for (int x = 0; x < i; x++) {
            Coin theFrom = froms.get(i);
            temporaryFromSet.remove(asString(theFrom.getOwner()));
        }
    }
    return ValidateResult.getFailedResult(CLASS_NAME,
TransactionErrorCode.TRANSACTION_REPEATED);
}

// from
if (!(fromOfFromCoin.getNa().equals(from.getNa()) && fromOfFromCoin.getLockTime()
== from.getLockTime())) {
    return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.DATA_ERROR);
}

fromTotal = fromTotal.add(fromOfFromCoin.getNa());
from.setFrom(fromOfFromCoin);
}

List<Coin> tos = coinData.getTo();
Na toTotal = Na.ZERO;
byte[] txBytes = transaction.getHash().serialize();
for (int i = 0; i < tos.size(); i++) {
    Coin to = tos.get(i);

    // nulsnuls(CoinBase)
    if(ContractConstant.TX_TYPE_CALL_CONTRACT != transaction.getType()
        && AddressTool.validContractAddress(to.getOwner())) {
        Log.error("Ledger verify error: {}. ",
ContractErrorCode.NON_CONTRACTUAL_TRANSACTION_NO_TRANSFER.getEnMsg());
        return ValidateResult.getFailedResult(CLASS_NAME,
ContractErrorCode.NON_CONTRACTUAL_TRANSACTION_NO_TRANSFER);
    }

    toTotal = toTotal.add(to.getNa());
    if (temporaryToMap != null) {
        temporaryToMap.put(asString(ArraysTool.concatenate(txBytes, new
VarInt(i).encode())), to);
    }
}
//

```

```

        if (fromTotal.compareTo(toTotal) < 0) {
            return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.INVALID_AMOUNT);
        }
    } catch (Exception e) {
        Log.error(e);
        return ValidateResult.getFailedResult(CLASS_NAME,
KernelErrorCode.SYS_UNKOWN_EXCEPTION);
    }

    return ValidateResult.getSuccessResult();
}

/**
 * resultdata
 *
 * @return ValidateResult<List
Transaction>> */ @Override public
ValidateResult<List<Transaction>> verifyDoubleSpend(Block block) {
    if (block == null) {
        return ValidateResult.getFailedResult(CLASS_NAME,
KernelErrorCode.BLOCK_IS_NULL);
    }
    return verifyDoubleSpend(block.getTxs());
}

/**
 * resultdata
 *
 * @return ValidateResult<List
Transaction>> */ @Override public
ValidateResult<List<Transaction>> verifyDoubleSpend(List<Transaction> txList) {
    if (txList == null) {
        return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.NULL_PARAMETER);
    }
    // HashMap
    int initialCapacity = 0;
    CoinData coinData;
    for (Transaction tx : txList) {

```



```

        coinData = tx.getCoinData();
        if (coinData == null) {
            continue;
        }
        initialCapacity += tx.getCoinData().getFrom().size();
    }
    initialCapacity = MapUtil.tableSizeFor(initialCapacity) << 1;
    HashMap<String, Transaction> fromMap = new HashMap<>(initialCapacity);
    List<Coin> froms;
    Transaction prePutTx;
    // fromCoin
    for (Transaction tx : txList) {
        coinData = tx.getCoinData();
        if (coinData == null) {
            continue;
        }
        froms = coinData.getFrom();
        for (Coin from : froms) {
            prePutTx = fromMap.put(asString(from.getOwner()), tx);
            // coinmap
            if (prePutTx != null) {
                List<Transaction> resultList = new ArrayList<>(2);
                resultList.add(prePutTx);
                resultList.add(tx);
                ValidateResult validateResult = ValidateResult.getFailedResult(CLASS_NAME,
TransactionErrorCode.TRANSACTION_REPEATED);
                validateResult.setData(resultList);
                return validateResult;
            }
        }
    }
    return ValidateResult.getSuccessResult();
}

private String asString(byte[] bytes) {
    AssertUtil.canNotEmpty(bytes);
    return Base64.getEncoder().encodeToString(bytes);
}

```

@Override

```

public Result unlockTxCoinData(Transaction tx, long newockTime) throws NulsException {
    if (tx == null || tx.getCoinData() == null) {

```

```

        return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.NULL_PARAMETER);
    }
    try {
        CoinData coinData = tx.getCoinData();
        List<Coin> tos = coinData.getTo();
        boolean isExistLockUtxo = false;
        Coin needUnLockUtxo = null;
        int needUnLockUtxoIndex = 0;
        for (Coin to : tos) {
            if (to.getLockTime() == -1) {
                isExistLockUtxo = true;
                needUnLockUtxo = to;
                break;
            }
            needUnLockUtxoIndex++;
        }
        if (!isExistLockUtxo) {
            return ValidateResult.getFailedResult(CLASS_NAME,
LedgerErrorCode.UTXO_STATUS_CHANGE);
        }
        byte[] txHashBytes = txHashBytes = tx.getHash().serialize();
        Coin needUnLockUtxoNew = new Coin(needUnLockUtxo.getOwner(),
needUnLockUtxo.getNa(), newockTime);
        needUnLockUtxoNew.setFrom(needUnLockUtxo.getFrom());
        Result result = utxoLedgerUtxoStorageService.saveUtxo(Arrays.concatenate(txHashBytes,
new VarInt(needUnLockUtxoIndex).encode()), needUnLockUtxoNew);
        if (result.isFailed()) {
            Result rollbackResult = rollbackUnlockTxCoinData(tx);
            if (rollbackResult.isFailed()) {
                throw new NulsException(rollbackResult.getErrorCode());
            }
        }
        return result;
    } catch (IOException e) {
        Result rollbackResult = rollbackUnlockTxCoinData(tx);
        if (rollbackResult.isFailed()) {
            throw new NulsException(rollbackResult.getErrorCode());
        }
        Log.error(e);
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    }
}

```

```

    }

    @Override
    public Result rollbackUnlockTxCoinData(Transaction tx) throws NulsException {
        if (tx == null || tx.getCoinData() == null) {
            return ValidateResult.getFailedResult(CLASS_NAME,
                LedgerErrorCode.NULL_PARAMETER);
        }
        try {
            CoinData coinData = tx.getCoinData();
            List<Coin> tos = coinData.getTo();
            boolean isExistLockUtxo = false;
            Coin needUnlockUtxo = null;
            int needUnlockUtxoIndex = 0;
            for (Coin to : tos) {
                if (to.getLockTime() == -1) {
                    isExistLockUtxo = true;
                    needUnlockUtxo = to;
                    break;
                }
                needUnlockUtxoIndex++;
            }
            if (!isExistLockUtxo) {
                return ValidateResult.getFailedResult(CLASS_NAME,
                    LedgerErrorCode.UTXO_STATUS_CHANGE);
            }
            byte[] txHashBytes = tx.getHash().serialize();
            Result result = utxoLedgerUtxoStorageService.saveUtxo(Arrays.concatenate(txHashBytes,
                new VarInt(needUnlockUtxoIndex).encode()), needUnlockUtxo);
            if (result.isFailed()) {
                throw new NulsException(result.getErrorCode());
            }
            return result;
        } catch (IOException e) {
            Log.error(e);
            throw new NulsException(KernelErrorCode.IO_ERROR);
        }
    }
}

```

@Override

```

public long getWholeUTXO() {
    long result = 0L;

```

```

List<byte[]> list = utxoLedgerUtxoStorageService.getAllUtxoBytes();
if (list == null || list.size() == 0) {
    return result;
}
Coin coin = null;
try {
    for (byte[] utxoBytes : list) {
        if (utxoBytes != null) {
            coin = new Coin();
            coin.parse(utxoBytes, 0);
            result = LongUtils.add(result, coin.getNa().getValue());
        }
    }
    return result;
} catch (NulsException e) {
    Log.error(e);
    return 0L;
}
}

```

@Override

```

public Coin getUtxo(byte[] owner) {
    if (owner == null) {
        return null;
    }
    return utxoLedgerUtxoStorageService.getUtxo(owner);
}

```

/\*\*

\* get UTXO by key

\*

\* keyUTXO

\* @param address

\* @return Coin

\*/

@Override

```

public List<Coin> getAllUtxo(byte[] address){
    List<Coin> coinList = new ArrayList<>();
    Collection<Entry<byte[], byte[]>> rawList
=utxoLedgerUtxoStorageService.getAllUtxoEntryBytes();
    for (Entry<byte[], byte[]> coinEntry : rawList) {

```

```

Coin coin = new Coin();
try {
    coin.parse(coinEntry.getValue(), 0);
} catch (NulsException e) {
    Log.info("parse coin form db error");
    continue;
}
if (java.util.Arrays.equals(coin.getAddress(), address)) {
    coin.setTempOwner(coin.getOwner());
    coin.setOwner(coinEntry.getKey());
    coinList.add(coin);
}
}
return coinList;
}
}

```

185:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-rpc\src\main\java\io\nuls\ledger\rpc\cmd\GetTxProcessor.java  
\*/

```
package io.nuls.ledger.rpc.cmd;
```

```

import io.nuls.core.tools.date.DateUtil;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

```

```

import java.util.Date;
import java.util.List;
import java.util.Map;

```

```
public class GetTxProcessor implements CommandProcessor {
```

```
    private RestFulUtils restFul = RestFulUtils.getInstance();
```

```

@Override
public String getCommand() {
    return "gettx";
}

```

```

@Override
public String getHelp() {
    CommandBuilder bulider = new CommandBuilder();
    bulider.newLine(getCommandDescription())
        .newLine("\t<hash> transaction hash -required");
    return bulider.toString();
}

```

```

@Override
public String getCommandDescription() {
    return "gettx <hash> --get the transaction information by txhash";
}

```

```

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length != 2) {
        return false;
    }
    return true;
}

```

```

@Override
public CommandResult execute(String[] args) {
    String hash = args[1];
    if(StringUtils.isBlank(hash)) {
        return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
    }
    RpcClientResult result = restFul.get("/accountledger/tx/" + hash, null);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    Map<String, Object> map = (Map)result.getData();
    map.put("fee", CommandHelper.naToNuls(map.get("fee")));
    map.put("value", CommandHelper.naToNuls(map.get("value")));
    map.put("time", DateUtil.convertDate(new Date((Long)map.get("time"))));
    map.put("status", statusExplains((Integer)map.get("status")));
}

```

```

map.put("type", CommandHelper.txTypeExplain((Integer)map.get("type")));

List<Map<String, Object>> inputs = (List<Map<String, Object>>)map.get("inputs");
for(Map<String, Object> input : inputs){
    input.put("value", CommandHelper.naToNuls(input.get("value")));
}
map.put("inputs", inputs);
List<Map<String, Object>> outputs = (List<Map<String, Object>>)map.get("outputs");
for(Map<String, Object> output : outputs){
    output.put("value", CommandHelper.naToNuls(output.get("value")));
    output.put("status", statusExplainForOutPut((Integer) output.get("status")));
}
map.put("outputs", outputs);
result.setData(map);
return CommandResult.getResult(result);
}

```

```

private String statusExplain(Integer status){
    if(status == 0){
        return "unConfirm";
    }
    if(status == 1){
        return "confirm";
    }
    return "unknown";
}

```

```

/**
 * 0:usable(), 1:timeLock(), 2:consensusLock(), 3:spent()
 * @param status
 * @return
 */

```

```

private String statusExplainForOutPut(Integer status){
    if(status == 0){
        return "usable";
    }
    if(status == 1){
        return "timeLock";
    }
    if(status == 2){
        return "consensusLock";
    }
}

```

```

        if(status == 3){
            return"spent";
        }
        return "unknown";
    }
}

```

186:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-rpc\src\main\java\io\nuls\ledger\rpc\model\AccountUtxoDto.java

```

package io.nuls.ledger.rpc.model;

```

```

import io.nuls.kernel.model.Coin;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

```

```

import java.util.List;

```

```

/**
 * @author: PierreLuo
 */
@ApiModel(value = "AccountUtxoDtoJSON")
public class AccountUtxoDto {

    @ApiModelProperty(name = "utxoDtoList", value = "")
    private List<UtxoDto> utxoDtoList;

    public List<UtxoDto> getUtxoDtoList() {
        return utxoDtoList;
    }

    public void setUtxoDtoList(List<UtxoDto> utxoDtoList) {
        this.utxoDtoList = utxoDtoList;
    }
}

```

187:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-rpc\src\main\java\io\nuls\ledger\rpc\model\Holder.java

```

public double getTotalNuls() {
    return totalNuls;
}

```



```

public void setTotalNuls(double totalNuls) {
    this.totalNuls = totalNuls;
}

public double getLockedNuls() {
    return lockedNuls;
}

public void setLockedNuls(double lockedNuls) {
    this.lockedNuls = lockedNuls;
}

public void addTotal(double value) {
    totalNuls = DoubleUtils.sum(totalNuls, value);
}

```

```

public void addLocked(double value) {
    lockedNuls = DoubleUtils.sum(lockedNuls, value);
}

```

@Override

```

public int compareTo(Object o) {
    if (null == o || !(o instanceof Holder)) {
        return -1;
    }
    Holder obj = (Holder) o;
    if (obj.getTotalNuls() > this.getTotalNuls()) {
        return 1;
    }
    if (obj.getTotalNuls() < this.getTotalNuls()) {
        return -1;
    }
    return 0;
}
}

```

188:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-rpc\src\main\java\io\nuls\ledger\rpc\model\HolderDto.java

```

    this.lockedNuls = DoubleUtils.getRoundStr(holder.getLockedNuls(),8,true);
}

public String getAddress() {

```

```

        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getTotalNuls() {
        return totalNuls;
    }

    public void setTotalNuls(String totalNuls) {
        this.totalNuls = totalNuls;
    }

    public String getLockedNuls() {
        return lockedNuls;
    }

    public void setLockedNuls(String lockedNuls) {
        this.lockedNuls = lockedNuls;
    }
}

189:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-
rpc\src\main\java\io\nuls\ledger\rpc\model\InputDto.java
*/

package io.nuls.ledger.rpc.model;

import io.nuls.core.tools.crypto.Base58;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.ledger.util.LedgerUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "inputJSON")
public class InputDto {

    @ApiModelProperty(name = "fromHash", value = "outputtxHash")
    private String fromHash;

```

```
@ApiModelProperty(name = "fromIndex", value = "outputoutIndex")
private Integer fromIndex;
```

```
@ApiModelProperty(name = "address", value = "")
private String address;
```

```
@ApiModelProperty(name = "value", value = "")
private Long value;
```

```
public InputDto(Coin input) {
    this.fromHash = LedgerUtil.getTxHash(input.getOwner());
    this.fromIndex = LedgerUtil.getIndex(input.getOwner());
    //this.address = AddressTool.getStringAddressByBytes(input.getFrom().());
    this.address = AddressTool.getStringAddressByBytes(input.getFrom().getAddress());
    this.value = input.getFrom().getNa().getValue();
}
```

```
public String getAddress() {
    return address;
}
```

```
public void setAddress(String address) {
    this.address = address;
}
```

```
public Long getValue() {
    return value;
}
```

```
public void setValue(Long value) {
    this.value = value;
}
```

```
public String getFromHash() {
    return fromHash;
}
```

```
public void setFromHash(String fromHash) {
    this.fromHash = fromHash;
}
```

```

    public Integer getFromIndex() {
        return fromIndex;
    }

    public void setFromIndex(Integer fromIndex) {
        this.fromIndex = fromIndex;
    }
}

```

190:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-rpc\src\main\java\io\nuls\ledger\rpc\model\OutputDto.java  
 \*/

```

package io.nuls.ledger.rpc.model;

import io.nuls.core.tools.crypto.Base58;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "outputJSON")
public class OutputDto {

    @ApiModelProperty(name = "txHash", value = "hash")
    private String txHash;

    @ApiModelProperty(name = "index", value = "")
    private Integer index;

    @ApiModelProperty(name = "address", value = "")
    private String address;

    @ApiModelProperty(name = "script", value = "")
    private String script;

    @ApiModelProperty(name = "value", value = "")
    private Long value;

    @ApiModelProperty(name = "lockTime", value = "")
    private Long lockTime;

```

```

@ApiModelProperty(name = "status",
    value = " 0:usable(), 1:timeLock(), 2:consensusLock(), 3:spent()")
private Integer status;

public OutputDto(Coin output) {
    this.address = AddressTool.getStringAddressByBytes(output.getAddress());
    this.script = AddressTool.getStringAddressByBytes(output.getOwner());
    this.value = output.getNa().getValue();
    this.lockTime = output.getLockTime();
}

public Integer getIndex() {
    return index;
}

public void setIndex(Integer index) {
    this.index = index;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public Long getValue() {
    return value;
}

public void setValue(Long value) {
    this.value = value;
}

public Long getLockTime() {
    return lockTime;
}

public void setLockTime(Long lockTime) {
    this.lockTime = lockTime;
}

```

```

public Integer getStatus() {
    return status;
}

public void setStatus(Integer status) {
    this.status = status;
}

public String getTxHash() {
    return txHash;
}

public void setTxHash(String txHash) {
    this.txHash = txHash;
}
}

191:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-
rpc\src\main\java\io\nuls\ledger\rpc\model\TokenInfoDto.java
}

public void setTotalNuls(String totalNuls) {
    this.totalNuls = totalNuls;
}

public String getLockedNuls() {
    return lockedNuls;
}

public void setLockedNuls(String lockedNuls) {
    this.lockedNuls = lockedNuls;
}

public List<HolderDto> getAddressList() {
    return addressList;
}

public void setAddressList(List<HolderDto> addressList) {
    this.addressList = addressList;
}
}

```

```
192:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-  
rpc\src\main\java\io\nuls\ledger\rpc\model\TransactionDto.java  
*/
```

```
package io.nuls.ledger.rpc.model;
```

```
import io.nuls.core.tools.crypto.Hex;  
import io.nuls.kernel.cfg.NulsConfig;  
import io.nuls.kernel.constant.TxStatusEnum;  
import io.nuls.kernel.context.NulsContext;  
import io.nuls.kernel.model.Coin;  
import io.nuls.kernel.model.CoinData;  
import io.nuls.kernel.model.Transaction;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;
```

```
import java.io.UnsupportedEncodingException;  
import java.util.ArrayList;  
import java.util.List;
```

```
/**
```

```
 * @author: PierreLuo
```

```
 */
```

```
@ApiModel(value = "transactionDtoJSON")
```

```
public class TransactionDto {
```

```
    @ApiModelProperty(name = "hash", value = "hash")  
    private String hash;
```

```
    @ApiModelProperty(name = "type", value = " ")  
    private Integer type;
```

```
    @ApiModelProperty(name = "time", value = "")  
    private Long time;
```

```
    @ApiModelProperty(name = "blockHeight", value = "")  
    private Long blockHeight;
```

```
    @ApiModelProperty(name = "fee", value = "")  
    private Long fee;
```

```
@ApiModelProperty(name = "value", value = "")
```

```
private Long value;
```

```
@ApiModelProperty(name = "remark", value = "")
```

```
private String remark;
```

```
@ApiModelProperty(name = "scriptSig", value = "")
```

```
private String scriptSig;
```

```
@ApiModelProperty(name = "status", value = " 0:unConfirm(), 1:confirm()")
```

```
private Integer status;
```

```
@ApiModelProperty(name = "confirmCount", value = "")
```

```
private Long confirmCount;
```

```
@ApiModelProperty(name = "size", value = "")
```

```
private int size;
```

```
@ApiModelProperty(name = "inputs", value = "")
```

```
private List<InputDto> inputs;
```

```
@ApiModelProperty(name = "outputs", value = "")
```

```
private List<OutputDto> outputs;
```

```
public TransactionDto(Transaction tx) {
```

```
    long bestBlockHeight = NulsContext.getInstance().getBestBlock().getHeader().getHeight();
```

```
    this.hash = tx.getHash().getDigestHex();
```

```
    this.type = tx.getType();
```

```
    this.time = tx.getTime();
```

```
    this.blockHeight = tx.getBlockHeight();
```

```
    this.fee = tx.getFee().getValue();
```

```
    this.size = tx.getSize();
```

```
    if (this.blockHeight > 0 || TxStatusEnum.CONFIRMED.equals(tx.getStatus())) {
```

```
        this.confirmCount = bestBlockHeight - this.blockHeight;
```

```
    } else {
```

```
        this.confirmCount = 0L;
```

```
    }
```

```
    if (TxStatusEnum.CONFIRMED.equals(tx.getStatus())) {
```

```
        this.status = 1;
```

```
    } else {
```

```
        this.status = 0;
```

```
    }
```



```

if (tx.getRemark() != null) {
    try {
        this.setRemark(new String(tx.getRemark(), NulsConfig.DEFAULT_ENCODING));
    } catch (UnsupportedEncodingException e) {
        this.setRemark(Hex.encode(tx.getRemark()));
    }
}
if (tx.getTransactionSignature() != null) {
    this.setScriptSig(Hex.encode(tx.getTransactionSignature()));
}

CoinData coinData = tx.getCoinData();
List<InputDto> inputs = new ArrayList<>();
if(coinData != null) {
    List<Coin> froms = coinData.getFrom();
    for(Coin from : froms) {
        inputs.add(new InputDto(from));
    }
}
this.inputs = inputs;
}

public String getHash() {
    return hash;
}

public void setHash(String hash) {
    this.hash = hash;
}

public Integer getType() {
    return type;
}

public void setType(Integer type) {
    this.type = type;
}

public Long getTime() {
    return time;
}

```

```
public void setTime(Long time) {  
    this.time = time;  
}
```

```
public Long getBlockHeight() {  
    return blockHeight;  
}
```

```
public void setBlockHeight(Long blockHeight) {  
    this.blockHeight = blockHeight;  
}
```

```
public Long getFee() {  
    return fee;  
}
```

```
public void setFee(Long fee) {  
    this.fee = fee;  
}
```

```
public Long getValue() {  
    return value;  
}
```

```
public void setValue(Long value) {  
    this.value = value;  
}
```

```
public List<InputDto> getInputs() {  
    return inputs;  
}
```

```
public void setInputs(List<InputDto> inputs) {  
    this.inputs = inputs;  
}
```

```
public List<OutputDto> getOutputs() {  
    return outputs;  
}
```

```
public void setOutputs(List<OutputDto> outputs) {
```

```
    this.outputs = outputs;
}

public String getRemark() {
    return remark;
}

public void setRemark(String remark) {
    this.remark = remark;
}

public String getScriptSig() {
    return scriptSig;
}

public void setScriptSig(String scriptSig) {
    this.scriptSig = scriptSig;
}

public Integer getStatus() {
    return status;
}

public void setStatus(Integer status) {
    this.status = status;
}

public Long getConfirmCount() {
    return confirmCount;
}

public void setConfirmCount(Long confirmCount) {
    this.confirmCount = confirmCount;
}

public int getSize() {
    return size;
}

public void setSize(int size) {
    this.size = size;
}
```

```
}
```

```
193:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-  
rpc\src\main\java\io\nuls\ledger\rpc\model\UtxoDto.java  
package io.nuls.ledger.rpc.model;
```

```
import io.nuls.kernel.model.Coin;  
import io.nuls.ledger.util.LedgerUtil;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;
```

```
/**
```

```
 * @author: PierreLuo
```

```
 */
```

```
@ApiModel(value = "UtxoDtoJSON")
```

```
public class UtxoDto {
```

```
    @ApiModelProperty(name = "txHash", value = "hash")
```

```
    private String txHash;
```

```
    @ApiModelProperty(name = "txIndex", value = "")
```

```
    private Integer txIndex;
```

```
    @ApiModelProperty(name = "value", value = "")
```

```
    private Long value;
```

```
    @ApiModelProperty(name = "lockTime", value = "")
```

```
    private Long lockTime;
```

```
    public UtxoDto(Coin coin) {
```

```
        this.txHash = LedgerUtil.getTxHash(coin.getOwner());
```

```
        this.txIndex = LedgerUtil.getIndex(coin.getOwner());
```

```
        this.value = coin.getNa().getValue();
```

```
        this.lockTime = coin.getLockTime();
```

```
    }
```

```
    public String getTxHash() {
```

```
        return txHash;
```

```
    }
```

```
    public void setTxHash(String txHash) {
```

```

        this.txHash = txHash;
    }

    public Integer getTxIndex() {
        return txIndex;
    }

    public void setTxIndex(Integer txIndex) {
        this.txIndex = txIndex;
    }

    public Long getValue() {
        return value;
    }

    public void setValue(Long value) {
        this.value = value;
    }

    public Long getLockTime() {
        return lockTime;
    }

    public void setLockTime(Long lockTime) {
        this.lockTime = lockTime;
    }
}

```

194:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-rpc\src\main\java\io\nuls\ledger\rpc\resource\TransactionResource.java  
package io.nuls.ledger.rpc.resource;

```

import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.constant.TxStatusEnum;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.func.TimeService;

```

```

import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.VarInt;
import io.nuls.ledger.constant.LedgerErrorCode;
import io.nuls.ledger.rpc.model.InputDto;
import io.nuls.ledger.rpc.model.OutputDto;
import io.nuls.ledger.rpc.model.TransactionDto;
import io.nuls.ledger.service.LedgerService;
import io.nuls.ledger.storage.service.UtxoLedgerUtxoStorageService;
import io.nuls.ledger.util.LedgerUtil;
import io.swagger.annotations.*;
import org.spongycastle.util.Arrays;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.io.IOException;
import java.util.*;

/**
 * @description:
 * @author: PierreLuo
 */
@Path("/tx")
@Api(value = "/transaction", description = "transaction")
@Component
public class TransactionResource {

    @Autowired
    private LedgerService ledgerService;
    @Autowired
    private UtxoLedgerUtxoStorageService utxoLedgerUtxoStorageService;

    @GET
    @Path("/hash/{hash}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "hash")
    @ApiResponse(value = {
        @ApiResponse(code = 200, message = "success", response = TransactionDto.class)
    })
    public RpcClientResult getTxByHash(@ApiParam(name="hash", value="hash", required = true)
        @PathParam("hash") String hash) {

```

```

if (StringUtils.isBlank(hash)) {
    return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
}
if (!NulsDigestData.validHash(hash)) {
    return Result.getFailed(LedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
}
Result result = null;
try {
    Transaction tx = ledgerService.getTx(NulsDigestData.fromDigestHex(hash));
    if (tx == null) {
        result = Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
    } else {
        tx.setStatus(TxStatusEnum.CONFIRMED);
        TransactionDto txDto = null;
        CoinData coinData = tx.getCoinData();
        if(coinData != null) {
            // from
            List<Coin> froms = coinData.getFrom();
            if(froms != null && froms.size() > 0) {
                byte[] fromHash, owner;
                int fromIndex;
                NulsDigestData fromHashObj;
                Transaction fromTx;
                Coin fromUtxo;
                for(Coin from : froms) {
                    owner = from.getOwner();
                    // ownertxHashindex
                    fromHash = LedgerUtil.getTxHashBytes(owner);
                    fromIndex = LedgerUtil.getIndex(owner);
                    // from UTXO
                    fromHashObj = new NulsDigestData();
                    fromHashObj.parse(fromHash,0);
                    fromTx = ledgerService.getTx(fromHashObj);
                    fromUtxo = fromTx.getCoinData().getTo().get(fromIndex);
                    from.setFrom(fromUtxo);
                }
            }
            txDto = new TransactionDto(tx);
            List<OutputDto> outputDtoList = new ArrayList<>();
            // to
            List<Coin> tos = coinData.getTo();
            if(tos != null && tos.size() > 0) {

```

```

byte[] txHashBytes = tx.getHash().serialize();
String txHash = hash;
OutputDto outputDto = null;
Coin to, temp;
long bestHeight = NulsContext.getInstance().getBestHeight();
long currentTime = TimeService.currentTimeMillis();
long lockTime;
for(int i = 0, length = tos.size(); i < length; i++) {
    to = tos.get(i);
    outputDto = new OutputDto(to);
    outputDto.setTxHash(txHash);
    outputDto.setIndex(i);
    temp =
utxoLedgerUtxoStorageService.getUtxo(Arrays.concatenate(txHashBytes, new
VarInt(i).encode()));
    if(temp == null) {
        //
        outputDto.setStatus(3);
    } else {
        lockTime = temp.getLockTime();
        if (lockTime < 0) {
            //
            outputDto.setStatus(2);
        } else if (lockTime == 0) {
            //
            outputDto.setStatus(0);
        } else if (lockTime > NulsConstant.BLOCKHEIGHT_TIME_DIVIDE) {
            //
            if (lockTime > currentTime) {
                //
                outputDto.setStatus(1);
            } else {
                //
                outputDto.setStatus(0);
            }
        } else {
            //
            if (lockTime > bestHeight) {
                //
                outputDto.setStatus(1);
            } else {
                //

```



```

        outputDto.setStatus(0);
    }
}
outputDtoList.add(outputDto);
}
}
txDto.setOutputs(outputDtoList);
//
calTransactionValue(txDto);
}
result = Result.getSuccess();
result.setData(txDto);
}
} catch (NulsRuntimeException e) {
    Log.error(e);
    result = Result.getFailed(e.getErrorCode());
} catch (Exception e) {
    Log.error(e);
    result = Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION);
}
return result.toRpcClientResult();
}

```

```

/**
 *
 * Calculate the actual amount of the transaction.
 *
 * @param txDto
 */

```

```

private void calTransactionValue(TransactionDto txDto) {
    if(txDto == null) {
        return;
    }
    List<InputDto> inputDtoList = txDto.getInputs();
    Set<String> inputAdressSet = new HashSet<>(inputDtoList.size());
    for(InputDto inputDto : inputDtoList) {
        inputAdressSet.add(inputDto.getAddress());
    }
    Na value = Na.ZERO;
    List<OutputDto> outputDtoList = txDto.getOutputs();
    for(OutputDto outputDto : outputDtoList) {

```

```

        if(inputAdressSet.contains(outputDto.getAddress())) {
            continue;
        }
        value = value.add(Na.valueOf(outputDto.getValue()));
    }
    txDto.setValue(value.getValue());
}

@GET
@Path("/bytes")
@Produces(MediaType.APPLICATION_JSON)
public RpcClientResult getTxBytes(@QueryParam("hash") String hash) throws IOException {
    Result result;
    if (!NulsDigestData.validHash(hash)) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Transaction tx = null;
    try {
        tx = ledgerService.getTx(NulsDigestData.fromDigestHex(hash));
    } catch (NulsException e) {
        Log.error(e);
    }
    if (tx == null) {
        result = Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
    } else {
        result = Result.getSuccess();
        Map<String, String> map = new HashMap<>();
        map.put("value", Base64.getEncoder().encodeToString(tx.serialize()));
        result.setData(map);
    }
    return result.toRpcClientResult();
}
}

```

195:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-rpc\src\main\java\io\nuls\ledger\rpc\resource\UtxoResource.java

```

package io.nuls.ledger.rpc.resource;

```

```

import io.nuls.core.tools.calc.DoubleUtils;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.db.model.Entry;

```

```

import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.ledger.constant.LedgerErrorCode;
import io.nuls.ledger.rpc.model.*;
import io.nuls.ledger.storage.service.UtxoLedgerUtxoStorageService;
import io.nuls.ledger.storage.util.CoinComparator;
import io.swagger.annotations.*;

```

```

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.util.*;

```

```

/**
 * @desription:
 * @author: PierreLuo
 */
@Path("/utxo")
@Api(value = "/utxo", description = "utxo")
@Component
public class UtxoResource {

    @Autowired
    private UtxoLedgerUtxoStorageService utxoLedgerUtxoStorageService;

    @GET
    @Path("/limit/{address}/{limit}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "addresslimitUTXO")
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = AccountUtxoDto.class)
    })
    public RpcClientResult getUtxoByAddressAndLimit(

```

```

    @ApiParam(name="address", value="", required = true) @PathParam("address") String
address,
    @ApiParam(name="limit", value="", required = true) @PathParam("limit") Integer limit) {
if (StringUtils.isBlank(address) || limit == null) {
    return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
}
if (!AddressTool.validAddress(address)) {
    return Result.getFailed(LedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
}
Result result = null;
try {
    List<Coin> coinList = getAllUtxoByAddress(address);
    int limitValue = limit.intValue();
    boolean isLoadAll = (limitValue == 0);
    AccountUtxoDto accountUtxoDto = new AccountUtxoDto();
    List<UtxoDto> list = new LinkedList<>();
    int i = 0;
    for (Coin coin : coinList) {
        if (!coin.usable()) {
            continue;
        }
        if (coin.getNa().equals(Na.ZERO)) {
            continue;
        }
        if(!isLoadAll) {
            if(i >= limitValue) {
                break;
            }
            i++;
        }
        list.add(new UtxoDto(coin));
    }
    accountUtxoDto.setUtxoDtoList(list);
    result = Result.getSuccess().setData(accountUtxoDto);
    return result.toRpcClientResult();
} catch (Exception e) {
    Log.error(e);
    result = Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION);
    return result.toRpcClientResult();
}
}

```

```

@GET
@Path("/amount/{address}/{amount}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "addressamountUTXO")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = AccountUtxoDto.class)
})
public RpcClientResult getUtxoByAddressAndAmount(
    @ApiParam(name="address", value="", required = true) @PathParam("address") String
address,
    @ApiParam(name="amount", value="", required = true) @PathParam("amount") Long
amount) {
    if (StringUtils.isBlank(address) || amount == null) {
        return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
    }
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(LedgerErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Result result = null;
    try {
        List<Coin> coinList = getAllUtxoByAddress(address);
        Na amountNa = Na.valueOf(amount.longValue());

        AccountUtxoDto accountUtxoDto = new AccountUtxoDto();
        List<UtxoDto> list = new LinkedList<>();
        Na values = Na.ZERO;
        for (Coin coin : coinList) {
            if (!coin.usable()) {
                continue;
            }
            if (coin.getNa().equals(Na.ZERO)) {
                continue;
            }
            list.add(new UtxoDto(coin));
            values = values.add(coin.getNa());

            if (values.isGreaterOrEquals(amountNa)) {
                break;
            }
        }
        accountUtxoDto.setUtxoDtoList(list);
        result = Result.getSuccess().setData(accountUtxoDto);
    }
}

```

```

        return result.toRpcClientResult();
    } catch (Exception e) {
        Log.error(e);
        result = Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION);
        return result.toRpcClientResult();
    }
}

private List<Coin> getAllUtxoByAddress(String address) {
    List<Coin> coinList = new ArrayList<>();
    byte[] addressBytes = AddressTool.getAddress(address);
    List<Entry<byte[], byte[]>> coinBytesList =
utxoLedgerUtxoStorageService.getAllUtxoEntryBytes();
    Coin coin;
    for (Entry<byte[], byte[]> coinEntryBytes : coinBytesList) {
        coin = new Coin();
        try {
            coin.parse(coinEntryBytes.getValue(), 0);
        } catch (NulsException e) {
            Log.info("parse coin form db error");
            continue;
        }
        //if (Arrays.equals(coin.(), addressBytes))
        if (Arrays.equals(coin.getAddress(), addressBytes))
        {
            coin.setOwner(coinEntryBytes.getKey());
            coinList.add(coin);
        }
    }
    Collections.sort(coinList, CoinComparator.getInstance());
    return coinList;
}

@GET
@Path("/info")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "")
@ApiResponses(value = {
    @ApiResponse(code = 200, message = "success", response = TokenInfoDto.class)
})
public RpcClientResult getInfo() throws NulsException {
    long height = NulsContext.getInstance().getBestHeight();

```

```

List<Entry<byte[], byte[]>> coinBytesList =
utxoLedgerUtxoStorageService.getAllUtxoEntryBytes();
double totalNuls = 0d;
double lockedNuls = 0d;
Map<String, Holder> map = new HashMap<>();
Coin coin = new Coin();
int index = 0;
for (Entry<byte[], byte[]> coinEntryBytes : coinBytesList) {
    coin.parse(coinEntryBytes.getValue(), 0);
    double value = coin.getNa().toDouble();
    String address = AddressTool.getStringAddressByBytes(coin.getOwner());
    Holder holder = map.get(address);
    if (null == holder) {
        holder = new Holder();
        holder.setAddress(address);
        map.put(address, holder);
    }
    holder.addTotal(value);
    totalNuls = DoubleUtils.sum(totalNuls, value);
    if (coin.getLockTime() == -1 || coin.getLockTime() >
System.currentTimeMillis()||(coin.getLockTime())<1531152000000L&&coin.getLockTime()>height))
{
        holder.addLocked(value);
        lockedNuls = DoubleUtils.sum(lockedNuls, value);
    }
    System.out.println(index++);
}
Result<TokenInfoDto> result = Result.getSuccess();
TokenInfoDto info = new TokenInfoDto();
info.setTotalNuls(DoubleUtils.getRoundStr(totalNuls, 8, true));
info.setLockedNuls(DoubleUtils.getRoundStr(lockedNuls, 8, true));
List<Holder> holderList = new ArrayList<>(map.values());
Collections.sort(holderList);
List<HolderDto> dtoList = new ArrayList<>();
for (Holder holder : holderList) {
    HolderDto dto = new HolderDto(holder);
    dtoList.add(dto);
}
info.setAddressList(dtoList);
result.setData(info);
return result.toRpcClientResult();
}

```

```
}

196:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-
storage\src\main\java\io\nuls\ledger\storage\constant\LedgerStorageConstant.java
*/
```

```
package io.nuls.ledger.storage.constant;
```

```
/**
 * @desription:
 * @author: PierreLuo
 */
public interface LedgerStorageConstant {

    String DB_NAME_LEDGER_TX = "ledger_tx";
    String DB_NAME_LEDGER_UTXO = "ledger_utxo";

}
```

```
197:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-
storage\src\main\java\io\nuls\ledger\storage\service\impl\UtxoLedgerTransactionStorageServiceIm
pl.java
package io.nuls.ledger.storage.service.impl;
```

```
import io.nuls.core.tools.log.Log;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.*;
import io.nuls.ledger.storage.constant.LedgerStorageConstant;
import io.nuls.ledger.storage.service.UtxoLedgerTransactionStorageService;
```

```
import java.io.IOException;
```

```
/**
 * @desription:
```



```

* @author: PierreLuo
*/
@Service
public class UtxoLedgerTransactionStorageServiceImpl implements
UtxoLedgerTransactionStorageService, InitializingBean {

    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    /**
     *
     * This method is invoked after all properties are set, and is used to assist object initialization.
     */
    @Override
    public void afterPropertiesSet() throws NulsException {
        Result result = dbService.createArea(LedgerStorageConstant.DB_NAME_LEDGER_TX);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveTx(Transaction tx) {
        if (tx == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        byte[] txHashBytes = new byte[0];
        try {
            txHashBytes = tx.getHash().serialize();
        } catch (IOException e) {
            Log.error(e);
            return Result.getFailed(KernelErrorCode.IO_ERROR);
        }
        //
        Result result = dbService.putModel(LedgerStorageConstant.DB_NAME_LEDGER_TX,
txHashBytes, tx);
        return result;
    }
}

```

```

@Override
public Transaction getTx(NulsDigestData hash) {
    if (hash == null) {
        return null;
    }
    byte[] hashBytes = new byte[0];
    try {
        hashBytes = hash.serialize();
    } catch (IOException e) {
        Log.error(e);
        throw new NulsRuntimeException(e);
    }
    Transaction tx = dbService.getModel(LedgerStorageConstant.DB_NAME_LEDGER_TX,
hashBytes, Transaction.class);
    if (tx != null) {
        tx.setHash(hash);
    }
    return tx;
}

```

```

@Override
public Result deleteTx(Transaction tx) {
    if (tx == null) {
        return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
    }
    byte[] txHashBytes = new byte[0];
    try {
        txHashBytes = tx.getHash().serialize();
    } catch (IOException e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    }
    //
    Result result = dbService.delete(LedgerStorageConstant.DB_NAME_LEDGER_TX,
txHashBytes);
    return result;
}

```

```

@Override
public byte[] getTxBytes(byte[] txBytes) {
    if (txBytes == null) {

```

```

        return null;
    }
    return dbService.get(LedgerStorageConstant.DB_NAME_LEDGER_TX, txBytes);
}
}

```

198:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-storage\src\main\java\io\nuls\ledger\storage\service\impl\UtxoLedgerUtxoStorageServiceImpl.java  
package io.nuls.ledger.storage.service.impl;

```

import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.model.Entry;
import io.nuls.db.service.BatchOperation;
import io.nuls.db.service.DBService;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Result;
import io.nuls.ledger.storage.constant.LedgerStorageConstant;
import io.nuls.ledger.storage.service.UtxoLedgerUtxoStorageService;

```

```

import java.io.IOException;
import java.util.List;

```

```

/**
 * @desription:
 * @author: PierreLuo
 */
@Service
public class UtxoLedgerUtxoStorageServiceImpl implements UtxoLedgerUtxoStorageService,
InitializingBean {

    /**
     *
     * Universal data storage services.
     */

```

```

@Autowired
private DBService dbService;

/**
 *
 * This method is invoked after all properties are set, and is used to assist object initialization.
 */
@Override
public void afterPropertiesSet() throws NulsException {
    Result result = dbService.createArea(LedgerStorageConstant.DB_NAME_LEDGER_UTXO);
    if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
        throw new NulsRuntimeException(result.getErrorCode());
    }
}

@Override
public BatchOperation createWriteBatch() {
    return dbService.createWriteBatch(LedgerStorageConstant.DB_NAME_LEDGER_UTXO);
}

@Override
public Result saveUtxo(byte[] owner, Coin coin) {
    try {
        Log.info("save utxo::" + Hex.encode(owner));
        return dbService.put(LedgerStorageConstant.DB_NAME_LEDGER_UTXO, owner,
coin.serialize());
    } catch (IOException e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.IO_ERROR);
    }
}

@Override
public Coin getUtxo(byte[] owner) {
    byte[] utxoBytes = getUtxoBytes(owner);
    Coin coin = null;
    try {
        if (utxoBytes != null) {
            coin = new Coin();
            coin.parse(utxoBytes, 0);
        }
    } catch (NulsException e) {

```

```

        Log.error(e);
        return null;
    }
    return coin;
}

@Override
public Result deleteUtxo(byte[] owner) {
    return dbService.delete(LedgerStorageConstant.DB_NAME_LEDGER_UTXO, owner);
}

@Override
public byte[] getUtxoBytes(byte[] owner) {
    if (owner == null) {
        return null;
    }
    return dbService.get(LedgerStorageConstant.DB_NAME_LEDGER_UTXO, owner);
}

@Override
public List<byte[]> getAllUtxoBytes() {
    return dbService.valueList(LedgerStorageConstant.DB_NAME_LEDGER_UTXO);
}

@Override
public List<Entry<byte[], byte[]>> getAllUtxoEntryBytes() {
    return dbService.entryList(LedgerStorageConstant.DB_NAME_LEDGER_UTXO);
}
}

```

199:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-storage\src\main\java\io\nuls\ledger\storage\service\UtxoLedgerTransactionStorageService.java  
package io.nuls.ledger.storage.service;

```

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;

```

```

/**
 * @desription:
 * @author: PierreLuo

```

```

*/
public interface UtxoLedgerTransactionStorageService {

    Result saveTx(Transaction tx);

    Transaction getTx(NulsDigestData hash);

    Result deleteTx(Transaction tx);

    byte[] getTxBytes(byte[] txBytes);
}

```

200:F:\git\coin\nuls\nuls-1.1.3\nuls\ledger-module\utxo\ledger-utxo-storage\src\main\java\io\nuls\ledger\storage\service\UtxoLedgerUtxoStorageService.java

package io.nuls.ledger.storage.service;

```

import io.nuls.db.model.Entry;
import io.nuls.db.service.BatchOperation;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Result;

```

```

import java.util.List;

```

```

/**
 * @desription:
 * @author: PierreLuo
 */
public interface UtxoLedgerUtxoStorageService {

    BatchOperation createWriteBatch();

    Result saveUtxo(byte[] owner, Coin coin);

    Coin getUtxo(byte[] owner);

    Result deleteUtxo(byte[] owner);

    byte[] getUtxoBytes(byte[] owner);

    List<byte[]> getAllUtxoBytes();

    List<Entry<byte[], byte[]>> getAllUtxoEntryBytes();

```

