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

0:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-base\src\main\java\io\nuls\contract\module\impl\ContractModuleBootstrap.java

```
package io.nuls.contract.module.impl;


import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.contract.entity.tx.CallContractTransaction;
import io.nuls.contract.entity.tx.processor.CreateContractTxProcessor;
import io.nuls.contract.ledger.manager.ContractBalanceManager;
import io.nuls.contract.module.AbstractContractModule;
import io.nuls.contract.util.VMContext;
import io.nuls.contract.vm.program.ProgramMethod;
import io.nuls.core.tools.io.StringFileLoader;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.utils.TransactionManager;

import java.util.Map;

/**
 * @Desription:
 * @Author: PierreLuo
 * @Date: 2018/4/20
 */
public class ContractModuleBootstrap extends AbstractContractModule {

    private final static String NRC20_STANDARD_FILE = "contract/nrc20.json";
    /**
     * execute when the project starts.
     */
    @Override
    public void init() {
        Log.debug("contract init");
        initERC20Standard();
    }

    private void initERC20Standard() {
        String json = null;
        try {
```

```
            json = StringFileLoader.read(NRC20_STANDARD_FILE);
        } catch (Exception e) {
            // skip it
            Log.error("init NRC20Standard error.", e);
        }
        if(json == null) {
            return;
        }

        Map<String, ProgramMethod> jsonMap = null;
        try {
            jsonMap = JSONUtils.json2map(json, ProgramMethod.class);
        } catch (Exception e) {
            Log.error("init NRC20Standard map error.", e);
        }
        VMContext.setNrc20Methods(jsonMap);
    }

    /**
     * execute when the project starts.
     */
    @Override
    public void start() {
        Log.debug("contract start");
        this.waitForDependencyRunning(ConsensusConstant.MODULE_ID_CONSENSUS);
        ContractBalanceManager balanceManager =
NulsContext.getServiceBean(ContractBalanceManager.class);
        balanceManager.initContractBalance();
        balanceManager.initAllTokensForAllAccounts();
    }

    @Override
    public void shutdown() {
        //TODO do something or not
    }

    @Override
    public void destroy() {
        //TODO do something or not
    }

    @Override
```

```
    public String getInfo() {
        return "contract module is " + this.getStatus();
    }
}

1:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
base\src\main\java\io\nuls\contract\service\impl\ContractServiceImpl.java
 */
package io.nuls.contract.service.impl;

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.model.CoinDataResult;
import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.account.service.AccountService;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTokenInfo;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.dto.ContractTransfer;
import io.nuls.contract.entity.tx.*;
import io.nuls.contract.entity.txdata.*;
import io.nuls.contract.helper.VMHelper;
import io.nuls.contract.ledger.manager.ContractBalanceManager;
import io.nuls.contract.ledger.service.ContractTransactionInfoService;
import io.nuls.contract.ledger.service.ContractUtxoService;
import io.nuls.contract.ledger.util.ContractLedgerUtil;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.po.TransactionInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.contract.storage.service.ContractExecuteResultStorageService;
import io.nuls.contract.storage.service.ContractTokenTransferStorageService;
import io.nuls.contract.storage.service.ContractTransferTransactionStorageService;
import io.nuls.contract.util.ContractCoinComparator;
import io.nuls.contract.util.ContractUtil;
import io.nuls.contract.util.VMContext;
import io.nuls.contract.vm.program.*;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.map.MapUtil;
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.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.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.VarInt;
import io.nuls.ledger.util.LedgerUtil;
import io.nuls.protocol.constant.ProtocolConstant;

import java.io.IOException;
import java.math.BigInteger;
import java.util.*;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

import static io.nuls.ledger.util.LedgerUtil.asBytes;
import static io.nuls.ledger.util.LedgerUtil.asString;

/**
 * @Desription:
 * @Author: PierreLuo
 * @Date:
 */
@Service
public class ContractServiceImpl implements ContractService, InitializingBean {

    @Autowired
    private ContractTransactionInfoService contractTransactionInfoService;

    @Autowired
    private ContractUtxoService contractUtxoService;

    @Autowired
    private ContractTransferTransactionStorageService
contractTransferTransactionStorageService;
```

```
    @Autowired
    private ContractAddressStorageService contractAddressStorageService;

    @Autowired
    private ContractExecuteResultStorageService contractExecuteResultStorageService;

    @Autowired
    private ContractBalanceManager contractBalanceManager;

    @Autowired
    private ContractTokenTransferStorageService contractTokenTransferStorageService;

    @Autowired
    private AccountService accountService;

    @Autowired
    private VMHelper vmHelper;

    @Autowired
    private VMContext vmContext;

    private ProgramExecutor programExecutor;

    private Lock lock = new ReentrantLock();

    private ThreadLocal<ProgramExecutor> localProgramExecutor = new ThreadLocal<>();

    @Override
    public void afterPropertiesSet() throws NulsException {
        programExecutor = vmHelper.getProgramExecutor();
    }

    /**
     *
     * @param executor
     * @param number
     * @param prevStateRoot
     * @param create
     * @return
     */
    private Result<ContractResult> createContract(ProgramExecutor executor, long number, byte[]
prevStateRoot, CreateContractData create) {
```

```
if(number < 0) {
    return Result.getFailed(ContractErrorCode.PARAMETER_ERROR);
}

// stateRoot
if(executor == null && prevStateRoot == null) {
    return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
}
try {
    byte[] contractAddress = create.getContractAddress();
    byte[] sender = create.getSender();
    long price = create.getPrice();
    ProgramCreate programCreate = new ProgramCreate();
    programCreate.setContractAddress(contractAddress);
    programCreate.setSender(sender);
    programCreate.setValue(BigInteger.ZERO);
    programCreate.setPrice(price);
    programCreate.setGasLimit(create.getGasLimit());
    programCreate.setNumber(number);
    programCreate.setContractCode(create.getCode());
    programCreate.setArgs(create.getArgs());

    ProgramExecutor track;
    if(executor == null) {
        track = programExecutor.begin(prevStateRoot);
    } else {
        track = executor.startTracking();
    }
    ProgramResult programResult = track.create(programCreate);
    // track
    if(executor == null) {
        track.commit();
    }

    // current state root
    byte[] stateRoot = executor == null ? track.getRoot() : null;
    ContractResult contractResult = new ContractResult();
    contractResult.setNonce(programResult.getNonce());
    contractResult.setGasUsed(programResult.getGasUsed());
    contractResult.setPrice(price);
    contractResult.setStateRoot(stateRoot);
    contractResult.setBalance(programResult.getBalance());
```

```
            contractResult.setContractAddress(contractAddress);
            contractResult.setSender(sender);
            contractResult.setRemark(ContractConstant.CREATE);
            // track
            contractResult.setTxTrack(track);

            if(!programResult.isSuccess()) {
                Result<ContractResult> result =
Result.getFailed(ContractErrorCode.CONTRACT_EXECUTE_ERROR);
                contractResult.setError(programResult.isError());
                contractResult.setRevert(programResult.isRevert());
                contractResult.setErrorMessage(programResult.getErrorMessage());
                contractResult.setStackTrace(programResult.getStackTrace());
                result.setMsg(programResult.getErrorMessage());
                result.setData(contractResult);
                return result;
            }

            // gas()
            contractResult.setError(false);
            contractResult.setRevert(false);
            contractResult.setEvents(programResult.getEvents());
            contractResult.setTransfers(generateContractTransfer(programResult.getTransfers()));

            Result<ContractResult> result = Result.getSuccess();
            result.setData(contractResult);
            return result;
        } catch (Exception e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_EXECUTE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        }
    }

    private List<ContractTransfer> generateContractTransfer(List<ProgramTransfer> transfers) {
        if(transfers == null || transfers.size() == 0) {
            return new ArrayList<>(0);
        }
        List<ContractTransfer> resultList = new ArrayList<>(transfers.size());
        ContractTransfer contractTransfer;
        for(ProgramTransfer transfer : transfers) {
```

```
            contractTransfer = new ContractTransfer();
            contractTransfer.setFrom(transfer.getFrom());
            contractTransfer.setTo(transfer.getTo());
            contractTransfer.setValue(Na.valueOf(transfer.getValue().longValue()));
            contractTransfer.setFee(Na.ZERO);
            resultList.add(contractTransfer);
        }
        return resultList;
    }

    /**
     *
     * @param executor
     * @param number
     * @param prevStateRoot
     * @param call
     * @return
     */
    private Result<ContractResult> callContract(ProgramExecutor executor, long number, byte[]
prevStateRoot, CallContractData call) {
        if(number < 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR);
        }
        if(executor == null && prevStateRoot == null) {
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
        }
        try {
            byte[] contractAddress = call.getContractAddress();
            byte[] sender = call.getSender();
            long price = call.getPrice();
            ProgramCall programCall = new ProgramCall();
            programCall.setContractAddress(contractAddress);
            programCall.setSender(sender);
            programCall.setValue(BigInteger.valueOf(call.getValue()));
            programCall.setPrice(price);
            programCall.setGasLimit(call.getGasLimit());
            programCall.setNumber(number);
            programCall.setMethodName(call.getMethodName());
            programCall.setMethodDesc(call.getMethodDesc());
            programCall.setArgs(call.getArgs());

            ProgramExecutor track;
```

```
        if(executor == null) {
            track = programExecutor.begin(prevStateRoot);
        } else {
            track = executor.startTracking();
        }

        ProgramResult programResult = track.call(programCall);

        // track
        if(executor == null) {
            track.commit();
        }

        // current state root
        byte[] stateRoot = executor == null ? track.getRoot() : null;
        ContractResult contractResult = new ContractResult();

        contractResult.setNonce(programResult.getNonce());
        contractResult.setGasUsed(programResult.getGasUsed());
        contractResult.setPrice(price);
        contractResult.setStateRoot(stateRoot);
        contractResult.setBalance(programResult.getBalance());
        contractResult.setContractAddress(contractAddress);
        contractResult.setSender(sender);
        contractResult.setValue(programCall.getValue().longValue());
        contractResult.setRemark(ContractConstant.CALL);
        // track
        contractResult.setTxTrack(track);

        if(!programResult.isSuccess()) {
            Result<ContractResult> result =
Result.getFailed(ContractErrorCode.CONTRACT_EXECUTE_ERROR);
            contractResult.setError(programResult.isError());
            contractResult.setRevert(programResult.isRevert());
            contractResult.setErrorMessage(programResult.getErrorMessage());
            contractResult.setStackTrace(programResult.getStackTrace());
            result.setMsg(programResult.getErrorMessage());
            result.setData(contractResult);
            return result;
        }

        // Gas()
```

```
            contractResult.setError(false);
            contractResult.setRevert(false);
            contractResult.setResult(programResult.getResult());
            contractResult.setEvents(programResult.getEvents());
            contractResult.setTransfers(generateContractTransfer(programResult.getTransfers()));

            Result<ContractResult> result = Result.getSuccess();
            result.setData(contractResult);
            return result;
        } catch (Exception e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_EXECUTE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        }
    }

    /**
     *
     * @param executor
     * @param number
     * @param prevStateRoot
     * @param delete
     * @return
     */
    private Result<ContractResult> deleteContract(ProgramExecutor executor, long number, byte[]
prevStateRoot, DeleteContractData delete) {
        if(number < 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR);
        }
        if(executor == null && prevStateRoot == null) {
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
        }
        try {
            byte[] contractAddress = delete.getContractAddress();
            byte[] sender = delete.getSender();
            ProgramExecutor track;

            if(executor == null) {
                track = programExecutor.begin(prevStateRoot);
            } else {
                track = executor.startTracking();
```

```
        }
        ProgramResult programResult = track.stop(contractAddress, sender);
        // track
        if(executor == null) {
            track.commit();
        }

        // current state root
        byte[] stateRoot = executor == null ? track.getRoot() : null;
        ContractResult contractResult = new ContractResult();
        contractResult.setNonce(programResult.getNonce());
        contractResult.setGasUsed(programResult.getGasUsed());
        contractResult.setStateRoot(stateRoot);
        contractResult.setBalance(programResult.getBalance());
        contractResult.setContractAddress(contractAddress);
        contractResult.setSender(sender);
        contractResult.setRemark(ContractConstant.DELETE);
        // track
        contractResult.setTxTrack(track);

        if(!programResult.isSuccess()) {
            Result<ContractResult> result =
Result.getFailed(ContractErrorCode.CONTRACT_EXECUTE_ERROR);
            contractResult.setError(programResult.isError());
            contractResult.setRevert(programResult.isRevert());
            contractResult.setErrorMessage(programResult.getErrorMessage());
            contractResult.setStackTrace(programResult.getStackTrace());
            result.setMsg(programResult.getErrorMessage());
            result.setData(contractResult);
            return result;
        }

        //
        contractResult.setError(false);
        contractResult.setRevert(false);

        Result<ContractResult> result = Result.getSuccess();
        result.setData(contractResult);
        return result;
    } catch (Exception e) {
        Log.error(e);
        Result result = Result.getFailed(ContractErrorCode.CONTRACT_EXECUTE_ERROR);
```

```
            result.setMsg(e.getMessage());
            return result;
        }
    }

    @Override
    public boolean isContractAddress(byte[] addressBytes) {
        return ContractLedgerUtil.isExistContractAddress(addressBytes);
    }

    private Result<Integer> saveConfirmedTransaction(Transaction tx) {
        if (tx == null) {
            Log.error("save confirmed contract tx error, tx is null.");
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
        }

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

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

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

        Result result = contractTransactionInfoService.saveTransactionInfo(txInfoPo, addresses);
        if (result.isFailed()) {
            Log.error("save confirmed contract transactionInfo error, reason is {}.", result.getMsg());
            return result;
        }

        // utxo
        if(tx.getType() != ContractConstant.TX_TYPE_CALL_CONTRACT) {
            result = contractUtxoService.saveUtxoForContractAddress(tx);
            if (result.isFailed()) {
                Log.error("save confirmed non-call-contract transfer utxo error, reason is {}.",
result.getMsg());
                return result;
            }
        }
```

```
        result.setData(new Integer(1));
        return result;
    }

    @Override
    public Result saveContractTransferTx(ContractTransferTransaction tx) {
        Result result = contractUtxoService.saveUtxoForContractAddress(tx);
        if (result.isFailed()) {
            Log.error("save contract transfer utxo error, reason is {}.", result.getMsg());
            return result;
        }

        result = contractTransferTransactionStorageService.saveContractTransferTx(tx.getHash(),
tx);
        if (result.isFailed()) {
            Log.error("save contract transfer tx error, reason is {}.", result.getMsg());
            contractUtxoService.deleteUtxoOfTransaction(tx);
            return result;
        }
        return result;
    }

    @Override
    public Result rollbackContractTransferTx(ContractTransferTransaction tx) {
        Result result =
contractTransferTransactionStorageService.deleteContractTransferTx(tx.getHash());
        if (result.isFailed()) {
            Log.error("rollback contract transfer tx error, reason is {}.", result.getMsg());
            return result;
        }
        result = contractUtxoService.deleteUtxoOfTransaction(tx);
        if (result.isFailed()) {
            contractTransferTransactionStorageService.saveContractTransferTx(tx.getHash(), tx);
            Log.error("rollback contract transfer utxo error, reason is {}.", result.getMsg());
            return result;
        }
        return result;
    }

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

```
    List<Transaction> savedTxList = new ArrayList<>();
    Result result;
    for (int i = 0; i < txs.size(); i++) {
        result = saveConfirmedTransaction(txs.get(i));
        if (result.isSuccess()) {
            if(result.getData() != null && (int) result.getData() == 1) {
                savedTxList.add(txs.get(i));
            }
        } else {
            rollbackTransactionList(savedTxList);
            return result;
        }
    }
    return Result.getSuccess().setData(savedTxList.size());
}

@Override
public Result<Integer> rollbackTransactionList(List<Transaction> txs) {
    //
    for (int i = txs.size() - 1; i >= 0; i--) {
        rollbackTransaction(txs.get(i));
    }
    return Result.getSuccess().setData(new Integer(txs.size()));
}

private Result<Integer> rollbackTransaction(Transaction tx) {

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

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

    TransactionInfoPo txInfoPo = new TransactionInfoPo(tx);
    //  - TransactionInfo
    Result result = contractTransactionInfoService.deleteTransactionInfo(txInfoPo, addresses);

    if (result.isFailed()) {
        return result;
    }
```

```
    // utxo
    if(tx.getType() != ContractConstant.TX_TYPE_CALL_CONTRACT) {
        result = contractUtxoService.deleteUtxoOfTransaction(tx);
        if (result.isFailed()) {
            Log.error("rollback non-call-contract transfer utxo error, reason is {}.", result.getMsg());
            return result;
        }
    }

    return result;
}

@Override
public Result saveContractExecuteResult(NulsDigestData hash, ContractResult result) {
    if (hash == null || result == null) {
        return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
    }
    vmHelper.updateLastedPriceForAccount(result.getSender(), result.getPrice());
    return contractExecuteResultStorageService.saveContractExecuteResult(hash, result);
}

@Override
public Result deleteContractExecuteResult(NulsDigestData hash) {
    if (hash == null) {
        return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
    }
    return contractExecuteResultStorageService.deleteContractExecuteResult(hash);
}

@Override
public ContractResult getContractExecuteResult(NulsDigestData hash) {
    if (hash == null) {
        return null;
    }
    return contractExecuteResultStorageService.getContractExecuteResult(hash);
}

/**
 * @param from
 * @param to
 * @param values
 * @param fee
```

```
    * @param isSendBack
    * @param orginHash
    * @param blockTime
    * @param toMaps
    * @param contractUsedCoinMap toMaps,  contractUsedCoinMap UTXO
    *                          (()UTXOUTXO
    *                            ()[DB][toMaps]utxo
    *                             contractUsedCoinMap UTXO)
    * @param bestHeight
    * @return
    */
   private Result<ContractTransferTransaction> transfer(byte[] from, byte[] to, Na values, Na fee,
boolean isSendBack, NulsDigestData orginHash, long blockTime,
                                                        Map<String, Coin> toMaps,
                                                        Map<String, Coin> contractUsedCoinMap, Long bestHeight) {
       try {
          if(!ContractLedgerUtil.isExistContractAddress(from)) {
             return Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST);
          }

          ContractTransferTransaction tx = new ContractTransferTransaction();
          tx.setTime(blockTime);

          CoinData coinData = new CoinData();
          List<Coin> tos = coinData.getTo();
          Coin toCoin = new Coin(to, values.subtract(fee));
          tos.add(toCoin);

          // toMapscontractUsedCoinMapUTXO
          // toMaps,  contractUsedCoinMap UTXO
          CoinDataResult coinDataResult = getContractSpecialTransferCoinData(from, values,
toMaps, contractUsedCoinMap, bestHeight);
          if (!coinDataResult.isEnough()) {
             return Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE);
          }
          coinData.setFrom(coinDataResult.getCoinList());
          if (coinDataResult.getChange() != null) {
             tos.add(coinDataResult.getChange());
          }

          tx.setCoinData(coinData);
          byte successByte;
```

```
            byte[] remark = null;
            if(isSendBack) {
                successByte = 0;
                remark =
ContractConstant.SEND_BACK_REMARK.getBytes(NulsConfig.DEFAULT_ENCODING);
            } else {
                successByte = 1;
            }
            tx.setRemark(remark);
            tx.setTxData(new ContractTransferData(orginHash, from, successByte));
            tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));

            // UTXO
            byte[] txBytes = tx.getHash().serialize();
            for (int i = 0, size = tos.size(); i < size; i++) {
                if (toMaps != null) {
                    toMaps.put(LedgerUtil.asString(ArraysTool.concatenate(txBytes, new
VarInt(i).encode())), tos.get(i));
                }
            }

            // ()
            return Result.getSuccess().setData(tx);
        } catch (IOException e) {
            e.printStackTrace();
        }

        return null;
    }

    /**
     * @param address
     * @param amount
     * @param bestHeight
     * @return
     * @throws NulsException
     */
    public CoinDataResult getContractSpecialTransferCoinData(byte[] address, Na amount,
Map<String, Coin> toMaps, Map<String, Coin> contractUsedCoinMap, Long bestHeight) {
        lock.lock();
        try {
            CoinDataResult coinDataResult = new CoinDataResult();
```

```
        List<Coin> coinList = contractBalanceManager.getCoinListByAddress(address);
        //pierre add contract coin key
        Set<Map.Entry<String, Coin>> toMapsEntries = toMaps.entrySet();
        Coin toCoin;
        Coin cloneCoin;
        String key;
        for(Map.Entry<String, Coin> toMapsEntry : toMapsEntries) {
            key = toMapsEntry.getKey();
            toCoin = toMapsEntry.getValue();
            if (Arrays.equals(toCoin.getAddress(), address) &&
!contractUsedCoinMap.containsKey(key)) {
                cloneCoin = new Coin(asBytes(key), toCoin.getNa(), toCoin.getLockTime());
                cloneCoin.setFrom(toCoin);
                cloneCoin.setKey(key);
                coinList.add(cloneCoin);
            }
        }

        if (coinList.isEmpty()) {
            coinDataResult.setEnough(false);
            return coinDataResult;
        }

        //
        Collections.sort(coinList, ContractCoinComparator.getInstance());

        boolean enough = false;
        List<Coin> coins = new ArrayList<>();
        Na values = Na.ZERO;
        Coin coin;
        //
        for (int i = 0, length = coinList.size(); i < length; i++) {
            coin = coinList.get(i);
            if(bestHeight != null) {
                if (!coin.usable(bestHeight)) {
                    continue;
                }
            } else {
                if (!coin.usable()) {
                    continue;
                }
            }
```

```
            if (coin.getNa().equals(Na.ZERO)) {
                continue;
            }
            // for contract, UTXO
            if(StringUtils.isNotBlank(coin.getKey())) {
                if(contractUsedCoinMap.containsKey(coin.getKey())) {
                    continue;
                }
                contractUsedCoinMap.put(coin.getKey(), coin);
            }
            coins.add(coin);

            //
            values = values.add(coin.getNa());
            if (values.isGreaterOrEquals(amount)) {
                //
                Na change = values.subtract(amount);
                if (change.isGreaterThan(Na.ZERO)) {
                    Coin changeCoin = new Coin();
                    changeCoin.setOwner(address);
                    changeCoin.setNa(change);
                    coinDataResult.setChange(changeCoin);
                }
                enough = true;
                coinDataResult.setEnough(true);
                coinDataResult.setFee(Na.ZERO);
                coinDataResult.setCoinList(coins);
                break;
            }
        }
        if (!enough) {
            coinDataResult.setEnough(false);
            return coinDataResult;
        }
        return coinDataResult;
    } finally {
        lock.unlock();
    }
}

    private Result<ContractResult> invokeContract(ProgramExecutor track, Transaction tx, long
height, byte[] stateRoot, boolean isForkChain) {
```

```
        if(tx == null || height < 0) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR);
        }
        int txType = tx.getType();

        //
        if(!isForkChain) {
            ContractTransaction contractTx = (ContractTransaction) tx;
            //
            ContractResult contractExecutedResult;
            contractExecutedResult = contractTx.getContractResult();
            if(contractExecutedResult == null) {
                contractExecutedResult = getContractExecuteResult(tx.getHash());
                if(contractExecutedResult != null) {
                    if(Log.isDebugEnabled()) {
                        Log.debug("===get ContractResult from db.");
                    }
                }
            } else {
                if(Log.isDebugEnabled()) {
                    Log.debug("===get ContractResult from tx object.");
                }
            }
            if(contractExecutedResult != null) {
                contractExecutedResult.setTxTrack(track);
                if(contractExecutedResult.isSuccess()) {
                    //
                    if(contractExecutedResult.isSuccess()) {
                        if(tx instanceof CallContractTransaction) {
                            this.refreshTempBalance((CallContractTransaction) tx, contractExecutedResult,
height);
                        }
                    } else {
                        // UTXO
                        contractExecutedResult.setValue(((ContractData) tx.getTxData()).getValue());
                    }
                    return Result.getSuccess().setData(contractExecutedResult);
                } else {
                    Log.info("contractExecutedResult failed. {}", contractExecutedResult.toString());
                    return Result.getFailed().setData(contractExecutedResult);
                }
            }
```

```
        }

        if (txType == ContractConstant.TX_TYPE_CREATE_CONTRACT) {
            CreateContractTransaction createContractTransaction = (CreateContractTransaction) tx;
            CreateContractData createContractData = createContractTransaction.getTxData();
            if(!ContractUtil.checkPrice(createContractData.getPrice())) {
                return Result.getFailed(ContractErrorCode.CONTRACT_MINIMUM_PRICE);
            }
            Result<ContractResult> result = createContract(track, height, stateRoot,
createContractData);
            ContractResult contractResult = result.getData();
            if (contractResult != null && contractResult.isSuccess()) {
                Result nrc20Result = vmHelper.validateNrc20Contract((ProgramExecutor)
contractResult.getTxTrack(), createContractTransaction, contractResult);
                if(nrc20Result.isFailed()) {
                    contractResult.setError(true);
if(ContractErrorCode.CONTRACT_NRC20_SYMBOL_FORMAT_INCORRECT.equals(nrc20Resul
t.getErrorCode())) {
                        contractResult.setErrorMessage("The format of the symbol is incorrect.");
                    } else
if(ContractErrorCode.CONTRACT_NAME_FORMAT_INCORRECT.equals(nrc20Result.getErrorC
ode())) {
                        contractResult.setErrorMessage("The format of the name is incorrect.");
                    } else
if(ContractErrorCode.CONTRACT_NRC20_MAXIMUM_DECIMALS.equals(nrc20Result.getErrorC
ode())) {
                        contractResult.setErrorMessage("The value of decimals ranges from 0 to 18.");
                    } else
if(ContractErrorCode.CONTRACT_NRC20_MAXIMUM_TOTAL_SUPPLY.equals(nrc20Result.get
ErrorCode())) {
                        contractResult.setErrorMessage("The value of totalSupply ranges from 1 to 2^256 -
1.");
                    } else {
                        contractResult.setErrorMessage("Unkown error.");
                    }
                    result.setData(contractResult);
                }
            }
            return result;
        } else if(txType == ContractConstant.TX_TYPE_CALL_CONTRACT) {
            CallContractTransaction callContractTransaction = (CallContractTransaction) tx;
            CallContractData callContractData = callContractTransaction.getTxData();
```

```
            if(!ContractUtil.checkPrice(callContractData.getPrice())) {
                return Result.getFailed(ContractErrorCode.CONTRACT_MINIMUM_PRICE);
            }
            Result<ContractResult> result = callContract(track, height, stateRoot, callContractData);
            byte[] contractAddress = callContractData.getContractAddress();
            BigInteger preBalance = vmContext.getBalance(contractAddress, height);
            ContractResult contractResult = result.getData();
            if(!contractResult.isSuccess()) {
                Log.info("contractResult failed. {}", contractResult.toString());
            }
            contractResult.setPreBalance(preBalance);
            //
            if(result.isSuccess()) {
                this.refreshTempBalance(callContractTransaction, contractResult, height);
            } else {
                // UTXO
                contractResult.setValue(callContractData.getValue());
            }
            return result;
        } else if(txType == ContractConstant.TX_TYPE_DELETE_CONTRACT) {
            DeleteContractTransaction deleteContractTransaction = (DeleteContractTransaction) tx;
            DeleteContractData deleteContractData = deleteContractTransaction.getTxData();
            Result<ContractResult> result = deleteContract(track, height, stateRoot,
deleteContractData);
            return result;
        } else {
            return Result.getSuccess();
        }
    }

    private void refreshTempBalance(CallContractTransaction callContractTransaction,
ContractResult contractExecutedResult, Long height) {
        CallContractData callContractData = callContractTransaction.getTxData();
        byte[] contractAddress = callContractData.getContractAddress();
        BigInteger preBalance = vmContext.getBalance(contractAddress, height);
        contractExecutedResult.setPreBalance(preBalance);
        //
        long value = callContractData.getValue();
        if(value > 0) {
            contractBalanceManager.addTempBalance(contractAddress, Na.valueOf(value));
        }
        // ,
```

```
    List<ContractTransfer> transfers = contractExecutedResult.getTransfers();
    if(transfers != null && transfers.size() > 0) {
        //Na outAmount = Na.ZERO;
        //Na inAmount = Na.ZERO;
        LinkedHashMap<String, Na>[] contracts = this.filterContractNa(transfers);
        LinkedHashMap<String, Na> contractOutNa = contracts[0];
        LinkedHashMap<String, Na> contractInNa = contracts[1];
        byte[] contractBytes;
        Set<Map.Entry<String, Na>> outs = contractOutNa.entrySet();
        for(Map.Entry<String, Na> out : outs) {
            contractBytes = asBytes(out.getKey());
            vmContext.getBalance(contractBytes, height);
            contractBalanceManager.minusTempBalance(contractBytes, out.getValue());
        }
        Set<Map.Entry<String, Na>> ins = contractInNa.entrySet();
        for(Map.Entry<String, Na> in : ins) {
            contractBytes = asBytes(in.getKey());
            vmContext.getBalance(contractBytes, height);
            contractBalanceManager.addTempBalance(contractBytes, in.getValue());
        }
        //contractBalanceManager.addTempBalance(contractAddress, inAmount.getValue());
        //contractBalanceManager.minusTempBalance(contractAddress, outAmount.getValue());
    }
}

private LinkedHashMap<String, Na>[] filterContractNa(List<ContractTransfer> transfers) {
    LinkedHashMap<String, Na> contractOutNa = MapUtil.createLinkedHashMap(4);
    LinkedHashMap<String, Na> contractInNa = MapUtil.createLinkedHashMap(4);
    LinkedHashMap<String, Na>[] contracts = new LinkedHashMap[2];
    contracts[0] = contractOutNa;
    contracts[1] = contractInNa;

    byte[] from,to;
    Na transferValue;
    for(ContractTransfer transfer : transfers) {
        from = transfer.getFrom();
        to = transfer.getTo();
        transferValue = transfer.getValue();
        if(ContractUtil.isLegalContractAddress(from)) {
            String contract = asString(from);
            Na na = contractOutNa.get(contract);
            if(na == null) {
```

```
                contractOutNa.put(contract, transferValue);
            } else {
                contractOutNa.put(contract, na.add(transferValue));
            }
        }
        if(ContractUtil.isLegalContractAddress(to)) {
            String contract = asString(to);
            Na na = contractInNa.get(contract);
            if(na == null) {
                contractInNa.put(contract, transferValue);
            } else {
                contractInNa.put(contract, na.add(transferValue));
            }
        }
        //if(ArraysTool.arrayEquals(transfer.getFrom(), contractAddress)) {
        //    outAmount = outAmount.add(transfer.getValue());
        //}
        //if(ArraysTool.arrayEquals(transfer.getTo(), contractAddress)) {
        //    inAmount = inAmount.add(transfer.getValue());
        //}
    }
    return contracts;
}



private void rollbackContractTempBalance(Transaction tx, ContractResult contractResult) {
    if(tx != null && tx.getType() == ContractConstant.TX_TYPE_CALL_CONTRACT) {
        CallContractTransaction callContractTransaction = (CallContractTransaction) tx;
        CallContractData callContractData = callContractTransaction.getTxData();
        byte[] contractAddress = callContractData.getContractAddress();
        // ,
        List<ContractTransfer> transfers = contractResult.getTransfers();
        if(transfers != null && transfers.size() > 0) {
            //Na outAmount = Na.ZERO;
            //Na inAmount = Na.ZERO;
            LinkedHashMap<String, Na>[] contracts = this.filterContractNa(transfers);
            LinkedHashMap<String, Na> contractOutNa = contracts[0];
            LinkedHashMap<String, Na> contractInNa = contracts[1];
            byte[] contractBytes;
            Set<Map.Entry<String, Na>> ins = contractInNa.entrySet();
            for(Map.Entry<String, Na> in : ins) {
```

```
                contractBytes = asBytes(in.getKey());
                contractBalanceManager.minusTempBalance(contractBytes, in.getValue());
            }
            Set<Map.Entry<String, Na>> outs = contractOutNa.entrySet();
            for(Map.Entry<String, Na> out : outs) {
                contractBytes = asBytes(out.getKey());
                contractBalanceManager.addTempBalance(contractBytes, out.getValue());
            }

            //contractBalanceManager.addTempBalance(contractAddress, outAmount.getValue());
            //contractBalanceManager.minusTempBalance(contractAddress, inAmount.getValue());
        }
        //
        long value = callContractData.getValue();
        if(value > 0) {
            contractBalanceManager.minusTempBalance(contractAddress, Na.valueOf(value));
        }
    }
}

@Override
public void createContractTempBalance() {
    contractBalanceManager.createTempBalanceMap();
}

@Override
public void removeContractTempBalance() {
    contractBalanceManager.removeTempBalanceMap();
}

private void rollbackContractTransferTxs(Map<String, ContractTransferTransaction>
successContractTransferTxs, Map<String, Coin> toMaps, Map<String, Coin>
contractUsedCoinMap) {
    if(successContractTransferTxs != null && successContractTransferTxs.size() > 0) {
        Collection<ContractTransferTransaction> values = successContractTransferTxs.values();
        for(Transaction tx : values) {
            rollbackToMapAndContractUsedCoinMap(tx, toMaps, contractUsedCoinMap);
        }
    }
}

private void rollbackToMapAndContractUsedCoinMap(Transaction tx, Map<String, Coin>
```

```
toMaps, Map<String,Coin> contractUsedCoinMap) {
        if (tx == null || tx.getCoinData() == null || contractUsedCoinMap == null) {
            return;
        }
        CoinData coinData = tx.getCoinData();
        List<Coin> froms = coinData.getFrom();
        List<Coin> tos = coinData.getTo();
        String key;
        for (Coin from : froms) {
            key = from.getKey();
            if(key != null) {
                contractUsedCoinMap.remove(from.getKey());
            }
        }
        try {
            byte[] txHashBytes = tx.getHash().serialize();
            for (int i = 0, size = tos.size(); i < size; i++) {
                toMaps.remove(LedgerUtil.asString(ArraysTool.concatenate(txHashBytes, new
VarInt(i).encode())));
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private Result<ContractTransferTransaction> createContractTransferTx(ContractTransfer
transfer, long blockTime, Map<String, Coin> toMaps, Map<String, Coin> contractUsedCoinMap,
Long bestHeight) {
        Result<ContractTransferTransaction> result;
        result = transfer(transfer.getFrom(), transfer.getTo(), transfer.getValue(), transfer.getFee(),
transfer.isSendBack(), transfer.getOrginHash(), blockTime, toMaps, contractUsedCoinMap,
bestHeight);
        if(result.isSuccess()) {
            result.getData().setTransfer(transfer);
        } else {
            Log.error("contract transfer failed. Reason: " + result);
        }
        return result;
    }

    private List<ContractTransfer> createReturnFundsContractTransfer(Transaction tx, Na
sendBack) {
```

```
        //
        Na transferFee = Na.ZERO;

        if(sendBack.compareTo(transferFee) <= 0) {
            transferFee = sendBack;
        }

        List<ContractTransfer> contractTransferList = new ArrayList<>();
        try {
            Set<String> addressFromTX = SignatureUtil.getAddressFromTX(tx);
            if(addressFromTX == null || addressFromTX.size() == 0) {
                return contractTransferList;
            }
            Object[] array = addressFromTX.toArray();
            String fromAddress = (String) array[0];
            byte[] fromAddressBytes = AddressTool.getAddress(fromAddress);
            CoinData coinData = tx.getCoinData();
            List<Coin> toList = coinData.getTo();
            HashMap<String, Na> sendBackMap = MapUtil.createHashMap(toList.size());
            String ownerStr;
            for(Coin coin : toList) {
                if(!ArraysTool.arrayEquals(fromAddressBytes, coin.getOwner())) {
                    ownerStr = AddressTool.getStringAddressByBytes(coin.getOwner());
                    Na addressNa = sendBackMap.get(ownerStr);
                    if(addressNa == null) {
                        sendBackMap.put(ownerStr, coin.getNa());
                    } else {
                        sendBackMap.put(ownerStr, addressNa.add(coin.getNa()));
                    }
                }
            }
            if(sendBackMap.size() > 0) {
                for(Map.Entry<String, Na> entry : sendBackMap.entrySet()) {
                    ContractTransfer transfer = new
ContractTransfer(AddressTool.getAddress(entry.getKey()), fromAddressBytes, entry.getValue(),
transferFee, true);
                    contractTransferList.add(transfer);
                }
            }

        } catch (NulsException e) {
            Log.error(e);
```

```
        }
        return contractTransferList;
    }

    @Override
    public Result<ContractTokenInfo> getContractTokenViaVm(String address, String
contractAddress) {
        try {
            if (StringUtils.isBlank(contractAddress) || StringUtils.isBlank(address)) {
                return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
            }

            if (!AddressTool.validAddress(contractAddress) || !AddressTool.validAddress(address)) {
                return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
            }

            byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
            Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
            ContractAddressInfoPo po = contractAddressInfoResult.getData();
            if(po == null) {
                return Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST);
            }
            if(!po.isNrc20()) {
                return Result.getFailed(ContractErrorCode.CONTRACT_NOT_NRC20);
            }

            ProgramResult programResult = vmHelper.invokeViewMethod(contractAddressBytes,
"balanceOf", null, address);
            Result<ContractTokenInfo> result;
            if(!programResult.isSuccess()) {
                result = Result.getFailed(ContractErrorCode.DATA_ERROR);
                result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
            } else {
                result = Result.getSuccess();
                ContractTokenInfo tokenInfo = new ContractTokenInfo(contractAddress,
po.getNrc20TokenName(), po.getDecimals(), new BigInteger(programResult.getResult()),
po.getNrc20TokenSymbol(), po.getBlockHeight());
                byte[] prevStateRoot =
ContractUtil.getStateRoot(NulsContext.getInstance().getBestBlock().getHeader());
                ProgramExecutor track = programExecutor.begin(prevStateRoot);
tokenInfo.setStatus(track.status(AddressTool.getAddress(tokenInfo.getContractAddress())).ordinal
```

```
());
                result.setData(tokenInfo);
            }
            return result;
        } catch (Exception e) {
            Log.error("get contract token via VM error.", e);
            return Result.getFailed(ContractErrorCode.SYS_UNKOWN_EXCEPTION);
        }

    }

    @Override
    public Result initAllTokensByAccount(String address) {
        try {
            if (StringUtils.isBlank(address)) {
                return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
            }

            if (!AddressTool.validAddress(address)) {
                return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
            }
            contractBalanceManager.initAllTokensByAccount(address);
            return Result.getSuccess();
        } catch (Exception e) {
            Log.error("initial all tokens of the account error.", e);
            return Result.getFailed(ContractErrorCode.SYS_UNKOWN_EXCEPTION);
        }
    }

    @Override
    public Result<List<ContractTokenInfo>> getAllTokensByAccount(String address) {
        try {
            if (StringUtils.isBlank(address)) {
                return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
            }

            if (!AddressTool.validAddress(address)) {
                return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
            }

            Result<List<ContractTokenInfo>> tokenListResult =
contractBalanceManager.getAllTokensByAccount(address);
```

```
            List<ContractTokenInfo> list = tokenListResult.getData();
            if(list != null && list.size() > 0) {
                byte[] prevStateRoot =
ContractUtil.getStateRoot(NulsContext.getInstance().getBestBlock().getHeader());
                ProgramExecutor track = programExecutor.begin(prevStateRoot);
                for(ContractTokenInfo tokenInfo : list) {
tokenInfo.setStatus(track.status(AddressTool.getAddress(tokenInfo.getContractAddress())).ordinal
());
                }
            }
            return tokenListResult;
        } catch (Exception e) {
            Log.error("initial all tokens of the account error.", e);
            return Result.getFailed(ContractErrorCode.SYS_UNKOWN_EXCEPTION);
        }
    }

    @Override
    public boolean isTokenContractAddress(String contractAddress) {
        try {
            if (StringUtils.isBlank(contractAddress)) {
                return false;
            }

            if (!AddressTool.validAddress(contractAddress)) {
                return false;
            }

            byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
            Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
            ContractAddressInfoPo po = contractAddressInfoResult.getData();
            if(po == null) {
                return false;
            }
            return po.isNrc20();
        } catch (Exception e) {
            Log.error("check if it is a token address error.", e);
            return false;
        }
    }
```

```
    @Override
    public Result<List<ContractTokenTransferInfoPo>> getTokenTransferInfoList(String address) {
        try {
            Result accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult;
            }
            byte[] addressBytes = AddressTool.getAddress(address);

            List<ContractTokenTransferInfoPo> tokenTransferInfoListByAddress =
contractTokenTransferStorageService.getTokenTransferInfoListByAddress(addressBytes);
            return Result.getSuccess().setData(tokenTransferInfoListByAddress);
        } catch (Exception e) {
            Log.error(e);
            return Result.getFailed();
        }
    }

    @Override
    public Result<List<ContractTokenTransferInfoPo>> getTokenTransferInfoList(String address,
NulsDigestData hash) {
        try {
            if(hash == null) {
                return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
            }
            Result accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult;
            }
            byte[] addressBytes = AddressTool.getAddress(address);

            List<ContractTokenTransferInfoPo> tokenTransferInfoListByAddress =
contractTokenTransferStorageService.getTokenTransferInfoListByAddress(addressBytes,
hash.serialize());
            return Result.getSuccess().setData(tokenTransferInfoListByAddress);
        } catch (Exception e) {
            Log.error(e);
            return Result.getFailed();
        }
    }

    @Override
```

```
    public Result<byte[]> handleContractResult(Transaction tx, ContractResult contractResult,
byte[] stateRoot, long time, Map<String, Coin> toMaps, Map<String, Coin> contractUsedCoinMap)
{
        if (contractResult == null) {
            return Result.getSuccess().setData(stateRoot);
        }
        List<ContractTransfer> transfers = contractResult.getTransfers();
        byte[] preStateRoot = stateRoot;
        stateRoot = contractResult.getStateRoot();
        if(tx instanceof CallContractTransaction) {
            // ()
            if (!contractResult.isSuccess() && contractResult.getValue() > 0) {
                Na sendBack = Na.valueOf(contractResult.getValue());
                List<ContractTransfer> transfer = this.createReturnFundsContractTransfer(tx,
sendBack);
                transfers.addAll(transfer);
            }

            // ()()
            stateRoot = this.handleContractTransferTxs((CallContractTransaction) tx, contractResult,
stateRoot, preStateRoot,
                    transfers, time, toMaps, contractUsedCoinMap, null);

        } else {
            //
            Object txTrackObj = contractResult.getTxTrack();
            if(contractResult.isSuccess() && txTrackObj != null && txTrackObj instanceof
ProgramExecutor) {
                ProgramExecutor txTrack = (ProgramExecutor) txTrackObj;
                if(Log.isDebugEnabled()) {
                    Log.debug("===tx track commit.");
                }
                txTrack.commit();
            }
        }

        // DB, GasCoinBase --> method: addConsensusTx
        ContractTransaction contractTx = (ContractTransaction) tx;
        contractTx.setContractResult(contractResult);
        return Result.getSuccess().setData(stateRoot);
    }
```

```
    private Result verifyTransfer(List<ContractTransfer> transfers) {
        if(transfers == null || transfers.size() == 0) {
            return Result.getSuccess();
        }
        for(ContractTransfer transfer : transfers) {
            if (transfer.getValue().isLessThan(ProtocolConstant.MININUM_TRANSFER_AMOUNT)) {
                return Result.getFailed(TransactionErrorCode.TOO_SMALL_AMOUNT);
            }
        }
        return Result.getSuccess();
    }

    private byte[] handleContractTransferTxs(CallContractTransaction tx, ContractResult
contractResult,
                                             byte[] stateRoot, byte[] preStateRoot,
                                             List<ContractTransfer> transfers, long time,
                                             Map<String,Coin> toMaps, Map<String,Coin> contractUsedCoinMap,
Long blockHeight) {
        boolean isCorrectContractTransfer = true;
        // ()
        if (transfers != null && transfers.size() > 0) {

            // ()
            Map<String, ContractTransferTransaction> successContractTransferTxs = new
LinkedHashMap<>();
            Result<ContractTransferTransaction> contractTransferResult;
            do {
                // ()
                Result result = this.verifyTransfer(transfers);
                if(result.isFailed()) {
                    isCorrectContractTransfer = false;
                    contractResult.setError(true);
                    String errorMsg = contractResult.getErrorMessage();
                    errorMsg = errorMsg == null ? result.getErrorCode().getEnMsg() : (errorMsg + "," +
result.getErrorCode().getEnMsg());
                    contractResult.setErrorMessage(errorMsg);
                    break;
                }

                // ()
                ContractTransferTransaction contractTransferTx;
                for (ContractTransfer transfer : transfers) {
```

```
                transfer.setOrginHash(tx.getHash());
                contractTransferResult = this.createContractTransferTx(transfer, time, toMaps,
contractUsedCoinMap, blockHeight);
                if (contractTransferResult.isFailed()) {
                    this.rollbackContractTransferTxs(successContractTransferTxs, toMaps,
contractUsedCoinMap);
                    isCorrectContractTransfer = false;
                    contractResult.setError(true);
                    String errorMsg = contractResult.getErrorMessage();
                    errorMsg = errorMsg == null ? contractTransferResult.getMsg() : (errorMsg + "," +
contractTransferResult.getMsg());
                    contractResult.setErrorMessage(errorMsg);
                    break;
                }
                contractTransferTx = contractTransferResult.getData();
                // hashhash
                transfer.setHash(contractTransferTx.getHash());
                successContractTransferTxs.put(contractTransferTx.getHash().getDigestHex(),
contractTransferTx);
            }
        } while (false);


        // ()
        if (!isCorrectContractTransfer) {
            Log.error("contract transfer execution failed, reason: {}",
contractResult.getErrorMessage());
            //
            stateRoot = preStateRoot;
            contractResult.setStateRoot(stateRoot);
            //
            contractResult.setBalance(contractResult.getPreBalance());
            //
            successContractTransferTxs.clear();
            //
            this.rollbackContractTempBalance(tx, contractResult);
            //
            transfers.clear();

            // ()()
            if (contractResult.getValue() > 0) {
```

```
                Na sendBack = Na.valueOf(contractResult.getValue());
                List<ContractTransfer> transferList = this.createReturnFundsContractTransfer(tx,
sendBack);
                for(ContractTransfer transfer : transferList) {
                    transfer.setOrginHash(tx.getHash());
                    contractTransferResult = this.createContractTransferTx(transfer, time, toMaps,
contractUsedCoinMap, blockHeight);

                    if (contractTransferResult.isFailed()) {
                        successContractTransferTxs.clear();
                        contractResult.setErrorMessage(contractResult.getErrorMessage() + ", " +
contractTransferResult.getMsg());
                        break;
                    } else {
                        ContractTransferTransaction _contractTransferTx =
contractTransferResult.getData();
                        // hashhash
                        transfer.setHash(_contractTransferTx.getHash());
                        transfers.add(transfer);
                        successContractTransferTxs.put(_contractTransferTx.getHash().getDigestHex(),
_contractTransferTx);
                    }
                }
            }
        }
        //
        tx.setContractTransferTxs(successContractTransferTxs.values());
    }

    // ()
    if(contractResult.isSuccess() && isCorrectContractTransfer) {
        Object txTrackObj = contractResult.getTxTrack();
        if(txTrackObj != null && txTrackObj instanceof ProgramExecutor) {
            ProgramExecutor txTrack = (ProgramExecutor) txTrackObj;
            if(Log.isDebugEnabled()) {
                Log.debug("===tx track commit.");
            }
            txTrack.commit();
        }
    }
    return stateRoot;
  }
```

```
    @Override
    public Result<byte[]> verifyContractResult(Transaction tx, ContractResult contractResult, byte[]
stateRoot, long time, Map<String, Coin> toMaps, Map<String, Coin> contractUsedCoinMap) {
        return this.verifyContractResult(tx, contractResult, stateRoot, time, toMaps,
contractUsedCoinMap, null);
    }

    @Override
    public Result<byte[]> verifyContractResult(Transaction tx, ContractResult contractResult, byte[]
stateRoot, long time, Map<String, Coin> toMaps, Map<String, Coin> contractUsedCoinMap, Long
blockHeight) {
        if (contractResult == null) {
            return Result.getSuccess().setData(stateRoot);
        }
        List<ContractTransfer> transfers = contractResult.getTransfers();
        byte[] preStateRoot = stateRoot;
        stateRoot = contractResult.getStateRoot();
        do {
            if(tx instanceof CallContractTransaction) {
                // ()
                if (!contractResult.isSuccess() && contractResult.getValue() > 0) {
                    // (), contractResult,
                    if (transfers.size() == 0) {
                        Na sendBack = Na.valueOf(contractResult.getValue());
                        List<ContractTransfer> transfer = this.createReturnFundsContractTransfer(tx,
sendBack);
                        transfers.addAll(transfer);
                    }
                }
                // ()()
                stateRoot = this.handleContractTransferTxs((CallContractTransaction) tx,
contractResult, stateRoot, preStateRoot,
                        transfers, time, toMaps, contractUsedCoinMap, blockHeight);
            } else {
                //
                Object txTrackObj = contractResult.getTxTrack();
                if(contractResult.isSuccess() && txTrackObj != null && txTrackObj instanceof
ProgramExecutor) {
                    ProgramExecutor txTrack = (ProgramExecutor) txTrackObj;
                    if(Log.isDebugEnabled()) {
                        Log.debug("===tx track commit.");
```

```
                }
                txTrack.commit();
            }
        }
    } while (false);

    // DB, GasCoinBase --> method: addConsensusTx
    ContractTransaction contractTx = (ContractTransaction) tx;
    contractTx.setContractResult(contractResult);
    return Result.getSuccess().setData(stateRoot);
}

@Override
public Result<ContractResult> batchPackageTx(Transaction tx, long bestHeight, Block block,
byte[] stateRoot, Map<String, Coin> toMaps, Map<String, Coin> contractUsedCoinMap) {
    if(stateRoot == null) {
        return Result.getFailed();
    }

    BlockHeader blockHeader = block.getHeader();
    long blockTime = blockHeader.getTime();
    ProgramExecutor executor = localProgramExecutor.get();
    if(executor == null) {
        return Result.getFailed();
    }

    ContractTransaction contractTx = (ContractTransaction) tx;
    contractTx.setBlockHeader(blockHeader);
    //
    Result<ContractResult> invokeContractResult = this.invokeContract(executor, tx, bestHeight,
null, false);
    ContractResult contractResult = invokeContractResult.getData();
    if (contractResult != null) {
        this.handleContractResult(tx, contractResult, stateRoot, blockTime, toMaps,
contractUsedCoinMap);
    }

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

@Override
public Result<ContractResult> batchProcessTx(Transaction tx, long bestHeight, Block block,
```

```
byte[] stateRoot, Map<String, Coin> toMaps, Map<String, Coin> contractUsedCoinMap, boolean isForkChain) {
        if(stateRoot == null) {
            return Result.getFailed();
        }

        BlockHeader blockHeader = block.getHeader();
        long blockTime = blockHeader.getTime();
        ProgramExecutor executor = localProgramExecutor.get();
        if(executor == null) {
            return Result.getFailed();
        }

        ContractTransaction contractTx = (ContractTransaction) tx;
        contractTx.setBlockHeader(blockHeader);
        //
        Result<ContractResult> invokeContractResult = this.invokeContract(executor, tx, bestHeight, null, isForkChain);
        ContractResult contractResult = invokeContractResult.getData();
        if (contractResult != null) {
            Result<byte[]> handleContractResult;
            if(isForkChain) {
                handleContractResult = this.verifyContractResult(tx, contractResult, stateRoot, blockTime, toMaps, contractUsedCoinMap, bestHeight);
            } else {
                handleContractResult = this.verifyContractResult(tx, contractResult, stateRoot, blockTime, toMaps, contractUsedCoinMap);
            }
        }

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

    @Override
    public Result<List<ContractTransferTransaction>> loadAllContractTransferTxList() {
        try {
            List<ContractTransferTransaction> list = contractTransferTransactionStorageService.loadAllContractTransferTxList();
            return Result.getSuccess().setData(list);
        } catch (Exception e) {
            Log.error(e);
            return Result.getFailed();
```

```
        }
    }

    @Override
    public void createBatchExecute(byte[] stateRoot) {
        localProgramExecutor.remove();
        if(stateRoot == null) {
            return;
        }
        ProgramExecutor executor = programExecutor.begin(stateRoot);
        localProgramExecutor.set(executor);
    }

    @Override
    public Result<byte[]> commitBatchExecute() {
        ProgramExecutor executor = localProgramExecutor.get();
        if(executor == null) {
            return Result.getSuccess();
        }
        executor.commit();
        byte[] stateRoot = executor.getRoot();
        return Result.getSuccess().setData(stateRoot);
    }

    @Override
    public void removeBatchExecute() {
        localProgramExecutor.remove();
    }

    @Override
    public void createCurrentBlockHeader(BlockHeader tempHeader) {
        vmContext.createCurrentBlockHeader(tempHeader);
    }

    @Override
    public void removeCurrentBlockHeader() {
        vmContext.removeCurrentBlockHeader();
    }
}
```

2:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-base\src\main\java\io\nuls\contract\util\ContractCoinComparator.java

```
 */

package io.nuls.contract.util;

import io.nuls.kernel.model.Coin;

import java.util.Comparator;

public class ContractCoinComparator implements Comparator<Coin> {

    private static ContractCoinComparator instance = new ContractCoinComparator();

    private ContractCoinComparator() {

    }

    public static ContractCoinComparator getInstance() {
        return instance;
    }

    @Override
    public int compare(Coin o1, Coin o2) {
        return o1.getNa().compareTo(o2.getNa());
    }
}

3:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
ledger\src\main\java\io\nuls\contract\ledger\manager\ContractBalanceManager.java
 */

package io.nuls.contract.ledger.manager;

import io.nuls.account.model.Account;
import io.nuls.account.service.AccountService;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.dto.ContractTokenInfo;
import io.nuls.contract.ledger.module.ContractBalance;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.contract.storage.service.ContractUtxoStorageService;
import io.nuls.core.tools.log.Log;
```

```
import io.nuls.core.tools.map.MapUtil;
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.Address;
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.math.BigInteger;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

import static io.nuls.ledger.util.LedgerUtil.asString;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/7
 */
@Component
public class ContractBalanceManager {

    @Autowired
    private ContractService contractService;

    @Autowired
    private ContractUtxoStorageService contractUtxoStorageService;

    @Autowired
    private ContractAddressStorageService contractAddressStorageService;

    @Autowired
    private AccountService accountService;

    /**
     * key: String - local account address
```

```
     *      value:
     *          key: String - contract address
     *          value: ContractTokenInfo - token name && amount
     */
    private Map<String, Map<String, ContractTokenInfo>> contractTokenOfLocalAccount = new ConcurrentHashMap<>();

    private Map<String, ContractBalance> balanceMap;

    private Lock lock = new ReentrantLock();

    private Lock tokenLock = new ReentrantLock();

    private ThreadLocal<Map<String, ContractBalance>> tempBalanceMapManager = new ThreadLocal<>();

    public void createTempBalanceMap() {
        tempBalanceMapManager.remove();
        tempBalanceMapManager.set(new ConcurrentHashMap<>());
    }

    public void removeTempBalanceMap() {
        tempBalanceMapManager.remove();
    }

    /**
     *
     */
    public void initContractBalance() {
        balanceMap = new ConcurrentHashMap<>();
        List<Entry<byte[], byte[]>> rawList = contractUtxoStorageService.loadAllCoinList();
        Coin coin;
        String strAddress;
        ContractBalance balance;
        for (Entry<byte[], byte[]> coinEntry : rawList) {
            coin = new Coin();
            try {
                coin.parse(coinEntry.getValue(), 0);
                //strAddress = asString(coin.getOwner());
                coin.setKey(asString(coinEntry.getKey()));
                strAddress = asString(coin.getAddress());
            } catch (NulsException e) {
```

```
            Log.error("parse contract coin error form db", e);
            continue;
        }
        balance = balanceMap.get(strAddress);
        if(balance == null) {
            balance = new ContractBalance();
            balanceMap.put(strAddress, balance);
        }
        // utxo
        if(coin.getLockTime() != 0) {
            balance.getConsensusRewardCoins().put(coin.getKey(), coin);
        } else {
            balance.addUsable(coin.getNa());
        }
    }
}

/**
 *
 */
public void refreshBalance(List<Coin> addUtxoList, List<Coin> deleteUtxoList) {
    lock.lock();
    try {
        ContractBalance balance;
        String strAddress;

        if(deleteUtxoList != null) {
            for (Coin coin : deleteUtxoList) {
                strAddress = asString(coin.getTempOwner());
                balance = balanceMap.get(strAddress);
                if(balance == null) {
                    balance = new ContractBalance();
                    balanceMap.put(strAddress, balance);
                }
                // utxo
                if(coin.getLockTime() != 0) {
                    balance.getConsensusRewardCoins().remove(coin.getKey());
                } else {
                    balance.minusUsable(coin.getNa());
                }
            }
        }
```

```
        if(addUtxoList != null) {
            for (Coin coin : addUtxoList) {
                strAddress = asString(coin.getTempOwner());
                balance = balanceMap.get(strAddress);
                if(balance == null) {
                    balance = new ContractBalance();
                    balanceMap.put(strAddress, balance);
                }
                // utxo
                if(coin.getLockTime() != 0) {
                    balance.getConsensusRewardCoins().put(coin.getKey(), coin);
                } else {
                    balance.addUsable(coin.getNa());
                }
            }
        }
    } finally {
        lock.unlock();
    }
}

/**
 *
 *
 * @param address
 * @return
 */
public Result<ContractBalance> getBalance(byte[] address) {
    return getBalance(address, NulsContext.getInstance().getBestHeight());
}
/**
 *
 *
 * @param address
 * @param bestHeight
 * @return
 */
public Result<ContractBalance> getBalance(byte[] address, Long blockHeight) {
    lock.lock();
    try {
        if (address == null || address.length != Address.ADDRESS_LENGTH) {
```

```
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR);
        }

        String addressKey = asString(address);
        ContractBalance balance;
        //
        Map<String, ContractBalance> tempBalanceMap = tempBalanceMapManager.get();
        if(tempBalanceMap != null) {
            balance = tempBalanceMap.get(addressKey);
            //
            if(balance == null) {
                balance = balanceMap.get(addressKey);
                //
                if (balance == null) {
                    balanceMap.put(addressKey, new ContractBalance());
                    balance = new ContractBalance();
                    tempBalanceMap.put(addressKey, balance);
                } else {
                    //
                    //
                    this.handleLockedBalances(balance, blockHeight);
                    balance = depthClone(balance);
                    tempBalanceMap.put(addressKey, balance);
                }
            }
        } else {
            balance = balanceMap.get(addressKey);
            if (balance == null) {
                balance = new ContractBalance();
                balanceMap.put(addressKey, balance);
            } else {
                //
                this.handleLockedBalances(balance, blockHeight);
            }
        }

        return Result.getSuccess().setData(balance);
    } finally {
        lock.unlock();
    }
}
```

```
    private void handleLockedBalances(ContractBalance balance, Long blockHeight) {
        balance.setLocked(Na.ZERO);
        balance.setUsableConsensusReward(Na.ZERO);
        Collection<Coin> lockedCoins = balance.getConsensusRewardCoins().values();
        List<Coin> list = new ArrayList<>(lockedCoins);
        Coin coin;
        int size = list.size();
        for(int i = 0; i < size; i++) {
            coin = list.get(i);
            if(coin.usable(blockHeight)) {
                balance.addUsableConsensusReward(coin.getNa());
            } else {
                balance.addLocked(coin.getNa());
            }
        }

    }

    private ContractBalance depthClone(ContractBalance contractBalance) {
        if(contractBalance == null) {
            return null;
        }
        LinkedHashMap<String, Coin> lockedCoins = contractBalance.getConsensusRewardCoins();
        LinkedHashMap<String, Coin> lockedCoinsClone =
MapUtil.createLinkedHashMap(lockedCoins.size());
        Collection<Coin> values = lockedCoins.values();
        for(Coin coin : values) {
            lockedCoinsClone.put(coin.getKey(), coin);
        }
        ContractBalance result = new
ContractBalance(Na.valueOf(contractBalance.getUsable().getValue()),
                Na.valueOf(contractBalance.getLocked().getValue()),
Na.valueOf(contractBalance.getUsableConsensusReward().getValue()), lockedCoinsClone);
        return result;
    }

    public void addTempBalance(byte[] address, Na amount) {
        Map<String, ContractBalance> tempBalanceMap = tempBalanceMapManager.get();
        String addressKey = asString(address);
        ContractBalance contractBalance = tempBalanceMap.get(addressKey);

        if(contractBalance != null) {
```

```
            contractBalance.addTempUsable(amount);
        }
    }

    public void minusTempBalance(byte[] address, Na amount) {
        Map<String, ContractBalance> tempBalanceMap = tempBalanceMapManager.get();
        String addressKey = asString(address);
        ContractBalance contractBalance = tempBalanceMap.get(addressKey);

        if(contractBalance != null) {
            contractBalance.minusTempUsable(amount);
        }
    }

    public List<Coin> getCoinListByAddress(byte[] address) {
        List<Coin> coinList = new ArrayList<>();
        List<Entry<byte[], byte[]>> rawList = contractUtxoStorageService.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.setOwner(coinEntry.getKey());
                coin.setKey(asString(coinEntry.getKey()));
                coinList.add(coin);
            }
        }
        return coinList;
    }

    public void initialContractToken(String account, String contract) {
        tokenLock.lock();
        try {
            Result<ContractTokenInfo> result = contractService.getContractTokenViaVm(account,
contract);
            if(result.isFailed()) {
                return;
            }
```

```
            ContractTokenInfo tokenInfo = result.getData();
            BigInteger amount = tokenInfo.getAmount();
            if(amount == null || amount.equals(BigInteger.ZERO)) {
                return;
            }
            Map<String, ContractTokenInfo> tokens = contractTokenOfLocalAccount.get(account);
            if(tokens == null) {
                tokens = new HashMap<>();
            }
            tokens.put(contract, tokenInfo);
            contractTokenOfLocalAccount.put(account, tokens);
        } finally {
            tokenLock.unlock();
        }
    }

    public void refreshContractToken(String account, String contract, ContractAddressInfoPo po,
BigInteger value) {
        tokenLock.lock();
        try {
            ContractTokenInfo tokenInfo = new ContractTokenInfo(contract,
po.getNrc20TokenName(), po.getDecimals(), value, po.getNrc20TokenSymbol(),
po.getBlockHeight());
            Map<String, ContractTokenInfo> tokens = contractTokenOfLocalAccount.get(account);
            if(tokens == null) {
                tokens = new HashMap<>();
            }
            tokens.put(contract, tokenInfo);
            contractTokenOfLocalAccount.put(account, tokens);
        } finally {
            tokenLock.unlock();
        }
    }

    public void initAllTokensForAllAccounts() {
        Result<Collection<Account>> result = accountService.getAccountList();
        if(result.isFailed()) {
            return;
        }
        Result<List<ContractAddressInfoPo>> allContractInfoListResult =
contractAddressStorageService.getAllNrc20ContractInfoList();
        if(allContractInfoListResult.isFailed()) {
```

```
            return;
        }
        List<ContractAddressInfoPo> contractAddressInfoPoList =
allContractInfoListResult.getData();

        Collection<Account> list = result.getData();
        for(Account account : list) {
            Address address = account.getAddress();
            String addressStr = address.getBase58();
            for(ContractAddressInfoPo po : contractAddressInfoPoList) {
                initialContractToken(addressStr,
AddressTool.getStringAddressByBytes(po.getContractAddress()));
            }
        }
    }

    public void initAllTokensByAccount(String account) {
        if(!AddressTool.validAddress(account)) {
            return;
        }
        Result<List<ContractAddressInfoPo>> allContractInfoListResult =
contractAddressStorageService.getAllNrc20ContractInfoList();
        if(allContractInfoListResult.isFailed()) {
            return;
        }
        List<ContractAddressInfoPo> contractAddressInfoPoList =
allContractInfoListResult.getData();
        for(ContractAddressInfoPo po : contractAddressInfoPoList) {
            initialContractToken(account,
AddressTool.getStringAddressByBytes(po.getContractAddress()));
        }
    }

    public Result<List<ContractTokenInfo>> getAllTokensByAccount(String account) {
        Map<String, ContractTokenInfo> tokensMap = contractTokenOfLocalAccount.get(account);
        if(tokensMap == null || tokensMap.size() == 0) {
            return Result.getSuccess().setData(new ArrayList<>());
        }
        List<ContractTokenInfo> resultList = new ArrayList<>();
        Set<Map.Entry<String, ContractTokenInfo>> entries = tokensMap.entrySet();
        String contractAddress;
        ContractTokenInfo info;
```

```
    for(Map.Entry<String, ContractTokenInfo> entry : entries) {
        contractAddress = entry.getKey();
        info = entry.getValue();
        info.setContractAddress(contractAddress);
        resultList.add(info);
    }
    return Result.getSuccess().setData(resultList);
}

public Result subtractContractToken(String account, String contract, BigInteger token) {
    tokenLock.lock();
    try {
        Map<String, ContractTokenInfo> tokens = contractTokenOfLocalAccount.get(account);
        if(tokens == null) {
            return Result.getSuccess();
        } else {
            ContractTokenInfo info = tokens.get(contract);
            if(info == null) {
                return Result.getSuccess();
            }
            BigInteger currentToken = info.getAmount();
            if(currentToken == null) {
                return Result.getSuccess();
            } else {
                if(currentToken.compareTo(token) < 0) {
                    return Result.getFailed(ContractErrorCode.INSUFFICIENT_BALANCE);
                }
                currentToken = currentToken.subtract(token);
                tokens.put(contract, info.setAmount(currentToken));
            }
        }
        return Result.getSuccess();
    } finally {
        tokenLock.unlock();
    }
}

public Result addContractToken(String account, String contract, BigInteger token) {
    tokenLock.lock();
    try {
        Map<String, ContractTokenInfo> tokens = contractTokenOfLocalAccount.get(account);
        do {
```

```
                if(tokens == null) {
                    break;
                } else {
                    ContractTokenInfo info = tokens.get(contract);
                    if(info == null) {
                        return Result.getSuccess();
                    }
                    BigInteger currentToken = info.getAmount();
                    if(currentToken == null) {
                        break;
                    } else {
                        currentToken = currentToken.add(token);
                        tokens.put(contract, info.setAmount(currentToken));
                    }
                }
            } while(false);
        } finally {
            tokenLock.unlock();
        }
        return Result.getSuccess();
    }
}
```

4:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-ledger\src\main\java\io\nuls\contract\ledger\module\ContractBalance.java

```
package io.nuls.contract.ledger.module;

import com.fasterxml.jackson.annotation.JsonIgnore;
import io.nuls.core.tools.map.MapUtil;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Na;

import java.io.Serializable;
import java.util.LinkedHashMap;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/7
 */
public class ContractBalance implements Serializable {
```

```
    private Na balance;

    private Na locked;

    private Na usable;

    private Na usableConsensusReward;

    private LinkedHashMap<String, Coin> consensusRewardCoins;

    public ContractBalance() {
        this.locked = Na.ZERO;
        this.usable = Na.ZERO;
        this.usableConsensusReward = Na.ZERO;
        this.consensusRewardCoins = MapUtil.createLinkedHashMap(64);
    }

    public ContractBalance(Na usable, Na locked, Na usableConsensusReward,
LinkedHashMap<String, Coin> lockedCoins) {
        if (usable == null) {
            usable = Na.ZERO;
        }
        if (locked == null) {
            locked = Na.ZERO;
        }
        if (usableConsensusReward == null) {
            usableConsensusReward = Na.ZERO;
        }
        this.usable = usable;
        this.locked = locked;
        this.usableConsensusReward = usableConsensusReward;
        this.consensusRewardCoins = lockedCoins;

    }

    public Na getBalance() {
        this.balance = this.getRealUsable().add(locked);
        return balance;
    }

    public void setLocked(Na locked) {
        this.locked = locked;
```

```
}

public Na getLocked() {
    return locked;
}

@JsonIgnore
public Na getRealUsable() {
    if (usableConsensusReward == null) {
        usableConsensusReward = Na.ZERO;
    }
    return usable.add(usableConsensusReward);
}

public Na getUsable() {
    return usable;
}

@JsonIgnore
public LinkedHashMap<String, Coin> getConsensusRewardCoins() {
    return consensusRewardCoins;
}

public void addLocked(Na locked) {
    this.locked = this.locked.add(locked);
}

public void addUsable(Na usable) {
    this.usable = this.usable.add(usable);
}

public void minusUsable(Na usable) {
    this.usable = this.usable.minus(usable);
}

public Na getUsableConsensusReward() {
    return usableConsensusReward;
}

public void setUsableConsensusReward(Na usableConsensusReward) {
    this.usableConsensusReward = usableConsensusReward;
}
```

```
    public void addUsableConsensusReward(Na usableConsensusReward) {
        this.usableConsensusReward =
this.usableConsensusReward.add(usableConsensusReward);
    }

    public void minusTempUsable(Na amount) {
        Na realUsable = this.getRealUsable();
        Na tempUsable = realUsable.minus(amount);
        this.usableConsensusReward = Na.ZERO;
        this.consensusRewardCoins.clear();
        this.usable = tempUsable;
    }

    public void addTempUsable(Na amount) {
        this.addUsable(amount);
    }
}

5:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
ledger\src\main\java\io\nuls\contract\ledger\service\ContractTransactionInfoService.java
 */

package io.nuls.contract.ledger.service;

import io.nuls.contract.storage.po.TransactionInfoPo;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
public interface ContractTransactionInfoService {

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

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

    boolean isDbExistTransactionInfo(TransactionInfoPo infoPo, byte[] address);
```

```
    Result deleteTransactionInfo(TransactionInfoPo infoPo, List<byte[]> addresses);
}

6:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
ledger\src\main\java\io\nuls\contract\ledger\service\ContractUtxoService.java
 */
package io.nuls.contract.ledger.service;

import io.nuls.account.model.Balance;
import io.nuls.contract.ledger.module.ContractBalance;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;

import java.math.BigInteger;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
public interface ContractUtxoService {

    /**
     *
     *
     *      -> txtoCoinData -> UTXO
     *
     *      -> txfromCoinData -> UTXO
     *
     * @param tx
     * @return
     */
    Result saveUtxoForContractAddress(Transaction tx);

    Result deleteUtxoOfTransaction(Transaction tx);

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

    Result<ContractBalance> getBalance(byte[] address, Long blockHeight);
}
```

7:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-ledger\src\main\java\io\nuls\contract\ledger\service\impl\ContractTransactionInfoServiceImpl.java
 */
```
package io.nuls.contract.ledger.service.impl;

import io.nuls.account.ledger.constant.AccountLedgerErrorCode;
import io.nuls.contract.ledger.service.ContractTransactionInfoService;
import io.nuls.contract.storage.po.TransactionInfoPo;
import io.nuls.contract.storage.service.ContractTransactionInfoStorageService;
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.Service;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Result;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
@Service
public class ContractTransactionInfoServiceImpl implements ContractTransactionInfoService {

    @Autowired
    private ContractTransactionInfoStorageService contractTransactionInfoStorageService;

    @Override
    public Result<List<TransactionInfoPo>> getTxInfoList(byte[] address) {
        try {
            List<TransactionInfoPo> infoPoList = contractTransactionInfoStorageService.getTransactionInfoListByAddress(address);
            return Result.getSuccess().setData(infoPoList);
        } 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 {
            byte[] txHashBytes = infoPo.getTxHash().serialize();
            int txHashLength = infoPo.getTxHash().size();
            byte[] infoKey;
            for (int i = 0; i < addresses.size(); i++) {
                infoKey = new byte[Address.ADDRESS_LENGTH + txHashLength];
                System.arraycopy(addresses.get(i), 0, infoKey, 0, Address.ADDRESS_LENGTH);
                System.arraycopy(txHashBytes, 0, infoKey, Address.ADDRESS_LENGTH,
txHashLength);
                contractTransactionInfoStorageService.saveTransactionInfo(infoKey, infoPo);
                savedKeyList.add(infoKey);
            }
        } catch (IOException e) {
            for (int i = 0; i < savedKeyList.size(); i++) {
                contractTransactionInfoStorageService.deleteTransactionInfo(savedKeyList.get(i));
            }
            return Result.getFailed(AccountLedgerErrorCode.IO_ERROR);
        }
        return Result.getSuccess().setData(new Integer(addresses.size()));
    }

    @Override
    public boolean isDbExistTransactionInfo(TransactionInfoPo infoPo, byte[] address) {
        try {
            byte[] txHashBytes = infoPo.getTxHash().serialize();
            int txHashLength = infoPo.getTxHash().size();
```

```
            byte[] infoKey = new byte[Address.ADDRESS_LENGTH + txHashLength];
            System.arraycopy(address, 0, infoKey, 0, Address.ADDRESS_LENGTH);
            System.arraycopy(txHashBytes, 0, infoKey, Address.ADDRESS_LENGTH, txHashLength);
            Result<byte[]> txInfoBytesResult =
contractTransactionInfoStorageService.getTransactionInfo(infoKey);
            if(txInfoBytesResult.getData() == null) {
                return false;
            } else {
                return true;
            }
        } catch (IOException e) {
            Log.error(e);
            return false;
        }
    }

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

        int addressCount = addresses.size();

        byte[] txHashBytes;
        try {
            txHashBytes = infoPo.getTxHash().serialize();
        } catch (IOException e) {
            Log.error(e);
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR);
        }
        for (int i = 0; i < addressCount; i++) {
            contractTransactionInfoStorageService.deleteTransactionInfo(
                    ArraysTool.concatenate(addresses.get(i), txHashBytes));
        }
        return Result.getSuccess().setData(new Integer(addressCount));
    }
}
```

8:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-ledger\src\main\java\io\nuls\contract\ledger\service\impl\ContractUtxoServiceImpl.java

```
 */
package io.nuls.contract.ledger.service.impl;

import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.ledger.manager.ContractBalanceManager;
import io.nuls.contract.ledger.module.ContractBalance;
import io.nuls.contract.ledger.service.ContractUtxoService;
import io.nuls.contract.storage.service.ContractTransferTransactionStorageService;
import io.nuls.contract.storage.service.ContractUtxoStorageService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.crypto.Hex;
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.math.BigInteger;
import java.util.ArrayList;
import java.util.List;

import static io.nuls.ledger.util.LedgerUtil.asString;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
@Component
public class ContractUtxoServiceImpl implements ContractUtxoService {

    @Autowired
    private LedgerService ledgerService;
```

```
    @Autowired
    private ContractUtxoStorageService contractUtxoStorageService;

    @Autowired
    private ContractTransferTransactionStorageService
contractTransferTransactionStorageService;

    @Autowired
    private ContractBalanceManager contractBalanceManager;

    @Autowired
    private AccountLedgerService accountLedgerService;



    /**
     *
     *
     *       -
     *       -
     *   /
     *
     * @param tx
     * @return
     */
    @Override
    public Result saveUtxoForContractAddress(Transaction tx) {
        if (tx == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }

        CoinData coinData = tx.getCoinData();

        if (coinData != null) {
            // ()fromCoinData -> delete - from
            List<byte[]> fromList = new ArrayList<>();
            //
            List<Coin> froms = new ArrayList<>();
            List<Coin> deleteFroms = new ArrayList<>();
            if(tx.getType() == ContractConstant.TX_TYPE_CONTRACT_TRANSFER) {
                froms = coinData.getFrom();
                byte[] fromSource;
                byte[] utxoFromTxHash;
```

```
            byte[] utxoFromIndex;
            int txHashSize = tx.getHash().size();
            Coin fromOfFromCoin;
            for (Coin from : froms) {
                fromSource = from.getOwner();
                utxoFromTxHash = new byte[txHashSize];
                utxoFromIndex = new byte[fromSource.length - txHashSize];
                System.arraycopy(fromSource, 0, utxoFromTxHash, 0, txHashSize);
                System.arraycopy(fromSource, txHashSize, utxoFromIndex, 0,
utxoFromIndex.length);

                fromOfFromCoin = from.getFrom();

                if (fromOfFromCoin == null) {
                    Transaction sourceTx = null;
                    try {
                        sourceTx =
ledgerService.getTx(NulsDigestData.fromDigestHex(Hex.encode(utxoFromTxHash)));

                    } catch (Exception e) {
                        throw new NulsRuntimeException(e);
                    }
                    if (sourceTx == null) {
                        return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
                    }
                    fromOfFromCoin = sourceTx.getCoinData().getTo().get((int) new
VarInt(utxoFromIndex, 0).value);
                }

                //
                if (!ContractUtil.isLegalContractAddress(fromOfFromCoin.getOwner())) {
                    continue;
                }

                from.setFrom(fromOfFromCoin);
                from.setTempOwner(fromOfFromCoin.getOwner());
                from.setKey(asString(fromSource));
                deleteFroms.add(from);
                fromList.add(fromSource);
            }
        }
```

```
        // save utxo - to
        List<Coin> tos = coinData.getTo();
        List<Coin> contractTos = new ArrayList<>();
        List<Entry<byte[], byte[]>> toList = new ArrayList<>();
        byte[] txHashBytes;
        try {
            txHashBytes = tx.getHash().serialize();
        } catch (IOException e) {
            throw new NulsRuntimeException(e);
        }
        Coin to;
        byte[] toAddress;
        byte[] outKey;
        for (int i = 0, length = tos.size(); i < length; i++) {
            to = tos.get(i);
            //toAddress = to.getOwner();
            //
            toAddress = to.getAddress();
            if (!ContractUtil.isLegalContractAddress(toAddress)) {
                continue;
            }
            try {
                outKey = ArraysTool.concatenate(txHashBytes, new VarInt(i).encode());
                to.setTempOwner(toAddress);
                to.setKey(asString(outKey));
                contractTos.add(to);
                toList.add(new Entry<byte[], byte[]>(outKey, to.serialize()));
            } catch (IOException e) {
                throw new NulsRuntimeException(e);
            }
        }
        Result<List<Entry<byte[], byte[]>>> result =
contractUtxoStorageService.batchSaveAndDeleteUTXO(toList, fromList);
        if (result.isFailed() || result.getData() == null) {
            return Result.getFailed();
        }
        //
        contractBalanceManager.refreshBalance(contractTos, deleteFroms);
    }
    return Result.getSuccess();
}
```

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

    CoinData coinData = tx.getCoinData();
    byte[] txHashBytes;
    try {
        txHashBytes = tx.getHash().serialize();
    } catch (IOException e) {
        throw new NulsRuntimeException(e);
    }
    if (coinData != null) {
        // delete utxo - to
        List<Coin> tos = coinData.getTo();
        List<Coin> contractTos = new ArrayList<>();
        List<byte[]> toList = new ArrayList<>();
        byte[] outKey;
        Coin to;
        byte[] toAddress;
        for (int i = 0, length = tos.size(); i < length; i++) {
            to = tos.get(i);
            //toAddress = to.();
            toAddress = to.getAddress();
            if(!ContractUtil.isLegalContractAddress(toAddress)) {
                continue;
            }
            outKey = ArraysTool.concatenate(txHashBytes, new VarInt(i).encode());
            to.setTempOwner(toAddress);
            to.setKey(asString(outKey));
            contractTos.add(to);
            toList.add(outKey);
        }

        // save - from
        List<Entry<byte[], byte[]>> fromList = new ArrayList<>();
        List<Coin> froms = new ArrayList<>();
        if(tx.getType() == ContractConstant.TX_TYPE_CONTRACT_TRANSFER) {
            froms = coinData.getFrom();
            int txHashSize = tx.getHash().size();
            byte[] fromSource;
```

```
            byte[] utxoFromHash;
            byte[] utxoFromIndex;
            Transaction sourceTx;
            Coin sourceTxCoinTo;
            for (Coin from : froms) {
                fromSource = from.getOwner();
                utxoFromHash = new byte[txHashSize];
                utxoFromIndex = new byte[fromSource.length - txHashSize];
                System.arraycopy(fromSource, 0, utxoFromHash, 0, txHashSize);
                System.arraycopy(fromSource, txHashSize, utxoFromIndex, 0,
utxoFromIndex.length);

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

                sourceTxCoinTo = sourceTx.getCoinData().getTo().get((int) new
VarInt(utxoFromIndex, 0).value);

                if(!ContractUtil.isLegalContractAddress(sourceTxCoinTo.getAddress())) {
                    continue;
                }

                from.setFrom(sourceTxCoinTo);
                from.setTempOwner(sourceTxCoinTo.getAddress());
                from.setKey(asString(fromSource));
                try {
                    fromList.add(new Entry<byte[], byte[]>(fromSource, sourceTxCoinTo.serialize()));
                } catch (IOException e) {
                    throw new NulsRuntimeException(e);
                }
            }
        }

        // to
        Result<List<Entry<byte[], byte[]>>> result =
```

```
contractUtxoStorageService.batchSaveAndDeleteUTXO(fromList, toList);
            if (result.isFailed() || result.getData() == null) {
                return Result.getFailed();
            }
            // , from  to
            contractBalanceManager.refreshBalance(froms, contractTos);

        }
        return Result.getSuccess();
    }

    @Override
    public Result<ContractBalance> getBalance(byte[] address) {
        return getBalance(address, null);
    }

    @Override
    public Result<ContractBalance> getBalance(byte[] address, Long blockHeight) {
        if (address == null || address.length != Address.ADDRESS_LENGTH) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR);
        }
        ContractBalance contractBalance;
        if(blockHeight != null) {
            contractBalance = contractBalanceManager.getBalance(address, blockHeight).getData();
        } else {
            contractBalance = contractBalanceManager.getBalance(address).getData();
        }

        if(contractBalance == null) {
            return Result.getFailed(ContractErrorCode.DATA_ERROR);
        }

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

9:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-ledger\src\main\java\io\nuls\contract\ledger\util\ContractLedgerUtil.java

```
package io.nuls.contract.ledger.util;

import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.kernel.lite.annotation.Autowired;
```

```
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Transaction;

import java.util.ArrayList;
import java.util.List;

import static io.nuls.contract.util.ContractUtil.isLegalContractAddress;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/6
 */
@Component
public class ContractLedgerUtil {

    @Autowired
    private static ContractAddressStorageService contractAddressStorageService;

    public static boolean isExistContractAddress(byte[] addressBytes) {
        if(addressBytes == null) {
            return false;
        }
        return contractAddressStorageService.isExistContractAddress(addressBytes);
    }

    /**
     * tx
     *
     * @param tx
     * @return
     */
    public static List<byte[]> getRelatedAddresses(Transaction tx) {
        List<byte[]> result = new ArrayList<>();
        if (tx == null) {
            return result;
        }
        List<byte[]> txAddressList = tx.getAllRelativeAddress();
        if (txAddressList == null || txAddressList.size() == 0) {
            return result;
        }
        for (byte[] txAddress : txAddressList) {
```

```
            if(isLegalContractAddress(txAddress)) {
                result.add(txAddress);
            }
        }

        return result;
    }

}
```

10:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\cmd\CallContractProcessor.java
*/

```
package io.nuls.contract.rpc.cmd;

import io.nuls.contract.rpc.form.ContractCall;
import io.nuls.contract.rpc.form.ContractCreate;
import io.nuls.core.tools.json.JSONUtils;
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.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;
import jline.console.ConsoleReader;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static io.nuls.kernel.utils.CommandHelper.getContractCallArgsJson;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/9/19
 */
```

```
public class CallContractProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    private ThreadLocal<ContractCall> paramsData = new ThreadLocal<>();

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<sender>            source address    -required")
                .newLine("\t<gasLimit>          gas limit    -required")
                .newLine("\t<price>             price (Unit: Na/Gas)    -required")
                .newLine("\t<contractAddress>   contract address    -required")
                .newLine("\t<methodName>        the method to call    -required")
                .newLine("\t<value>             transfer nuls to the contract (Unit: Nuls)    -required")
                .newLine("\t[-d methodDesc]       the method description    -not required")
                .newLine("\t[-r remark]           remark    -not required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "callcontract <sender> <gasLimit> <price> <contractAddress> <methodName>
<value> [-d methodDesc] [-r remark] --call contract";
    }

    @Override
    public boolean argsValidate(String[] args) {
        boolean result;
        do {
            int length = args.length;
            if (length != 7 && length != 9 && length != 11) {
                result = false;
                break;
            }
            if (!CommandHelper.checkArgsIsNull(args)) {
```

```
            result = false;
            break;
        }

        // gasLimit
        if (!StringUtils.isNumeric(args[2])) {
            result = false;
            break;
        }
        // price
        if (!StringUtils.isNumeric(args[3])) {
            result = false;
            break;
        }
        // value
        if (!StringUtils.isNumeric(args[6])) {
            result = false;
            break;
        }
        ContractCall form = getContractCall(args);
        if(null == form){
            result = false;
            break;
        }
        paramsData.set(form);

        result = form.getValue() >= 0;
    } while (false);
    return result;
}

private ContractCall getContractCall(String[] args) {
    ContractCall call = null;
    try {
        call = new ContractCall();
        call.setSender(args[1].trim());
        call.setGasLimit(Long.valueOf(args[2].trim()));
        call.setPrice(Long.valueOf(args[3].trim()));
        call.setContractAddress(args[4].trim());
        call.setMethodName(args[5].trim());
        long naValue = 0L;
        Na na = Na.parseNuls(args[6].trim());
```

```
    if (na != null) {
        naValue = na.getValue();
    }
    call.setValue(naValue);

    if(args.length == 9) {
        String argType = args[7].trim();
        if(argType.equals("-d")) {
            call.setMethodDesc(args[8].trim());
        } else if(argType.equals("-r")) {
            call.setRemark(args[8].trim());
        } else {
            return null;
        }
    }else if(args.length == 11) {
        String argType0 = args[7].trim();
        String argType1 = args[9].trim();
        boolean isType0D = argType0.equals("-d");
        boolean isType1D = argType1.equals("-d");
        boolean isType0R = argType0.equals("-r");
        boolean isType1R = argType1.equals("-r");
        if((isType0D && isType1D) || (isType0R && isType1R)) {
            // -d-r
            return null;
        }
        if(isType0D) {
            call.setMethodDesc(args[8].trim());
        }
        if(isType0R) {
            call.setRemark(args[8].trim());
        }
        if(isType1D) {
            call.setMethodDesc(args[10].trim());
        }
        if(isType1R) {
            call.setRemark(args[10].trim());
        }
    }
    return call;
} catch (Exception e) {
    e.fillInStackTrace();
    return null;
```

```
    }
}


@Override
public CommandResult execute(String[] args) {
    ContractCall form = paramsData.get();
    if (null == form) {
        form = getContractCall(args);
    }
    if (null == form) {
        return CommandResult.getFailed("parameter error.");
    }
    String sender = form.getSender();
    RpcClientResult res = CommandHelper.getPassword(sender, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String) res.getData();

    res = getContractCallArgsJson();
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    Object[] contractArgs = (Object[]) res.getData();

    Map<String, Object> parameters = new HashMap<>();
    parameters.put("sender", sender);
    parameters.put("gasLimit", form.getGasLimit());
    parameters.put("price", form.getPrice());
    parameters.put("password", password);
    parameters.put("remark", form.getRemark());
    parameters.put("contractAddress", form.getContractAddress());
    parameters.put("value", form.getValue());
    parameters.put("methodName", form.getMethodName());
    parameters.put("methodDesc", form.getMethodDesc());
    parameters.put("args", contractArgs);
    RpcClientResult result = restFul.post("/contract/call", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(result);
```

```
    }


}

11:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\CreateContractProcessor.java
 */

package io.nuls.contract.rpc.cmd;

import io.nuls.contract.rpc.form.ContractCreate;
import io.nuls.core.tools.json.JSONUtils;
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 jline.console.ConsoleReader;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/9/19
 */
public class CreateContractProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    private ThreadLocal<ContractCreate> paramsData = new ThreadLocal<>();

    @Override
    public String getCommand() {
        return "createcontract";
```

```
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<sender>          source address    -required")
                .newLine("\t<gasLimit>       gas limit    -required")
                .newLine("\t<price>          price (Unit: Na/Gas)    -required")
                .newLine("\t<contractCode>   contract code    -required")
                .newLine("\t[remark]         remark    -not required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "createcontract <sender> <gasLimit> <price> <contractCode> [remark] --create
contract";
    }

    @Override
    public boolean argsValidate(String[] args) {
        boolean result;
        do {
            int length = args.length;
            if (length != 5 && length != 6) {
                result = false;
                break;
            }
            if (!CommandHelper.checkArgsIsNull(args)) {
                result = false;
                break;
            }

            // gasLimit
            if (!StringUtils.isNumeric(args[2])) {
                result = false;
                break;
            }
            // price
            if (!StringUtils.isNumeric(args[3])) {
                result = false;
```

```
            break;
        }
        ContractCreate form = getContractCreate(args);
        if(null == form){
            result = false;
            break;
        }
        paramsData.set(form);
        result = true;
    } while (false);
    return result;
}

private ContractCreate getContractCreate(String[] args) {
    ContractCreate create = null;
    try {
        create = new ContractCreate();
        create.setSender(args[1].trim());
        create.setGasLimit(Long.valueOf(args[2].trim()));
        create.setPrice(Long.valueOf(args[3].trim()));
        create.setContractCode(args[4].trim());
        if(args.length == 6) {
            create.setRemark(args[5].trim());
        }
        return create;
    } catch (Exception e) {
        e.fillInStackTrace();
        return null;
    }
}


@Override
public CommandResult execute(String[] args) {
    ContractCreate form = paramsData.get();
    if (null == form) {
        form = getContractCreate(args);
    }
    if (null == form) {
        return CommandResult.getFailed("parameter error.");
    }
    String sender = form.getSender();
```

```
    RpcClientResult res = CommandHelper.getPassword(sender, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String) res.getData();

    String contractCode = form.getContractCode();
    res = createContractArgs(contractCode);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    Object[] contractArgs = (Object[]) res.getData();

    Map<String, Object> parameters = new HashMap<>();
    parameters.put("sender", sender);
    parameters.put("gasLimit", form.getGasLimit());
    parameters.put("price", form.getPrice());
    parameters.put("password", password);
    parameters.put("remark", form.getRemark());
    parameters.put("contractCode", form.getContractCode());
    parameters.put("args", contractArgs);
    RpcClientResult result = restFul.post("/contract/create", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(result);
}

private RpcClientResult createContractArgs(String contractCode) {
    Map<String, Object> parameters = new HashMap<>();
    parameters.put("contractCode", contractCode);
    RpcClientResult result = restFul.post("/contract/constructor", parameters);
    if (result.isSuccess()) {
        RpcClientResult rpcClientResult = new RpcClientResult();
        rpcClientResult.setSuccess(true);
        try {
            Map<String, Object> map = (Map) result.getData();
            Map<String, Object> constructorMap = (Map) map.get("constructor");
            List<Object> argsList = (List) constructorMap.get("args");
            Object[] argsObj;
            if(argsList.size() > 0) {
                String argsListStr = JSONUtils.obj2PrettyJson(argsList);
```

```
                //
                String argsJson = getArgsJson(argsListStr);
                argsObj = parseArgsJson(argsJson);
            } else {
                argsObj = new Object[0];
            }
            rpcClientResult.setData(argsObj);
        } catch (Exception e) {
            e.printStackTrace();
            rpcClientResult.setSuccess(false);
        }
        return rpcClientResult;
    }
    return result;
}

public String getArgsJson(String constructor) {
    System.out.println("The arguments structure: ");
    System.out.println(constructor);
    String prompt = "Please enter the arguments you want to fill in according to the arguments structure(eg. \"a\",2,[\"c\",4],\"\",\"e\" or \"'a',2,['c',4],'','e'\").\nEnter the arguments:";
    System.out.print(prompt);
    ConsoleReader reader = null;
    try {
        reader = new ConsoleReader();
        String args = reader.readLine();
        if(StringUtils.isNotBlank(args)) {
            args = "[" + args + "]";
        }
        return args;
    } catch (IOException e) {
        return null;
    } finally {
        try {
            if (!reader.delete()) {
                reader.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
    private Object[] parseArgsJson(String argsJson) {
        if(StringUtils.isBlank(argsJson)) {
            return new Object[0];
        }
        try {
            List<Object> list = JSONUtils.json2pojo(argsJson, ArrayList.class);
            return list.toArray();
        } catch (Exception e) {
            e.fillInStackTrace();
            return null;
        }
    }
}

12:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\DeleteContractProcessor.java
 */

package io.nuls.contract.rpc.cmd;

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

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

/**
 * Delete contract
 * Created by wangkun23 on 2018/9/21.
 */
public class DeleteContractProcessor implements CommandProcessor {

    /**
     * rest client utils
     */
```

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

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

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

@Override
public String getCommandDescription() {
    return "deletecontract <sender> <address> --delete contract";
}

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if (length != 3) {
        return false;
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    String sender = args[1];
    if (StringUtils.isBlank(sender)) {
        return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
    }
    String password = CommandHelper.getPwd();
    /**
     * assemble request body JSON
     */
    Map<String, Object> parameters = new HashMap<>(3);
    parameters.put("sender", args[1]);
    parameters.put("contractAddress", args[2]);
```

```
        parameters.put("password", password);
        String url = "/contract/delete";
        RpcClientResult result = restFulUtils.post(url, parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}

13:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetContractAddressValidProcessor.java
 */

package io.nuls.contract.rpc.cmd;

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

/**
 * contract address is valid
 * Created by wangkun23 on 2018/9/20.
 */
public class GetContractAddressValidProcessor implements CommandProcessor {
    /**
     * rest utils
     */
    private RestFulUtils restFulUtils = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
```

```
        builder.newLine(getCommandDescription())
                .newLine("\t<address> contract address -required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getcontractaddressvalid <address> --contract address is valid";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 2) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        if (StringUtils.isBlank(address)) {
            return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
        }
        String url = "/contract/" + address;
        RpcClientResult result = restFulUtils.get(url, null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}

14:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetContractBalanceProcessor.java
 */

package io.nuls.contract.rpc.cmd;

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

import java.util.Map;

/**
 * query contract balance by contact address.
 * Created by wangkun23 on 2018/9/20.
 */
public class GetContractBalanceProcessor implements CommandProcessor {
    /**
     * rest utils
     */
    private RestFulUtils restFul = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "getcontractbalance <address> --get the contract balance by contract address";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 2) {
            return false;
```

```
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        if (StringUtils.isBlank(address)) {
            return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
        }
        String url = "/contract/balance/" + address;
        RpcClientResult result = restFul.get(url, null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        /**
         * assemble display data info
         */
        Map<String, Object> data = (Map) result.getData();
        /**
         * convert balance unit Na to NULS
         * the result balance unit is Na
         */
        data.put("balance", CommandHelper.naToNuls(data.get("balance")));
        return CommandResult.getResult(result);
    }
}
```

15:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\cmd\GetContractConstructorProcessor.java

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<contractCode> contract code -required");
        return builder.toString();
    }
```

```
    @Override
    public String getCommandDescription() {
        return "getcontractcontructor <contractCode> --get contract contructor from smart contract
program";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 2) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String code = args[1];
        if (StringUtils.isBlank(code)) {
            return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
        }
        /**
         * assemble request body JSON
         */
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("contractCode", code);
        String url = "/contract/constructor";
        RpcClientResult result = restFulUtils.post(url, parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}

16:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetContractInfoProcessor.java
 */

package io.nuls.contract.rpc.cmd;
```

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

/**
 * query contract information by contact address.
 * Created by wangkun23 on 2018/9/20.
 */
public class GetContractInfoProcessor implements CommandProcessor {
    /**
     * rest utils
     */
    private RestFulUtils restFul = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "getcontractinfo <address> --get the contract info by contract address";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 2) {
            return false;
        }
```

```
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        if (StringUtils.isBlank(address)) {
            return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
        }
        String url = "/contract/info/" + address;
        RpcClientResult result = restFul.get(url, null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        /**
         * assemble display data info
         */
        //Map<String, Object> map = (Map) result.getData();
        //Map<String, Object> dataMap = (Map) map.get("data");
        return CommandResult.getResult(result);
    }
}

17:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetContractResultProcessor.java
 */

package io.nuls.contract.rpc.cmd;

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

import java.util.Map;

/**
 * @desription:
```

```
 * @author: PierreLuo
 * @date: 2018/9/19
 */
public class GetContractResultProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @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 "getcontractresult <hash> --get the contract execute result of the transaction 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("/contract/result/" + hash, null);
```

```
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> map = (Map) result.getData();
        Map<String, Object> dataMap = (Map) map.get("data");
        if(dataMap != null) {
            dataMap.put("totalFee", CommandHelper.naToNuls(dataMap.get("totalFee")));
            dataMap.put("txSizeFee", CommandHelper.naToNuls(dataMap.get("txSizeFee")));
            dataMap.put("actualContractFee",
CommandHelper.naToNuls(dataMap.get("actualContractFee")));
            dataMap.put("refundFee", CommandHelper.naToNuls(dataMap.get("refundFee")));
            dataMap.put("value", CommandHelper.naToNuls(dataMap.get("value")));
            dataMap.put("price", CommandHelper.naToNuls(dataMap.get("price")));
            dataMap.put("balance", CommandHelper.naToNuls(dataMap.get("balance")));
        }

        result.setData(dataMap);
        return CommandResult.getResult(result);
    }

}

18:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetContractTxListProcessor.java
 */

package io.nuls.contract.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.HashMap;
import java.util.List;
import java.util.Map;
```

```
/**
 * query contract transactions by contract address
 * Created by wangkun23 on 2018/9/20.
 */
public class GetContractTxListProcessor implements CommandProcessor {

    private RestFulUtils restFulUtils = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "getcontracttxlist address <account> <pageNumber> <pageSize> --get the contract
transactions by address";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length < 4 || length > 5) {
            return false;
        }
        if (args.length == 4) {
            if (!StringUtils.isNumeric(args[2]) || !StringUtils.isNumeric(args[3])) {
                return false;
            }
        } else {
            if (!StringUtils.isNumeric(args[3]) || !StringUtils.isNumeric(args[4])) {
                return false;
```

```
            }
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        int pageNumber = 0;
        int pageSize = 0;
        if (args.length == 4) {
            pageNumber = Integer.parseInt(args[2]);
            pageSize = Integer.parseInt(args[3]);
        } else {
            pageNumber = Integer.parseInt(args[3]);
            pageSize = Integer.parseInt(args[4]);
        }
        String address = args[1];
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("pageNumber", pageNumber);
        parameters.put("pageSize", pageSize);

        /**
         * user type accountAddress argument.
         */
        if (args.length == 5) {
            parameters.put("accountAddress", args[2]);
        }
        String url = "/contract/tx/list/" + address;
        RpcClientResult result = restFulUtils.get(url, parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        /**
         * format amount and trx fee. 1NULS = 100000000Na
         */
        List<Map<String, Object>> list = (List<Map<String, Object>>) ((Map)
result.getData()).get("list");
        for(Map<String, Object> map : list){
            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", statusExplain((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);

            Map<String, Object> txDataMap = (Map) map.get("txData");
            if(txDataMap != null) {
                Map<String, Object> dataMap = (Map) txDataMap.get("data");
                if(dataMap != null) {
                    dataMap.put("value", CommandHelper.naToNuls(dataMap.get("value")));
                    dataMap.put("price", CommandHelper.naToNuls(dataMap.get("price")));
                }
            }

            Map<String, Object> contractResultMap = (Map) map.get("contractResult");
            if(contractResultMap != null) {
                contractResultMap.put("totalFee",
CommandHelper.naToNuls(contractResultMap.get("totalFee")));
                contractResultMap.put("txSizeFee",
CommandHelper.naToNuls(contractResultMap.get("txSizeFee")));
                contractResultMap.put("actualContractFee",
CommandHelper.naToNuls(contractResultMap.get("actualContractFee")));
                contractResultMap.put("refundFee",
CommandHelper.naToNuls(contractResultMap.get("refundFee")));
                contractResultMap.put("value",
CommandHelper.naToNuls(contractResultMap.get("value")));
                contractResultMap.put("price",
CommandHelper.naToNuls(contractResultMap.get("price")));
                contractResultMap.put("balance",
CommandHelper.naToNuls(contractResultMap.get("balance")));
            }
        }
        result.setData(list);
```

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

19:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetContractTxProcessor.java
 */

package io.nuls.contract.rpc.cmd;

import io.nuls.core.tools.date.DateUtil;
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.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/9/19
 */
public class GetContractTxProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @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 "getcontracttx <hash> --get the contract 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("/contract/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", statusExplain((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);

        Map<String, Object> txDataMap = (Map) map.get("txData");
        if(txDataMap != null) {
            Map<String, Object> dataMap = (Map) txDataMap.get("data");
            if(dataMap != null) {
                dataMap.put("value", CommandHelper.naToNuls(dataMap.get("value")));
                dataMap.put("price", CommandHelper.naToNuls(dataMap.get("price")));
            }
        }

        Map<String, Object> contractResultMap = (Map) map.get("contractResult");
        if(contractResultMap != null) {
            contractResultMap.put("totalFee",
CommandHelper.naToNuls(contractResultMap.get("totalFee")));
            contractResultMap.put("txSizeFee",
CommandHelper.naToNuls(contractResultMap.get("txSizeFee")));
            contractResultMap.put("actualContractFee",
CommandHelper.naToNuls(contractResultMap.get("actualContractFee")));
            contractResultMap.put("refundFee",
CommandHelper.naToNuls(contractResultMap.get("refundFee")));
            contractResultMap.put("value",
CommandHelper.naToNuls(contractResultMap.get("value")));
            contractResultMap.put("price",
CommandHelper.naToNuls(contractResultMap.get("price")));
            contractResultMap.put("balance",
CommandHelper.naToNuls(contractResultMap.get("balance")));
        }


        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";
    }
}

20:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetTokenBalanceProcessor.java
 */

package io.nuls.contract.rpc.cmd;

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

```
import java.util.Map;

import static io.nuls.contract.util.ContractUtil.valueOf;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/9/22
 */
public class GetTokenBalanceProcessor implements CommandProcessor {
    /**
     * rest utils
     */
    private RestFulUtils restFul = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "gettokenbalance <contractAddress> <address> --get the token balance";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 3) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
```

```
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String contractAddress = args[1].trim();
        String address = args[2].trim();
        String url = "/contract/balance/token/" + contractAddress + "/" + address;
        RpcClientResult result = restFul.get(url, null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> data = (Map) result.getData();
        data.put("amount", CommandHelper.tokenRecovery(valueOf(data.get("amount")), (Integer)
data.get("decimals")));
        data.put("status", statusExplain((Integer) data.get("status")));
        data.remove("decimals");
        data.remove("blockHeight");
        return CommandResult.getResult(result);
    }

    private String statusExplain(Integer status){
        if(status == 0){
            return "none";
        }
        if(status == 1){
            return"normal";
        }
        if(status == 2){
            return"termination";
        }
        return "unknown";
    }
}

21:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\GetWalletContractsProcessor.java
 */

package io.nuls.contract.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.RestFulUtils;

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

/**
 * get contracts by wallet address
 * Created by wangkun23 on 2018/9/20.
 */
public class GetWalletContractsProcessor implements CommandProcessor {

    private RestFulUtils restFulUtils = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "getwalletcontracts address <pageNumber> <pageSize> --get contracts by wallet
address";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
```

```
        if (length != 4) {
            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);

        String url = "/contract/wallet/list/" + address;
        RpcClientResult result = restFulUtils.get(url, parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}

22:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\TokenTransferProcessor.java
 */

package io.nuls.contract.rpc.cmd;

import io.nuls.contract.rpc.form.ContractCreate;
import io.nuls.contract.rpc.form.ContractTokenTransfer;
import io.nuls.core.tools.map.MapUtil;
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.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;
import javafx.beans.binding.BooleanBinding;

import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.HashMap;
import java.util.Map;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/9/22
 */
public class TokenTransferProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    private ThreadLocal<ContractTokenTransfer> paramsData = new ThreadLocal<>();

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<address>           source address - Required")
                .newLine("\t<toaddress>         receiving address - Required")
                .newLine("\t<contractAddress>   contract address    -Required")
                .newLine("\t<gasLimit>          gas limit    -Required")
                .newLine("\t<price>             price (Unit: Na/Gas)    -Required")
                .newLine("\t<amount>            amount, you can have up to [decimals of the contract]
valid digits after the decimal point - Required")
                .newLine("\t[remark]            remark -not required");
        return builder.toString();
    }

    @Override
```

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

    @Override
    public boolean argsValidate(String[] args) {
        boolean result;
        do {
            int length = args.length;
            if (length != 7 && length != 8) {
                result = false;
                break;
            }
            if (!CommandHelper.checkArgsIsNull(args)) {
                result = false;
                break;
            }

            // gasLimit
            if (!StringUtils.isNumeric(args[4])) {
                result = false;
                break;
            }
            // price
            if (!StringUtils.isNumeric(args[5])) {
                result = false;
                break;
            }
            // amount
            if (!StringUtils.isNumberGtZero(args[6])) {
                result = false;
                break;
            }
            ContractTokenTransfer form = getTokenTransferForm(args);
            if(null == form){
                result = false;
                break;
            }
            paramsData.set(form);
            result = StringUtils.isNotBlank(form.getToAddress());
            if (!result) {
```

```
            break;
        }
        result = true;
    } while (false);
    return result;
}

private ContractTokenTransfer getTokenTransferForm(String[] args) {
    ContractTokenTransfer transfer = null;
    try {
        transfer = new ContractTokenTransfer();
        transfer.setAddress(args[1].trim());
        transfer.setToAddress(args[2].trim());
        transfer.setContractAddress(args[3].trim());
        transfer.setGasLimit(Long.valueOf(args[4].trim()));
        transfer.setPrice(Long.valueOf(args[5].trim()));
        transfer.setAmount(args[6].trim());
        if(args.length == 8) {
            transfer.setRemark(args[7].trim());
        }
        return transfer;
    } catch (Exception e) {
        e.fillInStackTrace();
        return null;
    }
}

@Override
public CommandResult execute(String[] args) {
    ContractTokenTransfer form = paramsData.get();
    if (null == form) {
        form = getTokenTransferForm(args);
    }
    String address = form.getAddress();
    RpcClientResult res = CommandHelper.getPassword(address, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();

    String contractAddress = form.getContractAddress();
    String url = "/contract/" + contractAddress;
```

```
        RpcClientResult checkResult = restFul.get(url, null);
        if (checkResult.isFailed()) {
            return CommandResult.getFailed(checkResult);
        }
        Map<String, Object> data = (Map) checkResult.getData();
        Boolean isNrc20 = (Boolean) data.get("isNrc20");
        if(!isNrc20) {
            return CommandResult.getFailed("Non-NRC20 contract, can not transfer token.");
        }
        Integer decimals = (Integer) data.get("decimals");
        BigDecimal amountBigD = new
BigDecimal(form.getAmount()).multiply(BigDecimal.TEN.pow(decimals));
        try {
            BigInteger amountBigI = amountBigD.toBigIntegerExact();
            form.setAmount(amountBigI.toString());
        } catch(Exception e) {
            return CommandResult.getFailed("Illegal amount, you can have up to " + decimals + " valid
digits after the decimal point.");
        }

        Map<String, Object> parameters = new HashMap<>();
        parameters.put("address", form.getAddress());
        parameters.put("toAddress", form.getToAddress());
        parameters.put("contractAddress", form.getContractAddress());
        parameters.put("gasLimit", form.getGasLimit());
        parameters.put("price", form.getPrice());
        parameters.put("password", password);
        parameters.put("amount", form.getAmount());
        parameters.put("remark", form.getRemark());
        RpcClientResult result = restFul.post("/contract/token/transfer", parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> resultMap = MapUtil.createLinkedHashMap(2);
        resultMap.put("txHash", result.getData());
        result.setData(resultMap);
        return CommandResult.getResult(result);
    }
}
```

23:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\cmd\TransferToContractProcessor.java

```
 */

package io.nuls.contract.rpc.cmd;

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

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

/**
 * Transfer to contract address
 * Created by wangkun23 on 2018/9/25.
 */
public class TransferToContractProcessor implements CommandProcessor {
    /**
     * rest utils
     */
    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<address> address -required")
                .newLine("\t<toAddress> toAddress -required")
                .newLine("\t<gasLimit> gasLimit -required")
                .newLine("\t<price> contract price -required")
                .newLine("\t<amount> transfer amount -required")
                .newLine("\t[remark] remark not -required");
        return builder.toString();
```

```
    }

    @Override
    public String getCommandDescription() {
        return "transfertocontract <address> <toAddress> <gasLimit> <price> <amount> [remark] --
create transfer to contract address";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length <6) {
            return false;
        }
        if (length >7) {
            return false;
        }
        return true;
    }

    /**
     * {
     * "address": "Nsdv1Hbu4TokdgbXreypXmVttYKdPT1g",
     * "toAddress": "NseDqffhWEB52a9cWfiyEhiP3wPGcjcJ",
     * "gasLimit": 800000,
     * "price": 27,
     * "password": "nuls123456",
     * "amount": 10000000,
     * "remark": ""
     * }
     *
     * @param args
     * @return
     */
    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        if (StringUtils.isBlank(address)) {
            return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
        }
        RpcClientResult res = CommandHelper.getPassword(address, restFul);
        if(!res.isSuccess()){
```

```
            return CommandResult.getFailed(res);
        }
        String password = (String) res.getData();
        /**
         * assemble request body JSON
         */
        Map<String, Object> parameters = new HashMap<>(7);
        parameters.put("address", address);
        parameters.put("toAddress", args[2]);
        parameters.put("gasLimit", args[3]);
        parameters.put("price", args[4]);
        parameters.put("amount", args[5]);
        if (args.length==7){
            parameters.put("remark",args[6]);
        }
        //password
        parameters.put("password", password);

        String url = "/contract/transfer";
        RpcClientResult result = restFul.post(url, parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}

24:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\cmd\ViewContractProcessor.java
 */

package io.nuls.contract.rpc.cmd;

import io.nuls.contract.rpc.form.ContractViewCall;
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;

import static io.nuls.kernel.utils.CommandHelper.getContractCallArgsJson;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/9/19
 */
public class ViewContractProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

    private ThreadLocal<ContractViewCall> paramsData = new ThreadLocal<>();

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<contractAddress>   contract address    -required")
                .newLine("\t<methodName>        the method to call    -required")
                .newLine("\t[-d methodDesc]        the method description    -not required");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "viewcontract <contractAddress> <methodName> [-d methodDesc] --view contract";
    }

    @Override
    public boolean argsValidate(String[] args) {
        boolean result;
        do {
            int length = args.length;
            if (length != 3 && length != 5) {
                result = false;
```

```
            break;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            result = false;
        break;
        }

        ContractViewCall form = getContractViewCall(args);
        if(null == form){
            result = false;
            break;
        }
        paramsData.set(form);

        result = true;
    } while (false);
    return result;
}

private ContractViewCall getContractViewCall(String[] args) {
    ContractViewCall call;
    try {
        call = new ContractViewCall();
        call.setContractAddress(args[1].trim());
        call.setMethodName(args[2].trim());

        if(args.length == 5) {
            String argType = args[3].trim();
            if(argType.equals("-d")) {
                call.setMethodDesc(args[4].trim());
            } else {
                return null;
            }
        }
        return call;
    } catch (Exception e) {
        e.fillInStackTrace();
        return null;
    }
}
```

```
    @Override
    public CommandResult execute(String[] args) {
        ContractViewCall form = paramsData.get();
        if (null == form) {
            form = getContractViewCall(args);
        }
        if (null == form) {
            return CommandResult.getFailed("parameter error.");
        }
        RpcClientResult res = getContractCallArgsJson();
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        Object[] contractArgs = (Object[]) res.getData();

        Map<String, Object> parameters = new HashMap<>();
        parameters.put("contractAddress", form.getContractAddress());
        parameters.put("methodName", form.getMethodName());
        parameters.put("methodDesc", form.getMethodDesc());
        parameters.put("args", contractArgs);
        RpcClientResult result = restFul.post("/contract/view", parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }


}

25:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractAddressBase.java
 */
package io.nuls.contract.rpc.form;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/15
```

```
 */
@ApiModel(value = "")
public class ContractAddressBase {

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

    public String getAccountAddress() {
        return accountAddress;
    }

    public void setAccountAddress(String accountAddress) {
        this.accountAddress = accountAddress;
    }

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

}
```

26:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\form\ContractBase.java

```
 */
package io.nuls.contract.rpc.form;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/4/21
 */
@ApiModel(value = "")
public class ContractBase {
```

```
@ApiModelProperty(name = "sender", value = "", required = true)
private String sender;
@ApiModelProperty(name = "gasLimit", value = "gas", required = true)
private long gasLimit;
@ApiModelProperty(name = "price", value = "", required = true)
private long price;
@ApiModelProperty(name = "password", value = "", required = true)
private String password;
@ApiModelProperty(name = "remark", value = "", required = false)
private String remark;

public String getSender() {
    return sender;
}

public void setSender(String sender) {
    this.sender = sender;
}

public long getGasLimit() {
    return gasLimit;
}

public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

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;
    }
}

27:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractCall.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "")
public class ContractCall extends ContractBase {

    @ApiModelProperty(name = "contractAddress", value = "", required = true)
    private String contractAddress;
    @ApiModelProperty(name = "value", value = "", required = false)
    private long value;
    @ApiModelProperty(name = "methodName", value = "", required = true)
    private String methodName;
    @ApiModelProperty(name = "methodDesc", value = "", required = false)
    private String methodDesc;
    @ApiModelProperty(name = "args", value = "", required = false)
    private Object[] args;

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    public long getValue() {
```

```
        return value;
    }

    public void setValue(long value) {
        this.value = value;
    }

    public String getMethodName() {
        return methodName;
    }

    public void setMethodName(String methodName) {
        this.methodName = methodName;
    }

    public String getMethodDesc() {
        return methodDesc;
    }

    public void setMethodDesc(String methodDesc) {
        this.methodDesc = methodDesc;
    }

    public Object[] getArgs() {
        return args;
    }

    public String[][] getArgs(String[] types) {
        return ContractUtil.twoDimensionalArray(args, types);
    }

    public void setArgs(Object[] args) {
        this.args = args;
    }

}

28:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractCode.java
 */
package io.nuls.contract.rpc.form;
```

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/4/20
 */
@ApiModel(value = "")
public class ContractCode {

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

    public String getContractCode() {
        return contractCode;
    }

    public void setContractCode(String contractCode) {
        this.contractCode = contractCode;
    }
}

29:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractCollection.java
 */
package io.nuls.contract.rpc.form;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/15
 */
@ApiModel(value = "")
public class ContractCollection extends ContractAddressBase {

    @ApiModelProperty(name = "remarkName", value = "", required = false)
    private String remarkName;
```

```
    public String getRemarkName() {
        return remarkName;
    }

    public void setRemarkName(String remarkName) {
        this.remarkName = remarkName;
    }
}

30:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractCreate.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/4/20
 */
@ApiModel(value = "")
public class ContractCreate extends ContractBase {

    @ApiModelProperty(name = "contractCode", value = "(Hex)", required = true)
    private String contractCode;
    @ApiModelProperty(name = "args", value = "", required = false)
    private Object[] args;

    public String getContractCode() {
        return contractCode;
    }

    public void setContractCode(String contractCode) {
        this.contractCode = contractCode;
    }

    public Object[] getArgs() {
        return args;
    }
```

```
    public String[][] getArgs(String[] types) {
        return ContractUtil.twoDimensionalArray(args, types);
    }

    public void setArgs(Object[] args) {
        this.args = args;
    }
}

31:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractCreateFile.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.io.InputStream;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/4/20
 */
@ApiModel(value = "-jar")
public class ContractCreateFile extends ContractBase {

    @ApiModelProperty(name = "contractCode", value = "", required = true)
    private InputStream contractCode;
    @ApiModelProperty(name = "args", value = "", required = false)
    private Object[] args;

    public InputStream getContractCode() {
        return contractCode;
    }

    public void setContractCode(InputStream contractCode) {
        this.contractCode = contractCode;
    }
```

```
    public Object[] getArgs() {
        return args;
    }

    public String[][] getArgs(String[] types) {
        return ContractUtil.twoDimensionalArray(args, types);
    }

    public void setArgs(Object[] args) {
        this.args = args;
    }
}

32:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractDelete.java
 */
package io.nuls.contract.rpc.form;

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

@ApiModel(value = "")
public class ContractDelete {

    @ApiModelProperty(name = "sender", value = "", required = true)
    private String sender;
    @ApiModelProperty(name = "contractAddress", value = "", required = true)
    private String contractAddress;
    @ApiModelProperty(name = "password", value = "", required = true)
    private String password;
    @ApiModelProperty(name = "remark", value = "", required = false)
    private String remark;

    public String getSender() {
        return sender;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }

    public String getContractAddress() {
```

```
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    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;
    }
}
```

33:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\form\ContractTokenTransfer.java

```
    private String amount;
    @ApiModelProperty(name = "remark", value = "", required = false)
    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 getContractAddress() {
    return contractAddress;
}

public void setContractAddress(String contractAddress) {
    this.contractAddress = contractAddress;
}

public long getGasLimit() {
    return gasLimit;
}

public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getAmount() {
    return amount;
}

public void setAmount(String amount) {
    this.amount = amount;
```

```
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }
}

34:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractTransfer.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "")
public class ContractTransfer {

    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;
    @ApiModelProperty(name = "toAddress", value = "()", required = true)
    private String toAddress;
    @ApiModelProperty(name = "gasLimit", value = "gas", required = true)
    private long gasLimit;
    @ApiModelProperty(name = "price", value = "", required = true)
    private long price;
    @ApiModelProperty(name = "password", value = "", required = false)
    private String password;
    @ApiModelProperty(name = "amount", value = "", required = true)
    private long amount;
    @ApiModelProperty(name = "remark", value = "", required = false)
    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 getGasLimit() {
    return gasLimit;
}

public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

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;
    }
}

35:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractTransferFee.java
 */
package io.nuls.contract.rpc.form;

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

@ApiModel(value = "")
public class ContractTransferFee {

    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;
    @ApiModelProperty(name = "toAddress", value = "()", required = true)
    private String toAddress;
    @ApiModelProperty(name = "gasLimit", value = "gas", required = true)
    private long gasLimit;
    @ApiModelProperty(name = "price", value = "", required = true)
    private long price;
    @ApiModelProperty(name = "amount", value = "", required = true)
    private long amount;
    @ApiModelProperty(name = "remark", value = "", required = false)
    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 getGasLimit() {
        return gasLimit;
    }

    public void setGasLimit(long gasLimit) {
        this.gasLimit = gasLimit;
    }

    public long getPrice() {
        return price;
    }

    public void setPrice(long price) {
        this.price = price;
    }

    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;
    }
}
```

```
36:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ContractViewCall.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "")
public class ContractViewCall {

    @ApiModelProperty(name = "contractAddress", value = "", required = true)
    private String contractAddress;
    @ApiModelProperty(name = "methodName", value = "", required = true)
    private String methodName;
    @ApiModelProperty(name = "methodDesc", value = "", required = false)
    private String methodDesc;
    @ApiModelProperty(name = "args", value = "", required = false)
    private Object[] args;

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    public String getMethodName() {
        return methodName;
    }

    public void setMethodName(String methodName) {
        this.methodName = methodName;
    }

    public String getMethodDesc() {
        return methodDesc;
    }

    public void setMethodDesc(String methodDesc) {
```

```
        this.methodDesc = methodDesc;
    }

    public Object[] getArgs() {
        return args;
    }

    public String[][] getArgs(String[] types) {
        return ContractUtil.twoDimensionalArray(args, types);
    }

    public void setArgs(Object[] args) {
        this.args = args;
    }
}

37:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ImputedGasContractCall.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/7/18
 */
@ApiModel(value = "Gas")
public class ImputedGasContractCall {

    @ApiModelProperty(name = "sender", value = "", required = true)
    private String sender;
    @ApiModelProperty(name = "contractAddress", value = "", required = true)
    private String contractAddress;
    @ApiModelProperty(name = "value", value = "", required = false)
    private long value;
    @ApiModelProperty(name = "methodName", value = "", required = true)
    private String methodName;
    @ApiModelProperty(name = "methodDesc", value = "", required = false)
```

```
private String methodDesc;
@ApiModelProperty(name = "price", value = "", required = true)
private long price;
@ApiModelProperty(name = "args", value = "", required = false)
private Object[] args;

public String getSender() {
    return sender;
}

public void setSender(String sender) {
    this.sender = sender;
}

public String getContractAddress() {
    return contractAddress;
}

public void setContractAddress(String contractAddress) {
    this.contractAddress = contractAddress;
}

public long getValue() {
    return value;
}

public void setValue(long value) {
    this.value = value;
}

public String getMethodName() {
    return methodName;
}

public void setMethodName(String methodName) {
    this.methodName = methodName;
}

public String getMethodDesc() {
    return methodDesc;
}
```

```
    public void setMethodDesc(String methodDesc) {
        this.methodDesc = methodDesc;
    }

    public long getPrice() {
        return price;
    }

    public void setPrice(long price) {
        this.price = price;
    }

    public Object[] getArgs() {
        return args;
    }

    public String[][] getArgs(String[] types) {
        return ContractUtil.twoDimensionalArray(args, types);
    }

    public void setArgs(Object[] args) {
        this.args = args;
    }
}

38:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ImputedGasContractCreate.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/4/20
 */
@ApiModel(value = "Gas")
public class ImputedGasContractCreate {
```

```
@ApiModelProperty(name = "sender", value = "", required = true)
private String sender;
@ApiModelProperty(name = "price", value = "", required = true)
private long price;
@ApiModelProperty(name = "contractCode", value = "(Hex)", required = true)
private String contractCode;
@ApiModelProperty(name = "args", value = "", required = false)
private Object[] args;

public String getSender() {
    return sender;
}

public void setSender(String sender) {
    this.sender = sender;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

public String getContractCode() {
    return contractCode;
}

public void setContractCode(String contractCode) {
    this.contractCode = contractCode;
}

public Object[] getArgs() {
    return args;
}

public String[][] getArgs(String[] types) {
    return ContractUtil.twoDimensionalArray(args, types);
}

public void setArgs(Object[] args) {
```

```
        this.args = args;
    }
}

39:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\ImputedPrice.java
 */
package io.nuls.contract.rpc.form;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/7/18
 */
@ApiModel(value = "price")
public class ImputedPrice {

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

    public String getSender() {
        return sender;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }

}

40:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\form\PreContractCreate.java
 */
package io.nuls.contract.rpc.form;

import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/4/20
 */
@ApiModel(value = "")
public class PreContractCreate {

    @ApiModelProperty(name = "sender", value = "", required = true)
    private String sender;
    @ApiModelProperty(name = "gasLimit", value = "gas", required = true)
    private long gasLimit;
    @ApiModelProperty(name = "price", value = "", required = true)
    private long price;
    @ApiModelProperty(name = "contractCode", value = "(Hex)", required = true)
    private String contractCode;
    @ApiModelProperty(name = "args", value = "", required = false)
    private Object[] args;
    @ApiModelProperty(name = "remark", value = "", required = false)
    private String remark;

    public String getContractCode() {
        return contractCode;
    }

    public void setContractCode(String contractCode) {
        this.contractCode = contractCode;
    }

    public Object[] getArgs() {
        return args;
    }

    public String[][] getArgs(String[] types) {
        return ContractUtil.twoDimensionalArray(args, types);
    }

    public void setArgs(Object[] args) {
        this.args = args;
    }
    public String getSender() {
        return sender;
```

```
    }

    public void setSender(String sender) {
        this.sender = sender;
    }

    public long getGasLimit() {
        return gasLimit;
    }

    public void setGasLimit(long gasLimit) {
        this.gasLimit = gasLimit;
    }

    public long getPrice() {
        return price;
    }

    public void setPrice(long price) {
        this.price = price;
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }
}
```

41:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\CallContractDataDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.entity.txdata.CallContractData;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.utils.AddressTool;

/**
 * @author: PierreLuo
 */
```

```
public class CallContractDataDto {

    private String sender;
    private String contractAddress;
    private long value;
    private long gasLimit;
    private long price;
    private String methodName;
    private String methodDesc;
    private byte argsCount;
    private String[][] args;

    public CallContractDataDto(CallContractData call) {
        this.sender = AddressTool.getStringAddressByBytes(call.getSender());
        this.contractAddress = AddressTool.getStringAddressByBytes(call.getContractAddress());
        this.value = call.getValue();
        this.gasLimit = call.getGasLimit();
        this.price = call.getPrice();
        this.methodName = call.getMethodName();
        this.methodDesc = call.getMethodDesc();
        this.argsCount = call.getArgsCount();
        this.args = call.getArgs();
    }

    public String getSender() {
        return sender;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    public long getValue() {
        return value;
```

```
}

public void setValue(long value) {
    this.value = value;
}

public long getGasLimit() {
    return gasLimit;
}

public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

public String getMethodName() {
    return methodName;
}

public void setMethodName(String methodName) {
    this.methodName = methodName;
}

public String getMethodDesc() {
    return methodDesc;
}

public void setMethodDesc(String methodDesc) {
    this.methodDesc = methodDesc;
}

public byte getArgsCount() {
    return argsCount;
}
```

```
    public void setArgsCount(byte argsCount) {
        this.argsCount = argsCount;
    }

    public String[][] getArgs() {
        return args;
    }

    public void setArgs(String[][] args) {
        this.args = args;
    }
}
```

42:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractAccountUtxoDto.java

```
package io.nuls.contract.rpc.model;

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

import java.util.List;

/**
 * @author: PierreLuo
 */
@ApiModel(value = "AccountUtxoDtoJSON")
public class ContractAccountUtxoDto {

    @ApiModelProperty(name = "utxoDtoList", value = "")
    private List<ContractUtxoDto> utxoDtoList;

    public List<ContractUtxoDto> getUtxoDtoList() {
        return utxoDtoList;
    }

    public void setUtxoDtoList(List<ContractUtxoDto> utxoDtoList) {
        this.utxoDtoList = utxoDtoList;
    }
}
```

43:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractAddressDto.java

```
 */
package io.nuls.contract.rpc.model;

import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.po.ContractCollectionInfoPo;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/15
 */
@ApiModel(value = "ContractAddressDtoJSON")
public class ContractAddressDto {

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

    @ApiModelProperty(name = "isCreate", value = "")
    private boolean isCreate;

    @ApiModelProperty(name = "createTime", value = "")
    private long createTime;

    @ApiModelProperty(name = "height", value = "")
    private long height;

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

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

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

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

```
    public ContractAddressDto() {
    }

    public ContractAddressDto(ContractCollectionInfoPo po, String address, boolean isCreate, int
status) {
        this.contractAddress = po.getContractAddress();
        this.createTime = po.getCreateTime();
        this.remarkName = po.getCollectorMap().get(address);
        this.isCreate = isCreate;
        this.height = po.getBlockHeight();
        this.status = status;
        long bestBlockHeight = NulsContext.getInstance().getBestHeight();
        if (this.height > 0) {
            this.confirmCount = bestBlockHeight - this.height;
            if(this.confirmCount == 0) {
                this.status = 0;
            } else if(this.confirmCount < 7) {
                this.status = 4;
            }
        } else {
            this.confirmCount = 0L;
        }
    }

    public ContractAddressDto(ContractAddressInfoPo po, boolean isCreate, int status) {
        this.contractAddress = AddressTool.getStringAddressByBytes(po.getContractAddress());
        this.createTime = po.getCreateTime();
        this.isCreate = isCreate;
        this.height = po.getBlockHeight();
        this.status = status;
        long bestBlockHeight = NulsContext.getInstance().getBestHeight();
        if (this.height > 0) {
            this.confirmCount = bestBlockHeight - this.height;
            if(this.confirmCount == 0) {
                this.status = 0;
            } else if(this.confirmCount < 7) {
                this.status = 4;
            }
        } else {
            this.confirmCount = 0L;
        }
    }
```

```
public String getContractAddress() {
    return contractAddress;
}

public void setContractAddress(String contractAddress) {
    this.contractAddress = contractAddress;
}

public long getCreateTime() {
    return createTime;
}

public void setCreateTime(long createTime) {
    this.createTime = createTime;
}

public long getHeight() {
    return height;
}

public void setHeight(long height) {
    this.height = height;
}

public long getConfirmCount() {
    return confirmCount;
}

public void setConfirmCount(long confirmCount) {
    this.confirmCount = confirmCount;
}

public String getRemarkName() {
    return remarkName;
}

public void setRemarkName(String remarkName) {
    this.remarkName = remarkName;
}

public boolean isCreate() {
```

```
        return isCreate;
    }

    public void setCreate(boolean create) {
        isCreate = create;
    }

    public int getStatus() {
        return status;
    }

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

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }
}
```

44:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractCollectionDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.storage.po.ContractCollectionInfoPo;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.util.List;

/**
 * @author: PierreLuo
 */
@ApiModel(value = "ContractCollectionDtoJSON")
public class ContractCollectionDto {

    @ApiModelProperty(name = "list", value = "")
    private List<ContractAddressDto> list;
```

```
    public List<ContractAddressDto> getList() {
        return list;
    }

    public void setList(List<ContractAddressDto> list) {
        this.list = list;
    }
}
```

45:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractResultDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.dto.ContractTransfer;
import io.nuls.contract.entity.txdata.ContractData;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.calc.LongUtils;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

/**
 * @author: PierreLuo
 */
@ApiModel(value = "contractResultDtoJSON")
public class ContractResultDto {

    @ApiModelProperty(name = "success", value = "")
    private boolean success;

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

```
private String errorMessage;

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

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

@ApiModelProperty(name = "gasUsed", value = "GasLimit")
private long gasLimit;

@ApiModelProperty(name = "gasUsed", value = "Gas")
private long gasUsed;

@ApiModelProperty(name = "price", value = "")
private long price;

@ApiModelProperty(name = "totalFee", value = "")
private BigInteger totalFee;

@ApiModelProperty(name = "txSizeFee", value = "")
private BigInteger txSizeFee;

@ApiModelProperty(name = "actualContractFee", value = "")
private BigInteger actualContractFee;

@ApiModelProperty(name = "refundFee", value = "")
private BigInteger refundFee;

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

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

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

@ApiModelProperty(name = "balance", value = "")
private BigInteger balance;

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

```
private BigInteger nonce;

@ApiModelProperty(name = "transfers", value = "()")
private List<ContractTransferDto> transfers;

@ApiModelProperty(name = "events", value = "")
private List<String> events;

@ApiModelProperty(name = "tokenTransfers", value = "")
private List<ContractTokenTransferDto> tokenTransfers;

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

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

@ApiModelProperty(name = "decimals", value = "token")
private long decimals;

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

public ContractResultDto() {}

public ContractResultDto(ContractResult result, Transaction tx) {
    ContractData contractData = (ContractData) tx.getTxData();
    this.totalFee = BigInteger.valueOf(tx.getFee().getValue());
    this.gasLimit = contractData.getGasLimit();
    this.gasUsed = result.getGasUsed();
    this.price = result.getPrice();
    this.actualContractFee = BigInteger.valueOf(LongUtils.mul(this.gasUsed, this.price));
    BigInteger contractFee = BigInteger.valueOf(LongUtils.mul(gasLimit, price));
    this.refundFee = contractFee.subtract(this.actualContractFee);
    this.txSizeFee = this.totalFee.subtract(contractFee);
    this.contractAddress = AddressTool.getStringAddressByBytes(result.getContractAddress());
    this.result = result.getResult();
    this.stateRoot = (result.getStateRoot() != null ? Hex.encode(result.getStateRoot()) : null);
    this.value = result.getValue();
    this.success = result.isSuccess();
    this.errorMessage = result.getErrorMessage();
    this.stackTrace = result.getStackTrace();
```

```
        this.balance = result.getBalance();
        this.nonce = result.getNonce();
        this.setOrginTransfers(result.getTransfers());
        this.events = result.getEvents();
        this.remark = result.getRemark();
        if(result.isSuccess()) {
            this.makeTokenTransfers(result.getEvents());
        }
    }

    public ContractResultDto(ContractResult result, Transaction tx, ContractAddressInfoPo po) {
        this(result, tx);
        if(result.isNrc20()) {
            this.name = po.getNrc20TokenName();
            this.symbol = po.getNrc20TokenSymbol();
            this.decimals = po.getDecimals();
        }
    }

    public ContractResultDto(ContractResult contractExecuteResult, Transaction tx,
ContractAddressInfoPo po, ContractTokenTransferInfoPo transferInfoPo) {
        this(contractExecuteResult, tx, po);
        if(transferInfoPo != null) {
            this.tokenTransfers = new ArrayList<>();
            this.tokenTransfers.add(new ContractTokenTransferDto(transferInfoPo));
        }
    }

    public List<ContractTokenTransferDto> getTokenTransfers() {
        return tokenTransfers == null ? new ArrayList<>() : tokenTransfers;
    }

    public void setTokenTransfers(List<ContractTokenTransferDto> tokenTransfers) {
        this.tokenTransfers = tokenTransfers;
    }

    private void makeTokenTransfers(List<String> tokenTransferEvents) {
        List<ContractTokenTransferDto> result = new ArrayList<>();
        if(tokenTransferEvents != null && tokenTransferEvents.size() > 0) {
            ContractTokenTransferInfoPo po;
            for(String event : tokenTransferEvents) {
                po = ContractUtil.convertJsonToTokenTransferInfoPo(event);
```

```
            if(po != null) {
                result.add(new ContractTokenTransferDto(po));
            }
        }
    }
    this.tokenTransfers = result;
}

public String getContractAddress() {
    return contractAddress;
}

public void setContractAddress(String contractAddress) {
    this.contractAddress = contractAddress;
}

public String getResult() {
    return result;
}

public void setResult(String result) {
    this.result = result;
}

public long getGasUsed() {
    return gasUsed;
}

public void setGasUsed(long gasUsed) {
    this.gasUsed = gasUsed;
}

public String getStateRoot() {
    return stateRoot;
}

public void setStateRoot(String stateRoot) {
    this.stateRoot = stateRoot;
}

public long getValue() {
    return value;
```

```
}

public void setValue(long value) {
    this.value = value;
}

public boolean isSuccess() {
    return success;
}

public void setSuccess(boolean success) {
    this.success = success;
}

public String getErrorMessage() {
    return errorMessage;
}

public void setErrorMessage(String errorMessage) {
    this.errorMessage = errorMessage;
}

public String getStackTrace() {
    return stackTrace;
}

public void setStackTrace(String stackTrace) {
    this.stackTrace = stackTrace;
}

public BigInteger getBalance() {
    return balance;
}

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

public BigInteger getNonce() {
    return nonce;
}
```

```
public void setNonce(BigInteger nonce) {
    this.nonce = nonce;
}

public List<ContractTransferDto> getTransfers() {
    return transfers == null ? new ArrayList<>() : transfers;
}

public void setTransfers(List<ContractTransferDto> transfers) {
    this.transfers = transfers;
}

public void setOrginTransfers(List<ContractTransfer> transfers) {
    List<ContractTransferDto> list = new LinkedList<>();
    this.transfers = list;
    if(transfers == null || transfers.size() == 0) {
        return;
    }
    for(ContractTransfer transfer : transfers) {
        list.add(new ContractTransferDto(transfer));
    }
}

public List<String> getEvents() {
    return events;
}

public void setEvents(List<String> events) {
    this.events = events;
}

public String getRemark() {
    return remark;
}

public void setRemark(String remark) {
    this.remark = remark;
}

public long getGasLimit() {
    return gasLimit;
}
```

```
public void setGasLimit(long gasLimit) {
    this.gasLimit = gasLimit;
}

public BigInteger getTotalFee() {
    return totalFee;
}

public void setTotalFee(BigInteger totalFee) {
    this.totalFee = totalFee;
}

public BigInteger getRefundFee() {
    return refundFee;
}

public void setRefundFee(BigInteger refundFee) {
    this.refundFee = refundFee;
}

public long getPrice() {
    return price;
}

public void setPrice(long price) {
    this.price = price;
}

public BigInteger getActualContractFee() {
    return actualContractFee;
}

public void setActualContractFee(BigInteger actualContractFee) {
    this.actualContractFee = actualContractFee;
}

public BigInteger getTxSizeFee() {
    return txSizeFee;
}

public void setTxSizeFee(BigInteger txSizeFee) {
```

```
        this.txSizeFee = txSizeFee;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSymbol() {
        return symbol;
    }

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

    public long getDecimals() {
        return decimals;
    }

    public void setDecimals(long decimals) {
        this.decimals = decimals;
    }
}
```

46:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractTokenInfoDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.dto.ContractTokenInfo;
import io.nuls.contract.util.ContractUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.math.BigInteger;

/**
 * @desription:
 * @author: PierreLuo
```

```
 * @date: 2018/8/19
 */
@ApiModel(value = "token")
public class ContractTokenInfoDto {

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

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

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

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

    @ApiModelProperty(name = "decimals", value = "token")
    private long decimals;

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

    @ApiModelProperty(name = "status", value = "(0-, 1-, 2-)")
    private int status;

    public ContractTokenInfoDto() {
    }

    public ContractTokenInfoDto(ContractTokenInfo info) {
        this.contractAddress = info.getContractAddress();
        this.name = info.getName();
        this.symbol = info.getSymbol();
        this.amount = ContractUtil.bigInteger2String(info.getAmount());
        this.decimals = info.getDecimals();
        this.blockHeight = info.getBlockHeight();
        this.status = info.getStatus();
    }

    public String getContractAddress() {
        return contractAddress;
    }
```

```
public void setContractAddress(String contractAddress) {
    this.contractAddress = contractAddress;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getAmount() {
    return amount;
}

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

public String getSymbol() {
    return symbol;
}

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

public long getDecimals() {
    return decimals;
}

public void setDecimals(long decimals) {
    this.decimals = decimals;
}

public long getBlockHeight() {
    return blockHeight;
}

public void setBlockHeight(long blockHeight) {
```

```
        this.blockHeight = blockHeight;
    }

    public int getStatus() {
        return status;
    }

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

47:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractTokenTransferDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.util.ContractUtil;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.math.BigInteger;

/**
 * @author: PierreLuo
 */
@ApiModel(value = "ContractTokenTransferDtoJSON")
public class ContractTokenTransferDto {

    @ApiModelProperty(name = "contractAddress", value = "")
    private String contractAddress;
    @ApiModelProperty(name = "from", value = "")
    private String from;
    @ApiModelProperty(name = "to", value = "")
    private String to;
    @ApiModelProperty(name = "value", value = "")
    private String value;
    @ApiModelProperty(name = "name", value = "token")
    private String name;
    @ApiModelProperty(name = "symbol", value = "token")
```

```
private String symbol;
@ApiModelProperty(name = "decimals", value = "token")
private long decimals;

public ContractTokenTransferDto(ContractTokenTransferInfoPo po) {
    this.contractAddress = po.getContractAddress();
    if(po.getFrom() != null) {
        this.from = AddressTool.getStringAddressByBytes(po.getFrom());
    }
    if(po.getTo() != null) {
        this.to = AddressTool.getStringAddressByBytes(po.getTo());
    }
    this.value = ContractUtil.bigInteger2String(po.getValue());
    this.name = po.getName();
    this.symbol = po.getSymbol();
    this.decimals = po.getDecimals();
}

public void setNrc20Info(ContractAddressInfoPo po) {
    this.name = po.getNrc20TokenName();
    this.symbol = po.getNrc20TokenSymbol();
    this.decimals = po.getDecimals();
}

public String getContractAddress() {
    return contractAddress;
}

public void setContractAddress(String contractAddress) {
    this.contractAddress = contractAddress;
}

public String getFrom() {
    return from;
}

public void setFrom(String from) {
    this.from = from;
}

public String getTo() {
    return to;
```

```
    }

    public void setTo(String to) {
        this.to = to;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSymbol() {
        return symbol;
    }

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

    public long getDecimals() {
        return decimals;
    }

    public void setDecimals(long decimals) {
        this.decimals = decimals;
    }
}

48:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\model\ContractTransactionDto.java
 */
```

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.entity.txdata.CallContractData;
import io.nuls.contract.entity.txdata.ContractTransferData;
import io.nuls.contract.entity.txdata.CreateContractData;
import io.nuls.contract.entity.txdata.DeleteContractData;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.map.MapUtil;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.NulsConstant;
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.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

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

/**
 * @author: PierreLuo
 */
@ApiModel(value = "ContractTransactionDtoJSON")
public class ContractTransactionDto {

    @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 = "txData", value = "")
protected Map<String, Object> txData;

@ApiModelProperty(name = "contractResult", value = "")
protected ContractResultDto contractResult;

public ContractTransactionDto(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();
    this.txData = makeTxData(tx);

    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;
}

private Map<String,Object> makeTxData(Transaction tx) {
    Map<String,Object> result = new HashMap<>();
    TransactionLogicData txData = tx.getTxData();
```

```
    if(type == ContractConstant.TX_TYPE_CREATE_CONTRACT) {
        CreateContractData create = (CreateContractData) txData;
        result.put("data", new CreateContractDataDto(create));
    } else if(type == ContractConstant.TX_TYPE_CALL_CONTRACT) {
        CallContractData call = (CallContractData) txData;
        result.put("data", new CallContractDataDto(call));
    } else if(type == ContractConstant.TX_TYPE_DELETE_CONTRACT) {
        DeleteContractData delete = (DeleteContractData) txData;
        result.put("data", new DeleteContractDataDto(delete));
    } else if(type == ContractConstant.TX_TYPE_CONTRACT_TRANSFER) {
        ContractTransferData transfer = (ContractTransferData) txData;
        result.put("data", new ContractTransferDataDto(transfer));
    } else if(type == NulsConstant.TX_TYPE_COINBASE) {
        Map<String, String> map = MapUtil.createLinkedHashMap(1);
        map.put("sender", EMPTY);
        result.put("data", map);
    }
    return result;
}

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 Map<String, Object> getTxData() {
    return txData;
```

```
    }

    public void setTxData(Map<String, Object> txData) {
        this.txData = txData;
    }

    public ContractResultDto getContractResult() {
        return contractResult;
    }

    public void setContractResult(ContractResultDto contractResult) {
        this.contractResult = contractResult;
    }
}
```

49:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractTransactionInfoDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.storage.po.TransactionInfoPo;
import io.nuls.kernel.model.NulsDigestData;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: PierreLuo
 * @date: 2018/7/23
 */
@ApiModel(value = "ContractTransactionInfoDtoJSON")
public class ContractTransactionInfoDto {

    private static final String CREATE_INFO = "contract create";
    private static final String CALL_INFO = "contract call";
    private static final String DELETE_INFO = "contract delete";
    private static final String TRANSFER_INFO = "contract transfer";

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

```
@ApiModelProperty(name = "time", value = "")
private long time;
@ApiModelProperty(name = "txType", value = "")
private int txType;
@ApiModelProperty(name = "status", value = "")
private byte status;
@ApiModelProperty(name = "info", value = "")
private String info;

public ContractTransactionInfoDto() {

}

public ContractTransactionInfoDto(TransactionInfoPo po) {
    if(po == null) {
        return;
    }
    this.txHash = po.getTxHash().getDigestHex();
    this.blockHeight = po.getBlockHeight();
    this.time = po.getTime();
    this.txType = po.getTxType();
    this.status = po.getStatus();
    if(this.txType == ContractConstant.TX_TYPE_CREATE_CONTRACT) {
        this.info = CREATE_INFO;
    } else if(this.txType == ContractConstant.TX_TYPE_CALL_CONTRACT) {
        this.info = CALL_INFO;
    } else if(this.txType == ContractConstant.TX_TYPE_DELETE_CONTRACT) {
        this.info = DELETE_INFO;
    } else if(this.txType == ContractConstant.TX_TYPE_CONTRACT_TRANSFER) {
        this.info = TRANSFER_INFO;
    }
}

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 int compareTo(long thatTime) {
    if(this.time > thatTime) {
        return -1;
```

```
        } else if(this.time < thatTime) {
            return 1;
        }
        return 0;
    }
}

50:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\model\ContractTransferDataDto.java
package io.nuls.contract.rpc.model;


import io.nuls.contract.entity.txdata.ContractTransferData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @Author: PierreLuo
 */
@ApiModel(value = "ContractTransferDataDtoJSON")
public class ContractTransferDataDto{

    @ApiModelProperty(name = "orginTxHash", value = "hash")
    private String orginTxHash;
    @ApiModelProperty(name = "contractAddress", value = "")
    private String contractAddress;
    @ApiModelProperty(name = "success", value = "", 0-, 1-")
    private byte success;

    public ContractTransferDataDto(ContractTransferData transferData) {
        NulsDigestData thatOrginTxHash = transferData.getOrginTxHash();
        this.orginTxHash = thatOrginTxHash == null ? null : thatOrginTxHash.getDigestHex();
        this.contractAddress =
AddressTool.getStringAddressByBytes(transferData.getContractAddress());
        this.success = transferData.getSuccess();
    }

    public String getOrginTxHash() {
        return orginTxHash;
    }
```

```
    public void setOrginTxHash(String orginTxHash) {
        this.orginTxHash = orginTxHash;
    }

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    public byte getSuccess() {
        return success;
    }

    public void setSuccess(byte success) {
        this.success = success;
    }
}
```

51:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractTransferDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.dto.ContractTransfer;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.AddressTool;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: PierreLuo
 */
@ApiModel(value = "ContractTransferDtoJSON")
public class ContractTransferDto {

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

```
@ApiModelProperty(name = "to", value = "")
private String to;
@ApiModelProperty(name = "value", value = "")
private long value;
@ApiModelProperty(name = "fee", value = "")
private long fee;
@ApiModelProperty(name = "isSendBack", value = "")
private boolean isSendBack;
@ApiModelProperty(name = "orginTxHash", value = "hash")
private String orginTxHash;

public ContractTransferDto(ContractTransfer transfer) {
    this.from = AddressTool.getStringAddressByBytes(transfer.getFrom());
    this.to = AddressTool.getStringAddressByBytes(transfer.getTo());
    this.value = transfer.getValue().getValue();
    this.fee = transfer.getFee().getValue();
    this.isSendBack = transfer.isSendBack();
    NulsDigestData thatHash = transfer.getHash();
    this.txHash = thatHash == null ? null : thatHash.getDigestHex();
    NulsDigestData thatOrginTxHash = transfer.getOrginHash();
    this.orginTxHash = thatOrginTxHash == null ? null : thatOrginTxHash.getDigestHex();
}

public String getFrom() {
    return from;
}

public void setFrom(String from) {
    this.from = from;
}

public String getTo() {
    return to;
}

public void setTo(String to) {
    this.to = to;
}

public long getValue() {
    return value;
}
```

```
    public void setValue(long value) {
        this.value = value;
    }

    public long getFee() {
        return fee;
    }

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

    public boolean isSendBack() {
        return isSendBack;
    }

    public void setSendBack(boolean sendBack) {
        isSendBack = sendBack;
    }

    public String getTxHash() {
        return txHash;
    }

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

    public String getOrginTxHash() {
        return orginTxHash;
    }

    public void setOrginTxHash(String orginTxHash) {
        this.orginTxHash = orginTxHash;
    }
}
```

52:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\ContractUtxoDto.java

```
package io.nuls.contract.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 ContractUtxoDto {
    @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 ContractUtxoDto(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;
    }
}
```

53:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\CreateContractDataDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.entity.txdata.CreateContractData;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.utils.AddressTool;

/**
 * @author: PierreLuo
 */
public class CreateContractDataDto {
    private String sender;
    private String contractAddress;
    private long value;
    private String hexCode;
    private long gasLimit;
    private long price;
    private byte argsCount;
    private String[][] args;

    public CreateContractDataDto(CreateContractData create) {
```

```
    this.sender = AddressTool.getStringAddressByBytes(create.getSender());
    this.contractAddress = AddressTool.getStringAddressByBytes(create.getContractAddress());
    this.value = create.getValue();
    this.hexCode = Hex.encode(create.getCode());
    this.gasLimit = create.getGasLimit();
    this.price = create.getPrice();
    this.argsCount = create.getArgsCount();
    this.args = create.getArgs();
}

public String getSender() {
    return sender;
}

public void setSender(String sender) {
    this.sender = sender;
}

public String getContractAddress() {
    return contractAddress;
}

public void setContractAddress(String contractAddress) {
    this.contractAddress = contractAddress;
}

public long getValue() {
    return value;
}

public void setValue(long value) {
    this.value = value;
}

public String getHexCode() {
    return hexCode;
}

public void setHexCode(String hexCode) {
    this.hexCode = hexCode;
}
```

```
    public long getGasLimit() {
        return gasLimit;
    }

    public void setGasLimit(long gasLimit) {
        this.gasLimit = gasLimit;
    }

    public long getPrice() {
        return price;
    }

    public void setPrice(long price) {
        this.price = price;
    }

    public byte getArgsCount() {
        return argsCount;
    }

    public void setArgsCount(byte argsCount) {
        this.argsCount = argsCount;
    }

    public String[][] getArgs() {
        return args;
    }

    public void setArgs(String[][] args) {
        this.args = args;
    }
}
```

54:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\model\DeleteContractDataDto.java

```
package io.nuls.contract.rpc.model;

import io.nuls.contract.entity.txdata.DeleteContractData;
import io.nuls.kernel.utils.AddressTool;

/**
 * @author: PierreLuo
```

```
 */
public class DeleteContractDataDto {
    private String sender;
    private String contractAddress;

    public DeleteContractDataDto(DeleteContractData delete) {
        this.sender = AddressTool.getStringAddressByBytes(delete.getSender());
        this.contractAddress = AddressTool.getStringAddressByBytes(delete.getContractAddress());
    }

    public String getSender() {
        return sender;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }
}

55:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\model\InputDto.java
 */

package io.nuls.contract.rpc.model;

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().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;
    }
}

56:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\main\java\io\nuls\contract\rpc\model\OutputDto.java
 */

package io.nuls.contract.rpc.model;

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 = "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.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;
    }
}
```

57:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\main\java\io\nuls\contract\rpc\resource\ContractResource.java

```
package io.nuls.contract.rpc.resource;

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.Account;
import io.nuls.account.service.AccountService;
import io.nuls.account.util.AccountTool;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTokenInfo;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.entity.ContractInfoDto;
import io.nuls.contract.entity.tx.CreateContractTransaction;
import io.nuls.contract.entity.txdata.ContractData;
import io.nuls.contract.entity.txdata.CreateContractData;
import io.nuls.contract.helper.VMHelper;
import io.nuls.contract.ledger.manager.ContractBalanceManager;
import io.nuls.contract.ledger.module.ContractBalance;
import io.nuls.contract.ledger.service.ContractTransactionInfoService;
import io.nuls.contract.ledger.service.ContractUtxoService;
import io.nuls.contract.ledger.util.ContractLedgerUtil;
import io.nuls.contract.rpc.form.*;
import io.nuls.contract.rpc.model.*;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.service.ContractTxService;
```

```
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.po.ContractCollectionInfoPo;
import io.nuls.contract.storage.po.TransactionInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.contract.storage.service.ContractCollectionStorageService;
import io.nuls.contract.storage.service.ContractTokenTransferStorageService;
import io.nuls.contract.storage.service.ContractUtxoStorageService;
import io.nuls.contract.util.ContractCoinComparator;
import io.nuls.contract.util.ContractUtil;
import io.nuls.contract.util.VMContext;
import io.nuls.contract.vm.program.*;
import io.nuls.core.tools.array.ArraysTool;
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.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.db.model.Entry;
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.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.TransactionFeeCalculator;
import io.nuls.kernel.utils.VarInt;
import io.nuls.ledger.constant.LedgerErrorCode;
import io.nuls.ledger.service.LedgerService;
import io.nuls.ledger.util.LedgerUtil;
import io.swagger.annotations.*;
import org.apache.commons.io.IOUtils;
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.IOException;
import java.io.InputStream;
import java.math.BigInteger;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static io.nuls.contract.constant.ContractConstant.MAX_GASLIMIT;
import static org.apache.commons.lang3.StringUtils.EMPTY;

/**
 * @author: PierreLuo
 */
@Path("/contract")
@Api(value = "/contract", description = "contract")
@Component
public class ContractResource implements InitializingBean {

    @Autowired
    private ContractTxService contractTxService;

    @Autowired
    private ContractService contractService;

    @Autowired
    private LedgerService ledgerService;

    @Autowired
    private ContractAddressStorageService contractAddressStorageService;

    @Autowired
    private ContractUtxoStorageService contractUtxoStorageService;

    @Autowired
    private ContractTransactionInfoService contractTransactionInfoService;

    @Autowired
    private ContractCollectionStorageService contractCollectionStorageService;
```

```
    @Autowired
    private ContractTokenTransferStorageService contractTokenTransferStorageService;

    @Autowired
    private ContractUtxoService contractUtxoService;

    @Autowired
    private ContractBalanceManager contractBalanceManager;

    @Autowired
    private AccountService accountService;

    @Autowired
    private AccountLedgerService accountLedgerService;

    @Autowired
    private VMHelper vmHelper;

    @Autowired
    private VMContext vmContext;

    private ProgramExecutor programExecutor;

    @Override
    public void afterPropertiesSet() throws NulsException {
        programExecutor = vmHelper.getProgramExecutor();
    }

    @POST
    @Path("/create")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult createContract(@ApiParam(name = "createForm", value = "", required =
true) ContractCreate create) {
        if (create == null || create.getGasLimit() < 0 || create.getPrice() < 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

        if (!AddressTool.validAddress(create.getSender())) {
```

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

        String contractCode = create.getContractCode();
        if(StringUtils.isBlank(contractCode)) {
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER).toRpcClientResult();
        }

        byte[] contractCodeBytes = Hex.decode(contractCode);

        ProgramMethod method =
vmHelper.getMethodInfoByCode(ContractConstant.CONTRACT_CONSTRUCTOR, null,
contractCodeBytes);
        String[][] args = null;
        if(method != null) {
            args = create.getArgs(method.argsType2Array());
        }

        return contractTxService.contractCreateTx(create.getSender(),
                create.getGasLimit(),
                create.getPrice(),
                contractCodeBytes,
                args,
                create.getPassword(),
                create.getRemark()).toRpcClientResult();
    }

    @POST
    @Path("/constructor")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = ContractInfoDto.class)
    })
    public RpcClientResult contractConstructor(@ApiParam(name = "createForm", value = "",
required = true) ContractCode code) {
        if (code == null) {
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER).toRpcClientResult();
        }

        String contractCode = code.getContractCode();
        if(StringUtils.isBlank(contractCode)) {
```

```
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER).toRpcClientResult();
        }

        byte[] contractCodeBytes = Hex.decode(contractCode);
        ContractInfoDto contractInfoDto = vmHelper.getConstructor(contractCodeBytes);
        if(contractInfoDto == null || contractInfoDto.getConstructor() == null) {
            return Result.getFailed(ContractErrorCode.ILLEGAL_CONTRACT).toRpcClientResult();
        }
        Map<String, Object> resultMap = MapUtil.createLinkedHashMap(2);
        resultMap.put("constructor", contractInfoDto.getConstructor());
        resultMap.put("isNrc20", contractInfoDto.isNrc20());
        return Result.getSuccess().setData(resultMap).toRpcClientResult();
    }

    @POST
    @Path("/precreate")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult preCreateContract(@ApiParam(name = "preCreateForm", value = "",
required = true) PreContractCreate create) {
        if (create == null || create.getGasLimit() < 0 || create.getPrice() < 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

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

        String contractCode = create.getContractCode();
        if(StringUtils.isBlank(contractCode)) {
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER).toRpcClientResult();
        }

        byte[] contractCodeBytes = Hex.decode(contractCode);

        ProgramMethod method =
vmHelper.getMethodInfoByCode(ContractConstant.CONTRACT_CONSTRUCTOR, null,
contractCodeBytes);
        String[][] args = null;
```

```
        if(method != null) {
            args = create.getArgs(method.argsType2Array());
        }

        return contractTxService.contractPreCreateTx(create.getSender(),
                create.getGasLimit(),
                create.getPrice(),
                contractCodeBytes,
                args,
                null,
                create.getRemark()).toRpcClientResult();
    }

    @POST
    @Path("/imputedgas/create")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "Gas")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult imputedGasCreateContract(@ApiParam(name =
"imputedGasCreateForm", value = "Gas", required = true) ImputedGasContractCreate create) {
        try {
            Map<String, Object> resultMap = MapUtil.createHashMap(1);
            resultMap.put("gasLimit", 1);
            long price = create.getPrice();
            if (create == null || price <= 0) {
                return Result.getSuccess().setData(resultMap).toRpcClientResult();
            }

            String sender = create.getSender();
            Result<Account> accountResult = accountService.getAccount(sender);
            if (accountResult.isFailed()) {
                return Result.getSuccess().setData(resultMap).toRpcClientResult();
            }

            String contractCode = create.getContractCode();
            if(StringUtils.isBlank(contractCode)) {
                return Result.getSuccess().setData(resultMap).toRpcClientResult();
            }

            //
```

```
        Address contractAddress = AccountTool.createContractAddress();
        byte[] contractAddressBytes = contractAddress.getAddressBytes();
        byte[] senderBytes = AddressTool.getAddress(sender);
        byte[] contractCodeBytes = Hex.decode(contractCode);

        ProgramMethod method =
vmHelper.getMethodInfoByCode(ContractConstant.CONTRACT_CONSTRUCTOR, null,
contractCodeBytes);
        String[][] args = null;
        if(method != null) {
            args = create.getArgs(method.argsType2Array());
        }

        //
        BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
        long blockHeight = blockHeader.getHeight();
        //
        byte[] prevStateRoot = ContractUtil.getStateRoot(blockHeader);
        AssertUtil.canNotEmpty(prevStateRoot, "All features of the smart contract are locked.");
        // VMGas
        ProgramCreate programCreate = new ProgramCreate();
        programCreate.setContractAddress(contractAddressBytes);
        programCreate.setSender(senderBytes);
        programCreate.setValue(BigInteger.valueOf(0L));
        programCreate.setPrice(price);
        programCreate.setGasLimit(MAX_GASLIMIT);
        programCreate.setNumber(blockHeight);
        programCreate.setContractCode(contractCodeBytes);
        if(args != null) {
            programCreate.setArgs(args);
        }
        programCreate.setEstimateGas(true);

        ProgramExecutor track = programExecutor.begin(prevStateRoot);
        ProgramResult programResult = track.create(programCreate);
        if(!programResult.isSuccess()) {
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        }
        long gasUsed = programResult.getGasUsed();
        // 1.5Gas
        gasUsed += gasUsed >> 1;
        resultMap.put("gasLimit", gasUsed);
```

```
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        } catch (Exception e) {
            return Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
        }
    }

    @POST
    @Path("/call")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult callContract(@ApiParam(name = "callForm", value = "", required = true)
ContractCall call) {
        if (call == null || call.getValue() < 0 || call.getGasLimit() < 0 || call.getPrice() < 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

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

        String contractAddress = call.getContractAddress();
        if (!AddressTool.validAddress(contractAddress)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }

        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }

        ProgramMethod method =
vmHelper.getMethodInfoByContractAddress(call.getMethodName(), call.getMethodDesc(),
contractAddressBytes);
        String[][] args = null;
        if(method != null) {
            args = call.getArgs(method.argsType2Array());
        }
```

```
        return contractTxService.contractCallTx(call.getSender(),
                Na.valueOf(call.getValue()),
                call.getGasLimit(),
                call.getPrice(),
                contractAddress,
                call.getMethodName(),
                call.getMethodDesc(),
                args,
                call.getPassword(),
                call.getRemark()).toRpcClientResult();
    }

    @POST
    @Path("/transfer")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult transfer(@ApiParam(name = "transferForm", value = "", required = true)
ContractTransfer transfer) {
        if (transfer == null || transfer.getAmount() < 0 || transfer.getGasLimit() < 0 || transfer.getPrice()
< 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

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

        String contractAddress = transfer.getToAddress();
        if (!AddressTool.validAddress(contractAddress)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }

        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }

        return contractTxService.contractCallTx(transfer.getAddress(),
```

```
            Na.valueOf(transfer.getAmount()),
            transfer.getGasLimit(),
            transfer.getPrice(),
            contractAddress,
            ContractConstant.BALANCE_TRIGGER_METHOD_NAME,
            ContractConstant.BALANCE_TRIGGER_METHOD_DESC,
            null,
            transfer.getPassword(),
            transfer.getRemark()).toRpcClientResult();
    }

    @POST
    @Path("/token/transfer")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "token")
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult tokenTransfer(@ApiParam(name = "tokenTransferForm", value = "token
", required = true) ContractTokenTransfer transfer) {
        if (transfer == null || transfer.getAmount() == null ||
            !StringUtils.isNumeric(transfer.getAmount()) ||
            new BigInteger(transfer.getAmount()).compareTo(BigInteger.ZERO) < 0 ||
transfer.getGasLimit() < 0 || transfer.getPrice() < 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

        String from = transfer.getAddress();
        String to = transfer.getToAddress();
        if (!AddressTool.validAddress(from)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }

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

        String contractAddress = transfer.getContractAddress();
        if (!AddressTool.validAddress(contractAddress)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
```

```
        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
        Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
        ContractAddressInfoPo po = contractAddressInfoResult.getData();
        if(po == null) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }
        if(!po.isNrc20()) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_NOT_NRC20).toRpcClientResult();
        }
        Object[] argsObj = new Object[] {to, transfer.getAmount()};

        return contractTxService.contractCallTx(transfer.getAddress(),
                Na.ZERO,
                transfer.getGasLimit(),
                transfer.getPrice(),
                contractAddress,
                ContractConstant.NRC20_METHOD_TRANSFER,
                null,
                ContractUtil.twoDimensionalArray(argsObj),
                transfer.getPassword(),
                transfer.getRemark()).toRpcClientResult();
    }

    @POST
    @Path("/transfer/fee")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult transferFee(@ApiParam(name = "transferFeeForm", value = "", required
= true) ContractTransferFee transferFee) {
        if (transferFee == null || transferFee.getAmount() < 0 || transferFee.getGasLimit() < 0 ||
transferFee.getPrice() < 0) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

        String fromAddress = transferFee.getAddress();
        if (!AddressTool.validAddress(fromAddress)) {
```

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

        String contractAddress = transferFee.getToAddress();
        if (!AddressTool.validAddress(contractAddress)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }

        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }

        Result result = contractTxService.transferFee(fromAddress,
                Na.valueOf(transferFee.getAmount()),
                transferFee.getGasLimit(),
                transferFee.getPrice(),
                contractAddress,
                ContractConstant.BALANCE_TRIGGER_METHOD_NAME,
                ContractConstant.BALANCE_TRIGGER_METHOD_DESC,
                null,
                transferFee.getRemark());
        if(result.isSuccess()) {
            Object[] datas = (Object[]) result.getData();
            if(datas == null) {
                return Result.getFailed(ContractErrorCode.DATA_ERROR).toRpcClientResult();
            }
            Na fee = (Na) datas[0];
            Transaction tx = (Transaction) datas[1];
            Result rs =
accountLedgerService.getMaxAmountOfOnce(AddressTool.getAddress(fromAddress), tx,
                    TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
            Map<String, Long> map = new HashMap<>();
            Long maxAmount = null;
            if (rs.isSuccess()) {
                maxAmount = ((Na) rs.getData()).getValue();
            }
            map.put("fee", fee.getValue());
            map.put("maxAmount", maxAmount);
            result.setData(map);
            return result.toRpcClientResult();
```

```
        } else {
            return result.toRpcClientResult();
        }
    }

    @POST
    @Path("/view")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult invokeViewContract(
            @ApiParam(name = "constantCallForm", value = "", required = true) ContractViewCall
viewCall) {
        try {
            String contractAddress = viewCall.getContractAddress();
            String methodName = viewCall.getMethodName();
            if (StringUtils.isBlank(contractAddress) || StringUtils.isBlank(methodName)) {
                return Result.getFailed(ContractErrorCode.NULL_PARAMETER).toRpcClientResult();
            }

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

            byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
            if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
                return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
            }

            ProgramMethod method = vmHelper.getMethodInfoByContractAddress(methodName,
viewCall.getMethodDesc(), contractAddressBytes);

            if(method == null || !method.isView()) {
                return
Result.getFailed(ContractErrorCode.CONTRACT_NON_VIEW_METHOD).toRpcClientResult();
            }

            ProgramResult programResult = vmHelper.invokeViewMethod(contractAddressBytes,
methodName, viewCall.getMethodDesc(),
```

```
                viewCall.getArgs(method.argsType2Array()));

        Result result;
        if(!programResult.isSuccess()) {
            result = Result.getFailed(ContractErrorCode.DATA_ERROR);
            result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
        } else {
            result = Result.getSuccess();
            Map<String, String> resultMap = MapUtil.createLinkedHashMap(2);
            resultMap.put("result", programResult.getResult());
            result.setData(resultMap);
        }
        return result.toRpcClientResult();
    } catch (Exception e) {
        Log.error("invoke contract view method error.", e);
        return Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
    }
}

@POST
@Path("/imputedgas/call")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "Gas")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success")
})
public RpcClientResult imputedGasCallContract(@ApiParam(name = "imputedGasCallForm",
value = "Gas", required = true) ImputedGasContractCall call) {
    try {

        Map<String, Object> resultMap = MapUtil.createHashMap(1);
        resultMap.put("gasLimit", 1);

        String sender = call.getSender();
        Result<Account> accountResult = accountService.getAccount(sender);
        if (accountResult.isFailed()) {
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        }

        String contractAddress = call.getContractAddress();
        if (!AddressTool.validAddress(contractAddress)) {
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
```

```
        }
        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);

        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        }

        String methodName = call.getMethodName();
        // gas0
        ProgramMethod programMethod =
vmHelper.getMethodInfoByContractAddress(methodName, call.getMethodDesc(),
contractAddressBytes);
        if(programMethod == null || programMethod.isView()) {
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        }

        //
        BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
        long blockHeight = blockHeader.getHeight();
        //
        byte[] prevStateRoot = ContractUtil.getStateRoot(blockHeader);
        AssertUtil.canNotEmpty(prevStateRoot, "All features of the smart contract are locked.");


        long price = call.getPrice();
        if (call == null || call.getValue() < 0 || call.getPrice() <= 0) {
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        }

        byte[] senderBytes = AddressTool.getAddress(sender);
        String[][] args = null;
        if(programMethod != null) {
            args = call.getArgs(programMethod.argsType2Array());
        }

        // VMGas
        ProgramCall programCall = new ProgramCall();
        programCall.setContractAddress(contractAddressBytes);
        programCall.setSender(senderBytes);
        programCall.setValue(BigInteger.valueOf(call.getValue()));
        programCall.setPrice(price);
        programCall.setGasLimit(MAX_GASLIMIT);
```

```
            programCall.setNumber(blockHeight);
            programCall.setMethodName(call.getMethodName());
            programCall.setMethodDesc(call.getMethodDesc());
            programCall.setArgs(args);
            programCall.setEstimateGas(true);

            ProgramExecutor track = programExecutor.begin(prevStateRoot);
            ProgramResult programResult = track.call(programCall);
            if(!programResult.isSuccess()) {
                return Result.getSuccess().setData(resultMap).toRpcClientResult();
            }
            long gasUsed = programResult.getGasUsed();
            // 1.5Gas
            gasUsed += gasUsed >> 1;
            resultMap.put("gasLimit", gasUsed);
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        } catch (Exception e) {
            Log.error(e);
            return Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
        }
    }

    @POST
    @Path("/imputedprice")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "price")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult imputedPrice(@ApiParam(name = "imputedPriceForm", value = "price",
required = true) ImputedPrice imputedPrice) {
        try {

            String address = imputedPrice.getSender();
            if (!AddressTool.validAddress(address)) {
                return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
            }

            Result<Account> accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult.toRpcClientResult();
            }
```

```
            byte[] addressBytes = AddressTool.getAddress(address);
            long price = vmHelper.getLastedPriceForAccount(addressBytes);

            return Result.getSuccess().setData(price).toRpcClientResult();
        } catch (Exception e) {
            return Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
        }
    }

    @POST
    @Path("/delete")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult deleteContract(@ApiParam(name = "deleteForm", value = "", required =
true) ContractDelete delete) {
        if (delete == null) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        if (!AddressTool.validAddress(delete.getSender())) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }

        String contractAddress = delete.getContractAddress();
        if (!AddressTool.validAddress(contractAddress)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }

        return contractTxService.contractDeleteTx(delete.getSender(),
                contractAddress,
                delete.getPassword(),
                delete.getRemark()).toRpcClientResult();
    }


    @GET
    @Path("/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
```

```
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult validateContractAddress(@ApiParam(name="address", value="",
required = true)
                                                   @PathParam("address") String address) {
        if (StringUtils.isBlank(address)) {
            return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
        }
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }

        try {
            boolean isContractAddress = false;
            boolean isPayable = false;
            boolean isNrc20 = false;
            long decimals = 0L;
            do {
                byte[] contractAddressBytes = AddressTool.getAddress(address);
                Result<ContractAddressInfoPo> contractAddressInfoPoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
                if(contractAddressInfoPoResult.isFailed()) {
                    break;
                }
                ContractAddressInfoPo contractAddressInfoPo =
contractAddressInfoPoResult.getData();
                if(contractAddressInfoPo == null) {
                    break;
                }
                isContractAddress = true;
                isPayable = contractAddressInfoPo.isAcceptDirectTransfer();
                isNrc20 = contractAddressInfoPo.isNrc20();
                if(isNrc20) {
                    decimals = contractAddressInfoPo.getDecimals();
                }
            } while (false);
            Map<String, Object> resultMap = MapUtil.createLinkedHashMap(2);
            resultMap.put("isContractAddress", isContractAddress);
            resultMap.put("isPayable", isPayable);
            resultMap.put("isNrc20", isNrc20);
            if(isNrc20) {
```

```
                resultMap.put("decimals", decimals);
            }


            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        } catch (Exception e) {
            return Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
        }
    }

    @GET
    @Path("/info/wallet/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult getContractInfo(
            @ApiParam(name = "address", value = "", required = true) @PathParam("address") String
contractAddress,
            @ApiParam(name = "accountAddress", value = "", required = false)
@QueryParam("accountAddress") String accountAddress) {
        return this.getContractInfoWithLock(contractAddress, accountAddress, true);
    }

    @GET
    @Path("/info/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult getContractInfo(
            @ApiParam(name = "address", value = "", required = true) @PathParam("address") String
contractAddress) {
        return this.getContractInfoWithLock(contractAddress, null, false);

    }

    private RpcClientResult getContractInfoWithLock(
            String contractAddress,
            String accountAddress,
```

```
            boolean isNeedLock) {
        try {
            boolean hasAccountAddress = false;
            if(StringUtils.isNotBlank(accountAddress)) {
                Result<Account> accountResult = accountService.getAccount(accountAddress);
                if (accountResult.isFailed()) {
                    return accountResult.toRpcClientResult();
                }
                hasAccountAddress = true;
            }

            if (contractAddress == null) {
                return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
            }

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

            byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);

            Result<ContractAddressInfoPo> contractAddressInfoPoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
            if(contractAddressInfoPoResult.isFailed()) {
                return contractAddressInfoPoResult.toRpcClientResult();
            }
            ContractAddressInfoPo contractAddressInfoPo = contractAddressInfoPoResult.getData();
            if(contractAddressInfoPo == null) {
                return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
            }

            if(isNeedLock && contractAddressInfoPo.isLock()) {
                return Result.getFailed(ContractErrorCode.CONTRACT_LOCK).toRpcClientResult();
            }

            byte[] prevStateRoot =
ContractUtil.getStateRoot(NulsContext.getInstance().getBestBlock().getHeader());

            ProgramExecutor track = programExecutor.begin(prevStateRoot);
            ProgramStatus status = track.status(contractAddressBytes);
            List<ProgramMethod> methods = track.method(contractAddressBytes);
```

```
        Map<String, Object> resultMap = MapUtil.createLinkedHashMap(8);
        try {
            byte[] createTxHash = contractAddressInfoPo.getCreateTxHash();
            NulsDigestData create = new NulsDigestData();
            create.parse(createTxHash, 0);
            resultMap.put("createTxHash", create.getDigestHex());
        } catch (Exception e) {
            Log.error("createTxHash parse error.", e);
        }

        if(hasAccountAddress) {
            //
            boolean isCollect = false;
            Result<ContractCollectionInfoPo> collectionInfoPoResult =
contractCollectionStorageService.getContractAddress(contractAddressBytes);
            ContractCollectionInfoPo contractCollectionPo = collectionInfoPoResult.getData();
            if (contractCollectionPo != null) {
                if(contractCollectionPo.getCollectorMap().containsKey(accountAddress)) {
                    isCollect = true;
                }
            }
            resultMap.put("isCollect", isCollect);
        }
        resultMap.put("address", contractAddress);
        resultMap.put("creater",
AddressTool.getStringAddressByBytes(contractAddressInfoPo.getSender()));
        resultMap.put("createTime", contractAddressInfoPo.getCreateTime());
        resultMap.put("blockHeight", contractAddressInfoPo.getBlockHeight());
        resultMap.put("isNrc20", contractAddressInfoPo.isNrc20());
        if(contractAddressInfoPo.isNrc20()) {
            resultMap.put("nrc20TokenName", contractAddressInfoPo.getNrc20TokenName());
            resultMap.put("nrc20TokenSymbol", contractAddressInfoPo.getNrc20TokenSymbol());
            resultMap.put("decimals", contractAddressInfoPo.getDecimals());
            resultMap.put("totalSupply",
ContractUtil.bigInteger2String(contractAddressInfoPo.getTotalSupply()));
        }
        resultMap.put("status", status.name());
        resultMap.put("method", methods);

        return Result.getSuccess().setData(resultMap).toRpcClientResult();
    } catch (Exception e) {
```

```
            Log.error(e);
            return Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
        }

    }

    @GET
    @Path("/balance/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult getContractBalance(@ApiParam(name = "address", value = "", required
= true) @PathParam("address") String contractAddress) {
        if (contractAddress == null) {
            return Result.getFailed(ContractErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

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

        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }

        Result<ContractBalance> result = contractUtxoService.getBalance(contractAddressBytes);
        ContractBalance balance = (ContractBalance) result.getData();
        Map<String, Object> resultMap = MapUtil.createLinkedHashMap(4);
        resultMap.put("address", contractAddress);
        resultMap.put("balance", balance == null ? Na.ZERO : balance.getBalance().toString());
        resultMap.put("usable", balance == null ? Na.ZERO : balance.getRealUsable().toString());
        resultMap.put("locked", balance == null ? Na.ZERO : balance.getLocked().toString());
        return Result.getSuccess().setData(resultMap).toRpcClientResult();
    }

    @GET
    @Path("/balance/token/{contractAddress}/{address}")
    @Produces(MediaType.APPLICATION_JSON)
```

```
    @ApiOperation(value = "token")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult getAccountTokenBalance(
            @ApiParam(name = "contractAddress", value = "", required = true)
@PathParam("contractAddress") String contractAddress,
            @ApiParam(name = "address", value = "", required = true) @PathParam("address") String
address) {
        Result<ContractTokenInfo> tokenInfoResult =
contractService.getContractTokenViaVm(address, contractAddress);
        if(tokenInfoResult.isFailed()) {
            return tokenInfoResult.toRpcClientResult();
        }
        ContractTokenInfo data = tokenInfoResult.getData();
        ContractTokenInfoDto dto = null;
        if(data != null) {
            dto = new ContractTokenInfoDto(data);
            dto.setStatus(data.getStatus());
        }
        return Result.getSuccess().setData(dto).toRpcClientResult();
    }

    @GET
    @Path("/result/{hash}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = ContractResultDto.class)
    })
    public RpcClientResult getContractTxResult(@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();
        }

        try {
            ContractResultDto contractResultDto = null;
```

```
        ContractResult contractExecuteResult;
        boolean flag = true;
        String msg = EMPTY;
        //long confirmCount = 0L;
        do {
            NulsDigestData txHash = NulsDigestData.fromDigestHex(hash);
            Transaction tx = ledgerService.getTx(txHash);
            if (tx == null) {
                flag = false;
                msg = TransactionErrorCode.TX_NOT_EXIST.getMsg();
                break;
            } else {
                if (!ContractUtil.isContractTransaction(tx)) {
                    flag = false;
                    msg = ContractErrorCode.NON_CONTRACTUAL_TRANSACTION.getMsg();
                    break;
                }
            }
            contractExecuteResult = contractService.getContractExecuteResult(txHash);
            if(contractExecuteResult != null) {
                //long bestBlockHeight = NulsContext.getInstance().getBestHeight();
                //confirmCount = bestBlockHeight - tx.getBlockHeight() + 1;
                Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractExecuteResult.getContractAddre
ss());
                ContractAddressInfoPo po = contractAddressInfoResult.getData();
                if(po != null && po.isNrc20()) {
                    contractExecuteResult.setNrc20(true);
                    if(contractExecuteResult.isSuccess()) {
                        contractResultDto = new ContractResultDto(contractExecuteResult, tx, po);
                    } else {
                        ContractData contractData = (ContractData) tx.getTxData();
                        byte[] sender = contractData.getSender();
                        byte[] infoKey = ArraysTool.concatenate(sender, tx.getHash().serialize(), new
VarInt(0).encode());
                        Result<ContractTokenTransferInfoPo> tokenTransferResult =
contractTokenTransferStorageService.getTokenTransferInfo(infoKey);
                        ContractTokenTransferInfoPo transferInfoPo = tokenTransferResult.getData();
                        contractResultDto = new ContractResultDto(contractExecuteResult, tx, po,
transferInfoPo);
                    }
                } else {
```

```
                    contractResultDto = new ContractResultDto(contractExecuteResult, tx);
                }
                break;
            } else {
                flag = false;
                msg = TransactionErrorCode.DATA_NOT_FOUND.getMsg();
                break;
            }
        } while (false);
        Map<String, Object> resultMap = MapUtil.createLinkedHashMap(2);
        resultMap.put("flag", flag);
        if(!flag && StringUtils.isNotBlank(msg)) {
            resultMap.put("msg", msg);
        }
        if(flag && contractResultDto != null) {
            List<ContractTokenTransferDto> tokenTransfers =
contractResultDto.getTokenTransfers();
            List<ContractTokenTransferDto> realTokenTransfers =
this.filterRealTokenTransfers(tokenTransfers);
            contractResultDto.setTokenTransfers(realTokenTransfers);
            resultMap.put("data", contractResultDto);
        }
        return Result.getSuccess().setData(resultMap).toRpcClientResult();
    } catch (Exception e) {
        Log.error(e);
        return Result.getFailed().setMsg(e.getMessage()).toRpcClientResult();
    }
}

private List<ContractTokenTransferDto>
filterRealTokenTransfers(List<ContractTokenTransferDto> tokenTransfers) {
    if(tokenTransfers == null || tokenTransfers.isEmpty()) {
        return tokenTransfers;
    }
    List<ContractTokenTransferDto> resultDto = new ArrayList<>();
    Map<String, ContractAddressInfoPo> cache =
MapUtil.createHashMap(tokenTransfers.size());
    for(ContractTokenTransferDto tokenTransfer : tokenTransfers) {
        try {
            if(StringUtils.isBlank(tokenTransfer.getName())) {
                String contractAddress = tokenTransfer.getContractAddress();
                ContractAddressInfoPo po = cache.get(contractAddress);
```

```
                if(po == null) {
                    po = contractAddressStorageService.getContractAddressInfo(
                            AddressTool.getAddress(contractAddress)).getData();
                    cache.put(contractAddress, po);
                }
                if(po == null || !po.isNrc20()) {
                    continue;
                }
                tokenTransfer.setNrc20Info(po);
                resultDto.add(tokenTransfer);
            }
        } catch (Exception e) {
            Log.error(e);
        }
    }
    return resultDto;
}

@GET
@Path("/tx/{hash}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response =
ContractTransactionDto.class)
})
public RpcClientResult getContractTx(@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;
    try {
        NulsDigestData txHashObj = NulsDigestData.fromDigestHex(hash);
        Transaction tx = ledgerService.getTx(txHashObj);
        if (tx == null) {
            result = Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
```

```
        } else {
            if(!ContractUtil.isContractTransaction(tx) && tx.getType() !=
NulsConstant.TX_TYPE_COINBASE) {
                return
Result.getFailed(ContractErrorCode.NON_CONTRACTUAL_TRANSACTION).toRpcClientResult()
;
            }

            tx.setStatus(TxStatusEnum.CONFIRMED);
            ContractTransactionDto txDto = null;
            CoinData coinData = tx.getCoinData();
            byte[] txHashBytes = tx.getHash().serialize();
            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 ContractTransactionDto(tx);
                List<OutputDto> outputDtoList = new ArrayList<>();
                // to
                List<Coin> tos = coinData.getTo();
                if(tos != null && tos.size() > 0) {
                    String txHash = hash;
                    OutputDto outputDto;
                    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(org.spongycastle.util.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);
        }
        //
        if(tx.getType() != ContractConstant.TX_TYPE_CONTRACT_TRANSFER) {
            ContractResult contractExecuteResult =
contractService.getContractExecuteResult(txHashObj);
            if(contractExecuteResult != null) {
                Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractExecuteResult.getContractAddre
ss());
                ContractAddressInfoPo po = contractAddressInfoResult.getData();
                if(po != null && po.isNrc20()) {
                    contractExecuteResult.setNrc20(true);
                    if(contractExecuteResult.isSuccess()) {
                        txDto.setContractResult(new ContractResultDto(contractExecuteResult, tx,
po));
                    } else {
                        ContractData contractData = (ContractData) tx.getTxData();
                        byte[] sender = contractData.getSender();
                        byte[] infoKey = ArraysTool.concatenate(sender, txHashBytes, new
VarInt(0).encode());
                        Result<ContractTokenTransferInfoPo> tokenTransferResult =
contractTokenTransferStorageService.getTokenTransferInfo(infoKey);
                        ContractTokenTransferInfoPo transferInfoPo = tokenTransferResult.getData();
                        txDto.setContractResult(new ContractResultDto(contractExecuteResult, tx,
po, transferInfoPo));
                    }
                } else {
                    txDto.setContractResult(new ContractResultDto(contractExecuteResult, tx));
                }
                ContractResultDto contractResultDto = txDto.getContractResult();
                List<ContractTokenTransferDto> tokenTransfers =
contractResultDto.getTokenTransfers();
                List<ContractTokenTransferDto> realTokenTransfers =
this.filterRealTokenTransfers(tokenTransfers);
                contractResultDto.setTokenTransfers(realTokenTransfers);
            }
```

```
            }
            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(ContractTransactionDto 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("/limit/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "addresslimitUTXO")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response =
ContractAccountUtxoDto.class)
    })
    public RpcClientResult getUtxoByAddressAndLimit(
            @ApiParam(name="address", value="", required = true) @PathParam("address") String
address,
            @ApiParam(name="limit", value="()", required = false) @QueryParam("limit") Integer limit)
{
        if (StringUtils.isBlank(address)) {
            return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
        }

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

        byte[] contractAddressBytes = AddressTool.getAddress(address);
        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }

        Result result;
        try {
            boolean isLoadAll = (limit == null);
            List<Coin> coinList = getAllUtxoByAddress(address);
            int limitValue = 0;
            if(!isLoadAll) {
                limitValue = limit.intValue();
            }
            ContractAccountUtxoDto accountUtxoDto = new ContractAccountUtxoDto();
            List<ContractUtxoDto> 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 ContractUtxoDto(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 =
ContractAccountUtxoDto.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(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }

    byte[] contractAddressBytes = AddressTool.getAddress(address);
```

```
        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }

        Result result;
        try {
            List<Coin> coinList = getAllUtxoByAddress(address);
            Na amountNa = Na.valueOf(amount.longValue());

            ContractAccountUtxoDto accountUtxoDto = new ContractAccountUtxoDto();
            List<ContractUtxoDto> 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 ContractUtxoDto(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 = contractUtxoStorageService.loadAllCoinList();
        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.getAddress(), addressBytes)) {
                coin.setOwner(coinEntryBytes.getKey());
                coinList.add(coin);
            }
        }
        Collections.sort(coinList, ContractCoinComparator.getInstance());
        return coinList;
    }

    @GET
    @Path("/tx/list/{contractAddress}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response =
ContractTransactionInfoDto.class)
    })
    public RpcClientResult getTxList(
            @ApiParam(name="contractAddress", value="", required = true)
            @PathParam("contractAddress") String contractAddress,
            @ApiParam(name = "pageNumber", value = "", required = true)
            @QueryParam("pageNumber") Integer pageNumber,
            @ApiParam(name = "pageSize", value = "", required = false)
            @QueryParam("pageSize") Integer pageSize,
            @ApiParam(name = "accountAddress", value = "")
            @QueryParam("accountAddress") String accountAddress) {
        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 (StringUtils.isBlank(contractAddress)) {
            return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
        }

        boolean isFilterAccountAddress = false;
        if(StringUtils.isNotBlank(accountAddress)) {
            Result<Account> accountResult = accountService.getAccount(accountAddress);
            if (accountResult.isFailed()) {
                return accountResult.toRpcClientResult();
            }
            isFilterAccountAddress = true;
        }

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


        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);

        if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
            return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
        }

        Result<List<TransactionInfoPo>> txInfoPoListResult =
contractTransactionInfoService.getTxInfoList(contractAddressBytes);
        List<TransactionInfoPo> orginTxInfoPoList = txInfoPoListResult.getData();
        List<TransactionInfoPo> txInfoPoList = new ArrayList<>();
        do {
            if(orginTxInfoPoList == null || orginTxInfoPoList.size() == 0) {
                break;
            }

            Stream<TransactionInfoPo> transactionInfoPoStream = orginTxInfoPoList.stream()
                    .filter(po -> po.getTxType() !=
ContractConstant.TX_TYPE_CONTRACT_TRANSFER);
            //
            if(isFilterAccountAddress) {
```

```
                byte[] accountAddressBytes = AddressTool.getAddress(accountAddress);
                txInfoPoList = transactionInfoPoStream.filter(po -> checkEquals(po.getAddresses(),
accountAddressBytes, 0)).collect(Collectors.toList());
            } else {
                txInfoPoList = transactionInfoPoStream.collect(Collectors.toList());;
            }
        } while (false);


        Result result = Result.getSuccess();
        List<ContractTransactionDto> infoDtoList = new ArrayList<>();
        Page<ContractTransactionDto> page = new Page<>(pageNumber, pageSize,
txInfoPoList.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();
        }

        //List<ContractTransactionInfoDto> resultList = new ArrayList<>();
        if(txInfoPoList.size() > 0) {
            txInfoPoList.sort(new Comparator<TransactionInfoPo>() {
                @Override
                public int compare(TransactionInfoPo o1, TransactionInfoPo o2) {
                    return o1.compareTo(o2.getTime());
                }
            });

            for (int i = start; i < end; i++) {
                TransactionInfoPo info = txInfoPoList.get(i);
                RpcClientResult txResult = this.getContractTx(info.getTxHash().getDigestHex());
                if (txResult.isFailed()) {
                    continue;
                }
                infoDtoList.add((ContractTransactionDto) txResult.getData());

            }
```

```
            }
            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();
        }
    }

    private boolean checkEquals(byte[] addresses, byte[] desc, int index) {
        try {
            int totalLength = addresses.length;
            int addressLength = Address.ADDRESS_LENGTH;
            int totalCount = totalLength / addressLength;
            int continuousHits = 0;

            for(int i = index, k = 0, flag = i, length = addressLength, count = 0; k < length && count <
totalCount;) {
                if(addresses[i] != desc[k]) {
                    k = 0;
                    i = flag + addressLength;
                    flag = i;
                    continuousHits = 0;
                    count++;
                    continue;
                } else {
                    continuousHits++;
                }

                if(continuousHits == addressLength) {
                    return true;
                }
                i++;
                k++;
            }
            return false;
        } catch (Exception e) {
```

```
            Log.error("check relative addresses error.", e);
            return false;
        }
    }

    @GET
    @Path("/token/list/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "NRC20")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response =
ContractTokenInfoDto.class)
    })
    public RpcClientResult getTokenList(
            @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 (StringUtils.isBlank(address)) {
                return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
            }

            Result<Account> accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult.toRpcClientResult();
            }

            Result<List<ContractTokenInfo>> tokenListResult =
contractBalanceManager.getAllTokensByAccount(address);
```

```
        if(tokenListResult.isFailed()) {
            return tokenListResult.toRpcClientResult();
        }

        List<ContractTokenInfo> tokenInfoList = tokenListResult.getData();

        Result result = Result.getSuccess();
        List<ContractTokenInfoDto> tokenInfoDtoList = new ArrayList<>();
        Page<ContractTokenInfoDto> page = new Page<>(pageNumber, pageSize,
tokenInfoList.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(tokenInfoList.size() > 0) {
            for (int i = start; i < end; i++) {
                ContractTokenInfo info = tokenInfoList.get(i);
                tokenInfoDtoList.add(new ContractTokenInfoDto(info));
            }
        }
        if(tokenInfoDtoList != null && tokenInfoDtoList.size() > 0) {
            byte[] prevStateRoot =
ContractUtil.getStateRoot(NulsContext.getInstance().getBestBlock().getHeader());
            ProgramExecutor track = programExecutor.begin(prevStateRoot);
            for(ContractTokenInfoDto tokenInfo : tokenInfoDtoList) {
tokenInfo.setStatus(track.status(AddressTool.getAddress(tokenInfo.getContractAddress())).ordinal
());
            }
        }
        page.setList(tokenInfoDtoList);

        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("/collection")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "/")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult contractCollection(@ApiParam(name = "collection", value = "/", required
= true) ContractCollection collection) {
        try {
            if (collection == null) {
                return Result.getFailed(ContractErrorCode.NULL_PARAMETER).toRpcClientResult();
            }

            //
            String address = collection.getAccountAddress();
            if (StringUtils.isBlank(address)) {
                return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
            }
            Result<Account> accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult.toRpcClientResult();
            }

            //
            String contractAddress = collection.getContractAddress();
            if (!AddressTool.validAddress(contractAddress)) {
                return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
            }
            byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
            if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
                return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
            }
```

```
        //
        String remarkName = collection.getRemarkName();

        // ,  - hash
        Result<ContractAddressInfoPo> contractAddressInfoPoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
        ContractAddressInfoPo contractAddressInfoPo = contractAddressInfoPoResult.getData();

        if(contractAddressInfoPo == null) {
            return Result.getFailed(ContractErrorCode.DATA_NOT_FOUND).toRpcClientResult();
        }

        //
        Result<ContractCollectionInfoPo> collectionInfoPoResult =
contractCollectionStorageService.getContractAddress(contractAddressBytes);
        ContractCollectionInfoPo po = collectionInfoPoResult.getData();
        Map<String, String> collectorMap;
        if(po != null) {
            collectorMap = po.getCollectorMap();
            if(collectorMap.containsKey(address)) {
                String preRemarkName = collectorMap.get(address);
                if(preRemarkName.equals(remarkName)) {
                    //
                    return Result.getSuccess().toRpcClientResult();
                }
            } else {
                collectorMap.put(address, EMPTY);
            }

        } else {
            po = new ContractCollectionInfoPo();
            po.setCreater(contractAddressInfoPo.getSender());
            po.setContractAddress(contractAddress);
            po.setBlockHeight(contractAddressInfoPo.getBlockHeight());
            Transaction tx = ledgerService.getTx(contractAddressInfoPo.getCreateTxHash());
            if(tx == null) {
                return Result.getFailed(ContractErrorCode.TX_NOT_EXIST).toRpcClientResult();
            }
            po.setCreateTime(tx.getTime());
            collectorMap = MapUtil.createHashMap(4);
            collectorMap.put(address, EMPTY);
            po.setCollectorMap(collectorMap);
```

```
            }

            //
            if(StringUtils.isNotBlank(remarkName)) {
                //if (!StringUtils.validAlias(remarkName)) {
                //    return
Result.getFailed(ContractErrorCode.CONTRACT_NAME_FORMAT_INCORRECT).toRpcClientRe
sult();
                //}
                collectorMap.put(address, remarkName);
            }

            Result result =
contractCollectionStorageService.saveContractAddress(contractAddressBytes, po);
            return result.toRpcClientResult();
        } catch (Exception e) {
            Log.error(e);
            return
Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
        }
    }

    @POST
    @Path("/collection/cancel")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult collectionCancel(@ApiParam(name = "collectionBase", value = "",
required = true) ContractAddressBase collection ) {
        try {
            //
            String address = collection.getAccountAddress();
            if (StringUtils.isBlank(address)) {
                return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
            }
            Result<Account> accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult.toRpcClientResult();
            }
```

```
            //
            String contractAddress = collection.getContractAddress();
            if (!AddressTool.validAddress(contractAddress)) {
                return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
            }

            byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
            if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
                return
Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST).toRpcClientResult();
            }

            //
            Result<ContractCollectionInfoPo> collectionInfoPoResult =
contractCollectionStorageService.getContractAddress(contractAddressBytes);
            ContractCollectionInfoPo po = collectionInfoPoResult.getData();
            Map<String, String> collectorMap;
            if(po != null) {
                collectorMap = po.getCollectorMap();
                collectorMap.remove(address);
            }
            Result result =
contractCollectionStorageService.saveContractAddress(contractAddressBytes, po);
            return result.toRpcClientResult();
        } catch (Exception e) {
            Log.error(e);
            return
Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
        }
    }

    @GET
    @Path("/wallet/list/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "()")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = ContractAddressDto.class)
    })
    public RpcClientResult getContractCollectionList(
                        @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 (StringUtils.isBlank(address)) {
                return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
            }

            Result<Account> accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult.toRpcClientResult();
            }

            byte[] addressBytes = AddressTool.getAddress(address);


            LinkedHashMap<String, ContractAddressDto> resultMap = new LinkedHashMap<>();
            //
            LinkedList<Map<String, String>> list =
contractTxService.getLocalUnconfirmedCreateContractTransaction(address);
            if(list != null) {
                String contractAddress;
                Long time;
                ContractAddressDto dto;
                String success;
                for(Map<String, String> map : list) {
                    contractAddress = map.get("contractAddress");
                    time = Long.valueOf(map.get("time"));
                    dto = new ContractAddressDto();
                    dto.setCreate(true);
                    dto.setContractAddress(contractAddress);
                    dto.setCreateTime(time);
```

```
                success = map.get("success");
                if(StringUtils.isNotBlank(success)) {
                    //
                    dto.setStatus(3);
                    dto.setMsg(map.get("msg"));
                } else {
                    dto.setStatus(0);
                }
                resultMap.put(contractAddress, dto);
            }
        }

        byte[] prevStateRoot =
ContractUtil.getStateRoot(NulsContext.getInstance().getBestBlock().getHeader());

        ProgramExecutor track = programExecutor.begin(prevStateRoot);
        byte[] contractAddressBytes;
        String contractAddress;

        //
        Result<List<ContractAddressInfoPo>> contractInfoListResult =
contractAddressStorageService.getContractInfoList(addressBytes);

        List<ContractAddressInfoPo> contractAddressInfoPoList =
contractInfoListResult.getData();
        if(contractAddressInfoPoList != null && contractAddressInfoPoList.size() > 0) {
            contractAddressInfoPoList.sort(new Comparator<ContractAddressInfoPo>() {
                @Override
                public int compare(ContractAddressInfoPo o1, ContractAddressInfoPo o2) {
                    return o1.compareTo(o2.getCreateTime());
                }
            });

            for(ContractAddressInfoPo po : contractAddressInfoPoList) {
                contractAddressBytes = po.getContractAddress();
                contractAddress = AddressTool.getStringAddressByBytes(contractAddressBytes);
                Result<ContractCollectionInfoPo> contractCollectionInfoPoResult =
contractCollectionStorageService.getContractAddress(contractAddressBytes);
                ContractCollectionInfoPo infoPo = contractCollectionInfoPoResult.getData();
                if(infoPo == null) {
                    resultMap.put(contractAddress, new ContractAddressDto(po, true,
```

```
track.status(contractAddressBytes).ordinal()));
                } else {
                    resultMap.put(contractAddress, new ContractAddressDto(infoPo, address, true,
track.status(contractAddressBytes).ordinal()));
                }
            }

        }

        //
        List<ContractCollectionInfoPo> contractCollectionPos =
getContractAddressCollection(addressBytes);
        if(contractCollectionPos.size() > 0) {
            contractCollectionPos.sort(new Comparator<ContractCollectionInfoPo>() {
                @Override
                public int compare(ContractCollectionInfoPo o1, ContractCollectionInfoPo o2) {
                    return o1.compareTo(o2.getCreateTime());
                }
            });
            for(ContractCollectionInfoPo po : contractCollectionPos) {
                contractAddress = po.getContractAddress();
                if(resultMap.containsKey(contractAddress)) {
                    continue;
                }
                contractAddressBytes = AddressTool.getAddress(contractAddress);
                resultMap.put(contractAddress, new ContractAddressDto(po, address, false,
track.status(contractAddressBytes).ordinal()));
            }
        }

        List<ContractAddressDto> infoList = new ArrayList<>(resultMap.values());

        Result result = Result.getSuccess();
        List<ContractAddressDto> contractAddressDtoList = new ArrayList<>();
        Page<ContractAddressDto> page = new Page<>(pageNumber, pageSize, infoList.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(infoList.size() > 0) {
                for (int i = start; i < end; i++) {
                    contractAddressDtoList.add(infoList.get(i));
                }
            }
            page.setList(contractAddressDtoList);

            result.setSuccess(true);
            result.setData(page);

            return result.toRpcClientResult();
        } catch (Exception e) {
            Log.error(e);
            return
Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
        }
    }

    @POST
    @Path("/unconfirmed/failed/remove")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public RpcClientResult removeFailedUnconfirmed(@ApiParam(name =
"ContractAddressBase", value = "", required = true)
                                                   ContractAddressBase addressBase) {
        try {
            //
            String address = addressBase.getAccountAddress();
            if (StringUtils.isBlank(address)) {
                return Result.getFailed(LedgerErrorCode.NULL_PARAMETER).toRpcClientResult();
            }
            Result<Account> accountResult = accountService.getAccount(address);
            if (accountResult.isFailed()) {
                return accountResult.toRpcClientResult();
            }
```

```
            //
            String contractAddress = addressBase.getContractAddress();
            if (!AddressTool.validAddress(contractAddress)) {
                return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
            }

            contractTxService.removeLocalFailedUnconfirmedCreateContractTransaction(address,
contractAddress);
            return Result.getSuccess().toRpcClientResult();
        } catch (Exception e) {
            Log.error(e);
            return
Result.getFailed(LedgerErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
        }
    }

    /**
     *
     *
     * @return
     * @param address
     */
    private List<ContractCollectionInfoPo> getContractAddressCollection(byte[] address) {
        //
        Result<List<ContractCollectionInfoPo>> contractAddressList =
contractCollectionStorageService.getContractAddressList();
        List<ContractCollectionInfoPo> contractCollectionPos = contractAddressList.getData();
        if (contractCollectionPos == null) {
            return new ArrayList<>();
        }
        //
        List<ContractCollectionInfoPo> result = new ArrayList<>();
        for(ContractCollectionInfoPo po : contractCollectionPos) {
            //
            if (Arrays.equals(po.getCreater(), address)) {
                continue;
            }
            if(po.getCollectorMap().containsKey(AddressTool.getStringAddressByBytes(address))) {
                result.add(po);
            }
        }
```

```
        return result;
    }


    @POST
    @Path("/upload/constructor")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    @ApiOperation(value = "jar")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult upload(@ApiParam(name = "jarfile", value = "jar", required =
true)@FormDataParam("jarfile") InputStream jarfile) {
        if (null == jarfile) {
            return Result.getFailed(AccountErrorCode.NULL_PARAMETER).toRpcClientResult();
        }
        try {
            byte[] contractCode = IOUtils.toByteArray(jarfile);
            ContractInfoDto contractInfoDto = vmHelper.getConstructor(contractCode);
            if(contractInfoDto == null || contractInfoDto.getConstructor() == null) {
                return Result.getFailed(ContractErrorCode.ILLEGAL_CONTRACT).toRpcClientResult();
            }
            Map<String, Object> resultMap = MapUtil.createLinkedHashMap(2);
            resultMap.put("constructor", contractInfoDto.getConstructor());
            resultMap.put("isNrc20", contractInfoDto.isNrc20());
            resultMap.put("code", Hex.encode(contractCode));
            return Result.getSuccess().setData(resultMap).toRpcClientResult();
        } catch (IOException e) {
            Log.error(e);
            return Result.getFailed(ContractErrorCode.DATA_ERROR).toRpcClientResult();
        }
    }


    @GET
    @Path("/export/{address}")
    @ApiOperation(value = "jar ")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success")
    })
    public void export(@ApiParam(name = "address", value = "", required = true)
```

```
                            @PathParam("address") String address,
                                    @Context HttpServletResponse response) {
        try {
            if (StringUtils.isBlank(address)) {
                return;
            }

            if (!AddressTool.validAddress(address)) {
                return;
            }

            byte[] contractAddressBytes = AddressTool.getAddress(address);
            if(!ContractLedgerUtil.isExistContractAddress(contractAddressBytes)) {
                return;
            }
            byte[] addressBytes = AddressTool.getAddress(address);
            Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(addressBytes);
            ContractAddressInfoPo po = contractAddressInfoResult.getData();
            if(po == null) {
                return;
            }
            Transaction tx = ledgerService.getTx(po.getCreateTxHash());
            CreateContractTransaction create = (CreateContractTransaction) tx;
            CreateContractData createTxData = create.getTxData();
            byte[] code = createTxData.getCode();

            //1.ContentType
            response.setContentType("application/octet-stream");
            //2.
            response.addHeader("Content-Disposition", "attachment;filename=" + address + ".jar");
            response.getOutputStream().write(code);
            response.getOutputStream().flush();
        } catch (Exception e) {
            Log.error("Export Exception!");
        }
    }

}
```

58:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-rpc\src\test\java\io\nuls\contract\BaseTest.java

```
 */
package io.nuls.contract;

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();
    }
}

59:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
rpc\src\test\java\io\nuls\contract\rpc\ContractTest.java
 */

package io.nuls.contract.rpc;
```

```
import io.nuls.contract.BaseTest;

public class ContractTest extends BaseTest {
    private static int successCount = 0;
    //private static String IP = "192.168.1.120";
    private static String IP = "127.0.0.1";

    public static void main(String[] args) {
        //createWrapper();
        callWrapper();
    }

    static void createWrapper() {
        long time = System.currentTimeMillis();
        for (int i = 0; i < 1; i++) {
            create();
        }
        System.out.println("" + (System.currentTimeMillis() - time) + " ms");
    }

    static void callWrapper() {
        long time = System.currentTimeMillis();
        for (int i = 0; i < 10000; i++) {
            call();
        }
        System.out.println("" + (System.currentTimeMillis() - time) + " ms");
    }

    private static void call() {
        String address = "Nsdz8mKKFMehRDVRZFyXNuuenugUYM7M";
        String contractAddress = "NseE4LB84BRT6BTgkvCkRmPzpuvowwxM";
        String password = "";
        String remark = "test";
        String methodName = "create";

        String param = "{\"sender\": \"" + address + "\", \"gasLimit\": 200000, \"price\": 25,
\"password\": \"" + password + "\", \"contractAddress\": \"" + contractAddress + "\", \"remark\": \"" +
remark + "\", \"methodName\": \"" + methodName + "\", \"value\" : 10000000000 , \"args\": [\"1\", \"
\", [\"\",\n" +
                "      \"\",\n" +
                "      \"\"]]}";
```

```
        String url = "http://" + IP + ":8001/api/contract/call";


        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(0L);
            //} catch (InterruptedException e) {
            //    e.printStackTrace();
            //}
        }
    }

    private static void create() {
        String address = "Nsdz8mKKFMehRDVRZFyXNuuenugUYM7M";
        String password = "";
        String remark = "test";
        String contractCode =
"504b03040a00000800001c841b4d0000000000000000000000003000400696f2ffeca0000504b0
3040a00000800001c841b4d0000000000000000000000008000000696f2f6e756c732f504b03040
a00000800001c841b4d000000000000000000000000d000000696f2f6e756c732f766f74652f504b
03040a00000800001c841b4d0000000000000000000000016000000696f2f6e756c732f766f7465
2f636f6e74726163742f504b03040a00000800001c841b4d000000000000000000000001c000000
696f2f6e756c732f766f74652f636f6e74726163742f6576656e742f504b03041400080808001c841b
4d0000000000000000000000002e000000696f2f6e756c732f766f74652f636f6e74726163742f657
6656e742f4164644974656d4576656e742e636c617373a5565d6f1b45143de3ddf5dacec64e9cafa
6691af70370d64997967e918490d6b890d6498114d30225ddd8ab645bc70ef63a124248bcf1ce23c
fa579c90348241520219e82c4ffe05f20e0ded9b5b19d1415a16467eedc7b67e69c7beeaefcdb9f3ff
e0ce0024a318c221b858e3778c8f17083873739f0968ec5186ee2560c79dceac1125b37b1cccb655
e2eebb81d431cd908dee6f91d1ede8d6025823b11bc17412182f7d97757c73d1d1f0884b7ab9eb35
812e8cb3fb4b76dab6c57d6ad7cb5b23e4b31d7733639d6c346b65af19c8a27906ccb5cf16aaeccd
538a5de3aa6e1b9652befd63d0a4557dcf58aed356a8ec0d5aef05cdead5a9546b96e310eab4877d
4eca2676d564b4ed92a30363a77769ed1ccb915d79b17c8a5bba1fecb3a00d88d6ab220a066e912
8144dead38cb8dcd35a776c75e2b3b4cb05ab4cb05bbe6f23a70aade864bfccc670076b6a936d6b5
5289f1e6784190873b0ffa74ab79d8e6ffa7f0dc9563aad175c72b044227d39387a5e684c540ed786
0b7041fecd8d1923c12e4d5dbce6cd77de690f3bfa8ed7cd2b0cb74f4507ba96eaf3d748aa41e35ae
```

a8767662109242d904ba77c5b38b8f96ec2d59727a7108f0865ddff04557d293e408d79c7aa34cc9
82a35ed5a726105ba9366a45e786cb5af5b78b7a8e2f3470161f1a18c30903e33c9cc48481144e19
f808f705d2cfdb223a3e3670099775acea7860600eaf098c7497fa7ac32d979c1a89f499ffb2cea40cd
858e3a14860a752fe7b3a93a2d7cf5f04d2cda44e13377aa2bebbce29e273030b384fb9ddc51338d
e44de025d2f3db2245659b382af7ab66cd7499a818ebe904eaa7fbcb3b93aee691638ded917a484b
db5e554a8f9a6d3875bed70f7052599edca6fb6c733f3718a3ea5a320a250116205e9ab1b6211e54
c3aca99a4a49cd3d23e8308d967f102d92f92e72b9a359a8f9b4f21cccc1e42e6d41e14737a0faa39
aaee41fb8ea20a5ee21c84694cd05d7d88a29fbebb490cd0df280691a6c845ff244cc204a4c58884b
41853485a8c4a9116e3529121db08b252844f0d904e916f1ae768649c16cd9ca399df23f4ad4c604
861e91c91d7fb8768c1f582b61cb559e9de3c76e4e69771fe88cd6af7e68923375fc02b4179db376b
dd9bcf1cb1d9e77e917cf422f908c40879b8628fcdcc1368ea6ee60071731fe10c3ddf40537633bf40
5fe2c21c6090a7297af611f91a891d7a9fa49fb298bc9fa0904f694f509a09aa9fa0924fdd47b495a03
613b4030cf0443eed07c442d84182ed5fa1ab3b5095dd56b79ca4de0431d489d93152799cd49c27
4956a9285f507d997bcae7d5ea98c7b882abc47a1c5fe255f2853023e54de871f1074ee8980d8b11f
93f1c5be0cad2272690e9774ae6065bf5ebd06bca1af430a6b9be89b10d9f3dbb95c0fda0e9577d3f
11363afcc435614a9ebd027e64ec1f7ec3b2db2f51ff5f2676579025e4f709755ac6249480d700fd9e
99c7eb0494d9cc21d2bf20fec21042c48748908dc00e0951e85cdeed5cd2190badceacd22ddc3bb9
9f10bff7148964df3efa7ddec9e4806f2b6c0ffa3611ec4f0ef130ec3b340e8e90bd8f63ddcd39dfd69c
b956675f9359d7ff06504b07087c0491c885040000e1090000504b0304140008080800 1c841b4d00
0000000000000000000000031000000696f2f6e756c732f766f74652f636f6e74726163742f6576656
e742f566f7465437265617465457665 6e742e636c617373ad55df531b5514fe36d964937421107e9
502256dd142121a6dab5640c452da4203ad0dc6b65ae992acb01012cc6e98711c677cf3dd479fabb
cf06067848e3ae3f88433fe1ffe178e7aceddcd926c9219997120f79e7beeb9e77ee79cef9efde3ef9f
7f057015c50806712f0c05191e967958e1e13e6f3c50f07e040f918d6015d933f880a587c8f132c7cb
9c820f23e8c4bd101ef1fc988727217c14c2c7213c0de19310d658f74c81a6605d4270af6ce98b050
95d992d6d4f4b17b5d2463a532e6d4c4b08588655d425c4eab6b256c5109b724137f3646358fa8ee
91eaf5a46319d314c8b2cc25963a3a459d50ab9b8e1d99ec918e574a95a34d37c7f3a5f2e59152d6
fa577ca05bd98ce3126f23b3d4b6e823346c9b06625dc19f7426cc6d54e73826b2247d8e7e91a09d
18c51d257aa3beb7a65555bb7232de7b5624eab18bc7694b2b5695084936d20eb7b7ac91290e72
bba66e90bbc26dcfd8dbe3edfadf92bff1f71fce70472bce10dddca39758e8d4f34573a6c9e18f434c1
63172172b16ad3a1b7c1834b8890e95af48d371bb013859cdc12b451cc9ac48e176d0eb9d0ea593
4d5a43c05771893e3bc16562313a6bdda53e535a87f56d58aa627e0fbeb5b7a9edc3f9120951b5f8f
b32538a559123ab29696df5ed6760535e87513e44dcddcb4f9e91f9f58a44b2aba592d92b1448b90
55b6f32921922d572b79fdb6216ae2e1df15be53c518f22a8630ac6204e7558cf210c7051505d0a1e
42908ade05315ef6056c186824d15b7b02061c05be59b55a358d02b44a82fecbe32155761608b87
6d829c8a8b963215bf48e1d02f9c8a731b116b92452f998ad3de972aeee22ab5156fee249caba176
019b85edb4002952e69075bea899a2ecf564154a4a7f6723c31beea9e5b7b3911854086d77572fd1
0b996c45ef2695938b698f7d8d1d6ded7181dafd202850c8f071f5e8cbe0e3028a79d499a98c64735
1c897a0923c8657487e9534dfd01ca0f95ce225a444f210be44ea10fec4e421e4c4a07c88c00bdaf5
e332db20486394eeea4218ddf46d88a187fe06d18b71dab96e7bc20412809018912424c6e41312a
3f20b8971c94892ac3a5671c2273b4853a49bc4151a19679a66b609247e84ef0761c09082423920
aeb79d049ceb253af29a737896acf9f2a008f0857b3a22b44374665878e8b7addc00824e00125ea7

0f6e3310bf17c8684b20d728332d80f8bd402ed199b13640387f7cf11b2d81c85e20975b0279b3351
0d90b244967526d808c3a40dec20d8742f540025e20e91640ecfabe4dba294c3b4e1ed403aae39c
0de81a9dbdde0690cdee131605058b6a7c9f21897a911db634401ae6def344f27b04e483e4313a13
470826e9f71d02fe83e46f50969962c7e8e52945bf2384be45749f5a92d0931597de36f093ce7f84b0
6be0af19c8b6814c3ab9de40ae19048ed1c313e9023f21e2c33ea22cff0e45de87ec3f7073709e5e3
928dd0ae5ec2c656c4444348b35bc8baf3027f212b7e372f3f29cf6e628ea117c8df748e7c34df150a
24aa7f4178615cc07a501f1df1f99e3fa519776b8f1a7538a353b0f1d099183338c69a66b7468d38e9
ed514b9caea6735bd6ceb65af9e628d26449c1d12ec9da193f8fa45df58a04e729bea7807f3d4d09f
62b1aede6b4e5c3dc890cd1d02cad1cc20d43d27fd833ef8281e0a826438b24f92728dcb478d4bf27
1d77d4b16ddc2dc59fa059d8f5f221aeb3a42b71d772cd663cb146c77ac97873e5b21bb8a7e5b116
0eb01928f70d6fb0e56eadec192fb201785d5d2bf504b07080d0cf3d605050000d40b0000504b0304
1400080808001c841b4d000000000000000000000002b000000696f2f6e756c732f766f74652f636
f6e74726163742f6576656e742f566f74654576656e742e636c6173738d55df6fdb5414fe6e9cc449e
a266dda655bbb6e651b6beaa433bfc6a0cdbab2d24120dd80a24af0849358a957cf2e89530921fe0
6de7987bef40124d60926219e3a89bf0901dfb59dac758a4051ee3dfeeeb9e77ce73bc7c91f7ffdfa1
b80d7f0710e45dcce2285b7e4f2b67c5c56b192430d77c6b08a3bd2ba2badbb2ad672c8e17606efc
8fd9e5cd6337837830d69de57f19e8af705d2fb9e6fd5db02138d47e6be6938a6db311a9edb59115
06ddf7a5c6ff786877ddf768c86ddf37998ddb23baee9f7bb96c0d5d8712d1e6b9517d235dbb5fd558
16be5f8713cfae2b64072dd6b3374a161bbd683fee3a6d5fdd46c3a448a0daf653adb66d796cf1198f
4776cf25c68d89ee1f69d9e21cb325a9eeb77cd966f58fb96eb1bdbc436a44536a5d351beda1b44b
af55fecce284ef2cd762c7f3bd2b2585e1c55334787fa40d0a1c74949af8f80670b697dd9371d06397
792eac3e623ab45e93e17109e5469e4881cf7070a088c6ff9666b77d3dc0beae614096476ccde4e2
8bb525eac3353d7eaf51d3a0b3e647c6fcbefda6e4760fa547d212a2bdcf2fadd9675df963ae6876ad
f94ae1a6650d7308d731a4a38afe1037c28f0f2ff6a978a86863216556c6a30f08ac0f978f67b7ddb69
5b5d96f87538d0cbf31a1ee0a15c3e1218abce47b3bc3ccf6abed1f03adee058c735129819301a92e
9b5778d4833aac24e67d8c875c7ecb10353a7840840ea903fddb40130ec2a8535f7f62c9793b2541
e157254dba8ba9598ffa0e5ffea8f97f83351047922c10fc5e7ef4842ea4fec42605f449af60c66695f2
2b2cb5de15ed09f42e8952748e8d527507e22a4608e6b1149ae1ad7716491e7a780cb44e6c36bb8
420b8125d389c09209132423ef0d908b44420a57895dc375ae9280c15dfaa4f49f91f831709069d30
1580c52854152512a4e116e44ec4f5e56e2974b675c0e192c10e37c450c9e471234f5ca0f48250f2b
c7c8eb474856f8fd1e29e5b0f23b529b92de31a6e556e5f708e9ef503840ae1ae0f492148e3125376
2ca2f5013384041dacfa1260f90540e87aa96c8086c824a0617c8738eec56a9d80b659b43659bd05
121d7396ca04a2c8125a293c8aa79f1272ea9b89916a5dc9a54822f4b54d4b7c10450a190f7b81e7
0ce4816b5892bb33b21db821e30cd0ac8832f6ab32f18e6031637384a0bcc570e9895c29011b329
267a957f5322e07319c9c935f1b7ec3c19910c6d44764248727c0523729f308eec4af919729f3dc55
8513b1a30cc170ba1ad487b82f61126e39d5d3ad1d9f2702c6e055e6ffe03504b07084a0e9208a50
300003f070000504b0304140008080800 1c841b4d000000000000000000000002f000000696f2f6
e756c732f766f74652f636f6e74726163742f6576656e742f566f7465496e69744576656e742e636c6
173738d545d6f1347143de3afb5978d9d98240442202d25d86b936d530a6d9c8680f968a869a5a6
8a449fb271b6c9c2b29bdaebbc54fc84fe81beb779c9432b1523b512e28554ea6f428533b36b8738
2045d6deb973f6cedc73cfbdebfffefffb3980393474143197431a9f4a73456e3fd37055c7357caee30
b69ae615e7af31a6a3a74cc65b120d72fa559cce27a164bd2bda1e1a686ba406627089de50d81e1

c6437bc7b63cdbdfb41a81bf5913d0e5bb7ae0ffe86e0a941a6e60f91daf6d49d46a067ed8b29ba1f5
38d8703c6bb51fc9739905d777c345815a69f0d2e35e525e1548d5890a141aaeef7cd379bceeb4be
b7d73d22c546d0b4bd55bbe5ca7d0ca6c22db72d50794f0667c7f143956199dc6ecb1d99e6369d70
3516a0582a1f9560280ee8a96032e8f83a383f756c8fa4c6ded6e1dbf5874e33ac957f1010812ce6c
82b558c1d32fb4a68371fddb7b7558decb44076cb6e6f45c2244be565266939ed8ec760c14d360c5
6c296eb93e9e8a1722254f67425e8b49ace1d570979488f59196e6002b70c8c62ccc0384e19b88d3
b02e5638baae1ae810bf848c357064c54044e0db2b8d971bd0da745f57f8e866f7edac032ee49f3b5
40be3a7d3077f3d32cec898159581cd141a504cef488f539b5371e598a8912886394650beb9edd66
1f4e1ed2448194247fb8eb0233c7eb30b5b7b7b71d9fb373b97454eba3f2c785d706e27b03f1de787
cc0afbd08f247823ff6867f0709d91eae19e213384d7b86bb07c4935c0be63308b3f21409b3fa14c93
fd58149da2252b427680de430843c7f67894c47c730857380f2641aa13c992841ff3ca3a2341657f9
2e6dfe85c41ffdcb330a1c51171a51407ca1600d1fbee37072f0f0d83b0f73a0e2c32fe302d7cccaef4
8a7f62afbc89b5da42a7c7e433ab9577981f47d496b1fa372a9f2e922f32b0abbd0ab0a67944cbd8f1
1b9104b76a1eda220bd7fa1a576914aee314752f11a270f30b7c6dd04194d5187452a71a0da5a5fb
5355cc40c994ea18e4bc412282941725a5ebcc2590de58c18d59764fdfc36e2927e515d052e47ac8
74cc5382b592c0c9f9fdc8ab84a983c73125e5b983ce097571c2e72386678e525c56b3cba30e6759
269aadc09c5e61c52234be235954e900fa9d047ec2784a436ab5a24a97da7fa0e94fe81fee0194e1
48d6e8f5fbe58e8f648e58bc3f4bb1819ec66f5ad6e96faddfc58457df206504b07085258d8a46b030
000d8060000504b03040a00000800001c841b4d000000000000000000000001b000000696f2f6e
756c732f766f74652f636f6e74726163742f66756e632f504b03041400080808001c841b4d0000000
0000000000000002900000696f2f6e756c732f766f74652f636f6e74726163742f66756e632f426
17365566f74652e636c617373ad59097854d515fe6f6679339397041206 98b08445719230445023
04c5008284b52c82806d1d92210c2433711614ab7503b5ad4ba5b8e102c505ab6095c210a4a2ad
15ea566dddaab6b5d66a6dad7bad4b11fb9ffb2693493290f87d7e212ff7dd7beeb9e7fce79cff9ef7f1
c4a107f70118a3867b3019371ab8c9833cdce8c6246c94c7cd1edc825be5719b079bb059e67eeac
116dc9e8fe370870777e22e035b3d28c08df2b85b1e3f93c73df2b8d7c0360f4a44f1760fbc22e3151
92feecbc7cf71bfc83c606087fcfd85a8dd29a35da23b6560b70743d02ae7eef1f0dc073dd88b5fcada
4322b54f1e0f8b9a3b64e11179fc4ae67e6de0510f46e237b2e93199d92f5207e4d8df8ac58f1b7842
a49f34f0940727e06991f99d9b4e3e23363cebc1eff107997b4e46cf8be80b1ebc889744e11fe5f565
59a09a570cbc6ae04f0a45cde1c8bc507db431123e2f18a90f29f49db932b83a58d51c4caca89a146
eac8b24428da1d87805c7ea682214e7164b2099083755cd0ab670c53d3fdc18092692316eafebb8
7a92f5da148c3456cd8c461ac7cf0c47ab22c9a6789568abaa8f4612b1607da2aa39da106aaa5ac8
b929914438b166fc04eacd1719b12ed6c07363dd69eeb4dc7650e68c78c3aaaa890d0db1503c9e2
d3c331c4f74513641ff28384f0a47c289090a3e7f6e5cca172ad827d37a01261c09cd4e362f0bc5160
4973571a67866b43ed8b430180bcb7b7ad29e5811a63b471f0689e5c9487dd5a4603c2460d08282
f98960fd2a7aa4b7ebd0fd9976d5c742c104b505fd59a6cf4fc4c21924b26796769d2aef6924687138
116a5618d1ed8e3aca0968225fd7a0a0ea9836546261d1c5046a6e08c5eb29231b8849490e3b25f
1824d496a70cab1a2b657e768297856670c56f0f7dc35979c2c1990d19ac989f1520b1de2b7a6a52
d8663bbe44f8fa09930dec05f0c040dbc46f231f057c196f9a530d23fbd5b1593a391e561066e09819
d2e29a0df7be26e7a274f13096ef54f5f3a5df4b856a74da34afeba5604e353ced568283925ac6058b
16470f296f2508ff56a2166c4da6a7358f7f5a430f71baf5003af931269552cd4106a6e4984a311fae8
d7ae19cb824d16a319f5c1c842edb7fbec6428b6c61a5752ace7795294d9392f144f36d1f962d9df9

909177499fc06bc645de8d3d37ba605e39607c3b27226a7720d43d07a51187c64610533d9d2404e
21df24927149c99ec25327245810d7fb649a654baca391a63573ce898462564828e2991f4dc6ea43
53c35242056d1c374a5c36310b7f53e8dd0e07dd5c41fc4c9c86692666601a79cfc41bf8bb8937f19
6817f98781bff343105530dfccbc432bc63e2df7897b52562ef91af7a66be89f7f181890ff191898fe5f1
1fcc26e4ed964c8cc5826bac8c3faa27656ee2137cc4e236f15f7c6ae2337c4eb63f02d9cb3e0b7413
5fe07f260ee24b1387f095c91254ccd4dc7b43ab43918455defa2e9822efc451e599caa6ec2636e03
a1688a91cca692a43b94ce5c6bba6f2a87c5399325720af85aa48a1bcbb23eac85299037a99b81ce
b4cd55b158be21253f5515e53f5c5a70c6b87ec37553fe53cacf7edea33aafb9bcac740aa524c3571
0dbe34d4001397c9cc4082a20661b6a906ab32430d31d5505566e25aac37d53055a630bc07f7295
9bebdc0e62c5b19aa6750fdddc446aefad8f2a050499f5c3d4007a5d6ada550d8b190db263295de8
3f49c9ce6779b5f8ac7d0b7e09ce5d28ab4914cd726cd5d1f6d6e09c6420ba2476859782b97e6e48
2d3695d5cf3fad9c9b0747376ff1239bb7f4ee959711ad7df7f585becf1f079216d3e0f2cc9189d7d6d
3be2a184dce825fece6b72ac8bab0baceec19ba3cdd1a850e454dd431471d4b193f570861826985
ab4823cd8498d157fcd9272525dba07f177ee03e41c3705da88d1eed784e78c87220d12fd21fef2ee
a855f4a7b970a8bf1bd2a66a5b4b92268fcd616c0ef3bb4e299cd605cf9e7489b91cb7879aa53f293
b8cd5ba70b5a0a37e45a87e950e3701b5358624c573c19dcb5e77633bbcaec64c2c8afde55d9bb2
5ee1f82cdec0e196a6d0fc5093aee0dedc322b78aef53a399a9480bbade8eb0aaaec718b249e8cef
0aded7d82daeb49d5be1efbec5c8f4665ea6552e6c1c8d568d64c0c82e20272b3528355b14d657ef
e4606456b421bc9c5df0f0c3e440c7f81635762e1bfb9229f3e6280c3c52965a514a67b48bcb91f872
199a561c632cdb66abf4a71f8e6b263545255ddc094ab275686e61c172f7944883ecc5507eb24e86
c2a9b0f1c39e973cbff6edd20b70ae8ee33c4ce7fb8cacf799c8e7781666f3398733ebf54ee0b88add
50157b91b778376cbb606f1f3a2addad70a660dc06c3be15765b2b5c1595bbe07e80db6cf8169fa5
70f239884795a20803f81c8c00ca30865ff873b9d2d73a02f3301fd02331958d20c7bd91f715ca9167
e074030b815a5ee75884332cfbf20652b58762ef05f6c30ceccbb2a1727ffb7864d6b8159e1abbcf5e
b1136e1a7d7396d13b61df837cc6eeac75ad306b1c7b5140170b6b9c3ea7cf914291cf5999422f9f7
3640abd39654fa1782f4a28d2a7c6b055bbbcaec0be2d18b3175e4ef5ad71fbdc5ed73a4f90aa52e8
e77307bcae3129f4f7193ef71ef8f2b068ad4b6dfdea692a325228f5396d290cf4395b312885c1628a
cfe173ee41990d8bf662c8629fa372a4cfd88da1ad18e673deafe328d02e437f3e8f224c4713e01144
ca8f517c4e4225c10ce04c54a11ec7228cd18812f0d5381e17a21a3fc0898ced586cc238dc8d1aece
05c2b4ec6c39880c7700a9ec644bccadc7987a190102d27caeb19aec55802372ea1f452ea7609f29
9b0bd876fe33b8c8b8cbecb519e1e9d8520d360141ea1ad5369f57cdc4c8b1ae0e0fef50851b3931
645d1881530682760528735b392339204d5f014d42a8ffe7710fd9808b5ea2bfae8d659b14aff6be2
ac8166031103bcb7d9049e403d2d38dbca1415e1d99286dbcbdc1bb2d22490951a1565bb313c90c
251e96c2863ecf6e0e83cda5e203993252ad11fb1b1e3fb31350e9f633f73c3b1077e85ec435228bf
093efea9482fde9ebd95f3953e3b350cdc8b918be5d4c06e041869fbb64ca027b072c0722d64b996
b240cbe8d40886662c019d82d3f9b388809e813803741e83f36306e07a86e116c27f2f0197201e4fd
009402660db391fd301dbce7d0986692c923a2476ea9acbd0ade08eb69058332b3923212943415b
3cca8979ad2a94e140625fab06d4aa3c6e4832dbac2a7d860a0b1883691565298cb250ef58adb9e
04ea14a10d5d87903fb6454918d9bac6c44efc03efbe6ce8170762dccb358805552d76e5bb5c7569
defcd97fa6228b630284e6ffe1e1c2b277b6b4c1f0b76b48fec30e626b8edd59ead70afcd67ad3eebf
56c845b0eb16deb54c69b68bf23cbfee36a5c3ed70189a766c91a57dbbacf65d5b4f62e85e337921
1484983f4be03596ec8a4cf6823801374560801547 74c8bcbc8f160985c0c930fab58f54dacf2082b3

a4a2e3f9ba18ab1ee925c5dcdd1b9381f6bb08e0972054737e002dc89efb3f62fc42e5cc4fa3f1ffbb0
168f52e200353f4ea9e7b8e765fc883c70253ec055f81c57f303e85ad507eb5539ae53d5b8414dc54
d3abdd632c677a204e730054dee398e7bcfe47db28eb7c95256bc9bfa66f0fcf3c82549be7fcf62103
5ad2d21394a272447e7d33a1b2af9b56225e4187e1a49924a42bea653d349d91769bff0c63be49e
0b798fb974ba0ee14e4b6a6546eaa28cd4c59492141e87626fadfa12a335675cc2a4edcddf432866
120b871c829784a2d4c82fe0f07c86a283a83270693ab9d7d22d8b586a689cd4d5ad1d89459349e
7b49e9d4d372cfb11c62694e8a2d8d8697e7387f71377606cc72bb615e352a8913291e58e371967
c65bb32705f4c6629d31d6655c2d9548722ac466c66a0bef8e3b783fdcc5d2df4afcee2185dc8b661
24212f7f196d8c10fe09d64ec948e70055da5a31902b9351dafa3185589928dbbfccc1ce1f9769238
88219a21fa12b9fc5a552233a59ac3b9f5f20c8e23d2385ed009c72e101e80c3b62d20ec40eff58b5
4d3c9ad98b0e15678029a982d891164067745c0b11ba7c89b63539b70ad08df8292740828c3d23a
c5b62d0ba7e928e6732f717a887e3c8ce13cfc68d64625efc763b19f55f63889f70976534fd2fca748c
0cf30bb9e25e53ec73a7c9eb9fa0251798959fe4a16761764b0bb208d5d1989dac2ae926b92e1167
6fde0d1d8f988dd41984cd06255e0f60966573030567bf6601ab371dd61d631ed8ea9b1070486d24
07b27613569afd1dfd769d31bb4e64df60c6fb14f783bcbfe7119fbc765ec2f4ddb3f9a75257569cf5c
19d6cccab44705b06b8f06e86ce0f20f491279da8f87d27ed47e0d3fba502eb3a1dd9dbe5adfbb74e
77d9af821ddf998edcd2759aed4665ca9cdb83234edcac9fc153a6977c59ab938b72b56ef7c25c5ae
c2d5e97476a6fb8df011fa8d23f7185d1cd4ed45d6fd17d02b6d53ed97c260521d48d5856c824a711
0237108e3c8aa33c85c67287b562f10ce6010cef402615cc3eec1c65d27a7a97706350a1aedbd803
57371ba1718a0535503324803a2ff5e5aabc8a7c4e55af67956ba4ee39b8d7f9d9503d9ec3ca0511
383e5bf1da15c70f06e999be66f4a658c736a43941efd84b7461e7f36e0bab4d68d34d1fef58b60bce
eb527f2a5fd4b45678d3251a80a51a67af1fa29c658559265d4910a2067925fafbdbce1ff504b07088
d9e92e9580d00006c1d0000504b03041400080808001c841b4d00000000000000000000000002e0
00000696f2f6e756c732f766f74652f636f6e74726163742f66756e632f566f7465496e74657266616
3652e636c6173738d50cd4e023118fc8a2bcbe20f08fe9d3dadd1d8c42bc4c4188d9235266238c8a
9760b292e2d76bb24bc9a071fc087327e5b086220d1ddc3b4d3997ed3f9fc7aff00807338f461df870
302456e04b382000ba3011b339a30d5a76d6ba4ea379699ee32751c494d5596a474acada05c2b6
b18b774a86391d00e72d7ca4a3b6910f0a49296c069d8fad373a5554fe2e5cfe8ca3598346c755bf9
dee74c751ce5858e281b118be1c84aad08046f993093e9f9099eff3f5d65ee7c14699660d05aee770
fceac4ce83d1ba12c68cbbe62363338e06949d09ccf9b8f4ae3577a19c746a46963411cc9d43617e
a8c349679813f81ba4b32f3dcb274fa9aa385da565eeeba68ebcc7071231374d472e39db2c2f4181
767f92c02e1ea427a99e2f4979e40f527ddc3cb40705b2440a000f957f208ac810780b80e45873e94
1c065076b831c34dd872b80d15f457d15d801ddcd7a08e6b02bb8ed9fb06504b07082c00d50e580
100009e020000504b03041400080808001c841b4d0000000000000000000000002b000000696f2f
6e756c732f766f74652f636f6e74726163742f66756e632f566f74655374617475732e636c6173735d
905d4a033114854f3a6d476bffd42717e0ab017730142b01e90849c7c71243949621814ed2c5f9d0
05b828f1662c74f0ed7e5f4eb827f9fef93a0278c46d8eeb1c370c73a90ab5969bb742a88d5809c5c
004c364e15d13b40b95aea3cde80efe65ab523d25cf18a6279f94583d27db63189fec6bb1966d323
bbbc54bf9e7fa0c23e9e3ded8e5b6b60cb3ca072b830eb179d8e98366b8df7aee62ddf0039d70e35
dd86b13f84774869fc3542ec579addd272fdf77d68421bd043dda31c01d32f48906a91986c479872f
882f3b3c22beeaf09878d2e169fb19b3769eff02504b07081b9fa402de00000052010000504b03040
a00000800001c841b4d000000000000000000000001c000000696f2f6e756c732f766f74652f636f

6e747261637 42f6d6f64656c2f504b0304140008080800 1c841b4d00000000000000000000000002c
000000696f2f6e756c732f766f74652f636f6e74726163742f6d6f64656c2f566f7465436f6e6669672e
636c6173738d56df731355143e9bfd91346cd326502a4da10920a469208a284a11910aa525e587c
10215846db2b44b37d99a6c181dc607c717ff007c601c47c719f5056774c61647671c9f9cd177ff1b4
7fcceddcd265952c6e9ecbde79c7bcf3ddff9ceb937fdebdf5f7e23a223b41aa7519a1da0013ac7c31
c0ff33c9ce7a114a705bac05b2e46e952942ec7294eb3317a9be7320f5762f44e8c1659bc1aa36b3c
5f8fd1528cde65f146946e4a34d0748d867bc5aa991249f31245cd7ad5d386ade642cb76ad75db2c
9bb65971b16149a244cdf8c0d3679c569d8d73120dde735c73c6a82f3855ebce87126927acbae59e
9448cdcdcf4f2e4aa4cc38559c3954b2eae685566dd96c5c31966d585225a762d88b46c362dd372a
eeaad5942857b29c62bd65378b7c7cb1e2d4dd8651718b359c65171739a453bf63ad4c0336e22cc
d71a498909658542bab66654d22393709e08365d7a8ac2d18eb7e107dc574cb9de4b109e9ebcd1
ea39213e8e3d87aa6cd4bbcd9a524a184595272227aa216b22771ca42883c449df34f09ad283991
ce90d55cece576189b4326cd7cbf65d8606c2457ba6bdc338ab6515f295e5cbe8be3a63977c961a6
9f5a124c1b2eda07bcad1acd55af4a5ac36c023b6cae53761b567d45a21db9c92e7fcf0aff78d96935
2ae6598b191dea94e430efd52943efe9f41cedd2698c87348debb49bf6e83441e351baa5d36d3aa4
9341cb121df87fb59668340ce374cbb2ab6603c5bb1f34f3f18c4e15aaf20060db0a19bfad8f67904a
21136e6d6ffb1d9054c8f4f6b7b782b0c942a6a7c9f924e9239d66e82dd4244cada82cca37d6ce2a4
8a8595d2b9eb61deecb01178800b9b60eaad11c33b6d14411b7f7502d8c605a33d6d79184448772
4f17e2e9daf8a4c0711c3dfccce5a5672fcf6dbd4c59bc43a378ab548a70992145b8d26246b1c58c7
a8b7942e84992d015598c7ba15d2719be4443f9c724e527362892cf6e90fca370d88731450ac6418
c09841ac2384cfb61c9c0116ef43c1d201212879784c4006411e6a01fe663e851cce9de30f9117583
94fc88b6416a27e418697ee0ed08b903214730ee847d54843e2a10a783d0e92074da0fcd1267efed
e3fc350127e7c3f90c7b6298b35bc3c98f443748eb80da23f0ef02a831804a03d43840ed86b607074
f08602705f06c002c1b00cb06c0b201b0ac0f8c252e4d143b2729ef43bc8f48ec7938ff13290f691093
aa7c4daafc0852049ffce0735f937fa6e8832f58511ec143eec2bb0fd5d88f921f44ac1ca24ce1fc02fe
8a02afee45f0f14e89369293836c2ed0211f481104f03695e37e1ff0a109e34b5de7a8fe39124e2cfac
e278186b3d604c91d36e398898ec1e75571c24ea16b01739acf9c442ff4052287814cf705f2627f20
7218c81bf039b50590311f483a284d51e81e10c178379099be408e80a936908817229f46af8581cc
c2e7dc16408ea24738f0cbf44a1f206a18c8f9be408e81f03e40d430908bf0b9b405106e5b0efc5a5f
46b43090725f20c7fb33a285815c85cfb52d804cf88c4cd309ffacbf716bb8ccd5fcd4b7b82f8fa6fea0
447e936253f8bee11b32f53b0d2c702717f03d78d8be4105be4f6d4d81a67ce92b2a14b5ad6850b4
af28aa7c478adcb96b7bc563720337ee26faf6160832709b96415085ce52150c985d6f6635c8a04a
af237b09fb67d18207901fdfc1240d4413d23f948ad2a9415d4f9ce214df0c8afe271267222efbef406
438f3d1279f9e189e48af7226fc09cbedb609af870738b0a881a475d6d29d7cbc07f82edeb835bc4d
362a5503d23a9ac2e9aac4653f8f047afeb46809469f27e5092e6d04e0815b7ac2e5147244c23faf3
d2a3cf0dbeda7754ffc6210957ea5f8f5c7b42da56f8a472fb24989d49027cb2c0f7bb2b249c954ca93
d54dda9edae1c91adb47206fd2ce1f429de876756229e8c43362d7d9ff00504b07080d0cf59679050
000f50b0000504b0304140008080800 1c841b4d00000000000000000000002c000000696f2f6e7
56c732f766f74652f636f6e74726163742f6d6f64656c2f566f7465456e746974792e636c617373955
76b731355187e36b7dd94d096b694d682a48818d24b4051a1ad58282091722d968b37b6c9922e4
d9392dd8014d17abfe21d2fa838cea87ce9a8cc481875c6f113cef8c5bfe237bf38eafbeec96e93eda
6d41972ced9f77dce39cf3eef65cbeffffcf40b80bb71a50e31e4c258823c0f533c9ce6a1c083c183c94

39187330c3e2be3a93a9cc3741dcef3f0340f17787806d34bf02cafce6186bd33ec9d61ef0c7b67d83
b23e3b93ab421a7e0799e5fe0e145052f297859c12b0a5e55f01adb5e57f0868237155ce4a7b7787
85bc63b127c7a5a42e3f029f58c9ac8aab94c62389fcbf44b089aba99d5243455b846cc826e390369
cd484908a5f2b9937a46426c58cf2772c5ac91389337b50499cd829a321393f9b4964d8c926dc842
d2d69061aa66d1902025e992fcd99c569070bbb3dfd96aa42712dbd2e9826618b42b52d052f94c4e
9f56732922d52a484daae67862bb9e49e64c2da31598b56e6a9386f3424553cf268675c324577844
cfe4e8ea02eddfec720fdc927f92ceeddfcafc07f49c6e6e95e08fad1f2525860822a16158cf69fb8a93
635ae1b03a2674cba7d4eca85ad0f9b96c0c98e3bab118b976e648fe73fc4219cd4c52849a62eb3d
6264086773cced636a0a6d3d2c82d852b5db09a3623888e5b1f9003e44a6437658c1960d7b1526d
b5039f2713a79d1b10f1b731bbb169d33f6bb2445681d252a83db37cff83f42ca32940fb7959c3b9b6
f6fad8ee5b9293b9efd6ef8a26fe56359c79172355036258540b625104bda6fbe5f5449b442eb1a75c
2af524677c61606f3e10d74f8a1aaca6ab3859c5f5b0d861bdc1ef3c6f2d121ed7451cd1aaeccda3f7
64a4b91a8c7a9fef3d5cda5ecb28a4435252c252152137bd5294b6b6a93f476e3aa312e0a2e44ef5
0cc124c31f3225d25d48de48b8594b64be7d034cc95512f5f12411ffa23588f78045de88ea087875e
2422b80ff747b0011b23d4c1ef896013ee8de05dbc2761dde2ca54c6fb118c2125e3031997647c28
e323191f47a0e3888415eebada5ed4b3698e50f0bc9eee8b46f0092ef3f029bd4177d46abb7dd135
2410fdc2dd516eb5d6333945bfed8b5a0fa2918a033eb3a05637656f6377b4b261b289dc566f14f8cf
2398c02e02baf517093794550dab1a2adb8665a4f8d457f79aaa43ec50dc5239bb0dac5c2847a971
79e5974dc1a93a4a07756a4acb512becf1ea63f34ce510f4bbf076762e805f4955b9a07ba8b61b9df
4d58fd11f0a0a7c9c88b4f2712e5a734f79a68cb466ca476ba694b466ca4a6ba664057dc0acf516b4
d09af29ac601b2246896680ec66f40ba66411ea0316419156ca5312200781083344bd886edf336ff0
0dff7aecd2d9e9b87b0a3bc792ba17d8c8e775d876feeea3acbda467bdaad135a05aa7c02af5807b
e7827767910f1bb89acf424f210767b11f1bb8944694f670d221c08be38e94924e026b2d693c8c3de
44026e2231dab3be06919e32913d18f620127413e9f624b217fbbc8804dd4436d09e8d3588f4962f
de8f03e4731309b9896cf22022f2f420d90e61a47cc8814a422126e4af20b499f66ea94148e4fe61e7
16b66c71aae1115a8de2888766f2ac8bea80a76647716c9e661dd721bb351ba43ddb6a50e4b2e58
b8fe3510f228a5bb31d9e441ec3e35ec153dc4476d39e640d22dc37f8e227f0a40791b09bc8b0279
11350bd8884dd440ed09e83358870e3e28be90329ce92be248c9fe69678d737080666bb6ea23e5e
425d17fdbe46d03fdbf52b96ece5b875d3ef0bb6704fba89169ec8e62b2172190d57e92b68d905c02
f007eb2f94b58ea00fc3620200001b2052a01011b10148020d98225d43b80a00d500440219b5242
8303506c405800c2640b97d0e800c236207413cd3c912df42396f970150dbcfe0d72e02a02fe59a7
1c7ae963014a6a9992ba8d727315e5549cb26a0fa5479612e42285f65b0aee1f14a53f294eac7e54
e86aab4fab3434527d15fec249b2f990217327dae57ae96f2c9331be2224b9feb5d60d72b874bb922
4b31cfa6b22084be356009a98ed40e3ea8e71213d9b49f666369fb0ed01610fb8ed416127915baae
cb2b3520482545e5e85080b3b89db5a65275d1be296a62b24084fc79c96eb10a6f1249ae9fdbb304
e1f2e9d943c45da4cd07fa7b3b884497c875c45065f2b6bd88c2b84e39a66e574d42f1b94fec572f8
483b9289d628af7d9234bae0e38905bd47ab1fe9c209e743f41551e2c29cfe196dc76ea0bde9b612
3a44185636ad126bd2bea3e9761e560b43c031448521c8e84eb1964b58d37487582b6c5f2bd6615
edf29d6215a375e28615d0977b9db8551d12ea69d7691b55093ff01504b070828022e5591060000
0f110000504b03041400080808001c841b4d0000000000000000000002a000000696f2f6e756c
732f766f74652f636f6e74726163742f6d6f64656c2f566f74654974656d2e636c6173738d935b4f134
114c7ffd3db96755ba05c54102d88587a61bde1051051bca1451f30187d62291b585dbad8ddf262fc

087e01df95171e34d19a68627c11133f9351cf996eb12c3531cd9e993973ce9cdffccff4c7af4f5f009c
c13d155d186f4314e7d95ce0e5450597544c6052c5149b095ce6d96505d32a548cc77185c719365
7e3b816c76c1cd7797543c14d8190b522d0517c626c1aba6d9457f5a2535e9d14504a4ed933cb9e
40aa6973c1ab58723b3665952d6f5a209c195d1488cc3a2ba6407bd12a9bf7aaebcb66e581b16c9
b9cec940c7bd1a858bcf69d116fcd7205468a96a397abb6ab6f3a9ea973c18a51f2f4753acbd617c9
37e799eb542cba6a7a738499ca8cee078dbaf5cdae4c708fc1544a9d6ddca47b4ffeee5d54b729a62
7b33f840f8a99cfaa86ed0602ee2f3f314bdee4e86301e1ec95cadf12886ffa3711482c7846e9e9bcb
12185a0ced1ee9ae1aed5d52329e7a850c574ab3691085ac43da7ce40940b4eb552326f5aac60a2
a1ce1817d47008873574a347432f0e6ab885db02c3ffa3ae82390d2791517047c318748183c1eb5f
ab5af68a5921a59f5b2b13690d775164332fa0e5d3fe3399480f11317f2f347aa9c3f4a68252d06db8
19b6e1badcade65648274995dcdbc13d87347488191b1b66991a5e68d5a97d2e9f7e3210df68dc3f
e33148ffb02efadb85e847caca19894b631c820527db472b9d46416334fb11e29d0ceb271b93ce30
8e90d5ea0118c0511a058e21bd2ff93d426f03c96d2d930731e4274f4b368acee63e20f4b7b42abd0
9ca49ca137aeb51fe093ce3fb70e1e3186e01120e8274b6043981915620e120480fe5f4fe038405e5
c2f402fdb3be514c98c6a56cee0da291eddc0e92d91a2239fa5e231adece7d45749ee5da41370f79f
a6a88bd42fb16d4bcf453145f63079d3c902f5c83b285769e7d8712d94224bc2dbbd32fa9a264fba0
d0ea10b10d507fa6a9434c9caeb3ec122f611459221dc02c72e40b212ff5695392e2278e2828c444
b73ac3571a939af2955efaf214ead489ac248e33c554c7b1feb53a2bbb89b38ddd4b53fd7ff99292e1
043dc611aa79b249c982cfd545654ee13495639aa38874ce88dfa47b88780885e6f0e721c1686776
dbfe90cee1ae163e437df41107525aadc1974cb5d71a508954079b4e3235a482cf23dff43c0abbcfe
3ac8c3af707504b0708bd3eec285a030000bd060000504b0304140008080800 1c841b4d00000000
0000000000000000028000000696f2f6e756c732f766f74652f636f6e74726163742f566f7465436f6e
74726163742e636c61737395557b571b4514ff6d5e4b92059a1650a82db4a535e1d1283eaa80541
a5b9b180a028d026a1d36933034d9c5dd4d14bf865fc33fea397a28f61c3f801fcae39dd96d884978
7872327b67f6dedffdddc7dcfdfb9f3fff023087e7095c4651c74a024fb09a40086b71a4f06502ebd848
62134f759412e8c35772f3b55cb6e4b22d971db97c23976fe5f29d8e673abed7c134f4ed3297976c8
f6b982a0a3b6b356a6eb649fbac695b9ec34c2f5b69586656aae42d8f3b1566f2050db14561096f49
43245dc894e891b3cb84315814167fd2a8ef726793edd6e8e472d13659adc41c21f7c161c4db13ae
86c9533c4a67b96043be06ebc25ae7a65db5c4cfcc32c95e2b10f36707ecd0870ba7258598e97026
0361e9e23e6bb26c8d59d5ec86e708abbad07db2d37d9439854f9d62ab29560f2d4f7887c4294a4f
3fbc2e140aafcc5d937484c7eb14e6951eae34249a2d3c0de98b7b1e5d6f9058e725e10a8a7ed9b2
6c8f79c2b6c8d349002d04b7fc3ccb5a3ad9353f6792a42ca08678ba50286ce7b733db9441e93c5fa
643d7638eb7495e34e8dc2afbd225e1ae346a9e38a8f10d5ee326596b643550673ff9fb9c4dd4e830
afa15f42e598b5629745e55016c7b62aa27a9148734a53326caabe8ca50b3b05494f97f9cc9729ce
d00e3540c2e1655e3f9061f96d28554c66f9dd7ce7dc549404ff91bcc47f6870e7d0379a26948b576
2b065b9ce5d4a0c7583b457b56e78a2965d6107d2c186a85acc6b38e460b34b61b137cfe572d9e1
aebbd0a65c14aeb7d8d649459bfa68897ed4618a4960f398b97e3437d385b3c155c698bfd170fd6c
654af886dd704cfe48c8c64fb55fd2bb929581114c68b875c61479100c1b1dbb064c94a5093730846
11d1503554ce8d833700337a98017eb1309210ccc60dfc047f8d8c0223e31b024974fb16ce001720
61ee1734ac7f9c3867afc24bfabbbfbaac7aff54ccb89c99032a9336f2ffb4054e590ac728712db64b5
065fadd08d6d15bd4349dee674ef3772a08df574fc947a81aa157378933be43e7dd1a94d2333ed5f
75c29e69eb8d732e62669b0a92a2cf0ff519fd43b25cf44d4a903c8237687d93766b88224ccfa1a92

36853af10da1a7f89f011227f20fa1b9d87304aeb80d21946842caf90ed18ed46e81dd9e12ade0294
740dd70955c3382602f439fa6bf48fbc44ec042da12cc708f3aa42327c9d0049936d441ad2be4aa75
17aa6a67e47747a66f6187a04f391d1c88b0eb071a238a1c0def70d5ab452b88549e52085db24859
474076f937b19561a19723245a731d29a560466301b04f00be2484afc57e8dbba31121dd687fb86e
34788cf2724a5f1d1c4311211fcda416792b27e5bd159a74fbba4f79ace38eeaa2c49294b525849ef
901455d2bb94335d49ef5128beed07741627c47e7c887baa7e7461028205221f913954742841c97
63631e529a3984c285f468b89d16262601e0bf456937730c0bd4f7ba99f54b8c730c25db0b36d6d9
06cc12603584ddee5a08c9d60fddd60736782bd2ecf7d551e9a0fa7000f84f1a203f8deff00a699437
0bd8007bb8117cf04fe4c693fec7040f32c60de55b74bdd755b3eb76e8f91a7b7fff55150285ffc0b50
4b0708c90dc6d686040000820a0000504b01020a000a00000800001c841b4d0000000000000000
0000000003000400000000000000000000000000000696f2ffeca0000504b01020a000a0000080
0001c841b4d000000000000000000000000800000000000000000000000025000000696f2f6e
756c732f504b01020a000a00000800001c841b4d00000000000000000000000d0000000000000
00000000000004b000000696f2f6e756c732f766f74652f504b01020a000a00000800001c841b4d00
0000000000000000000001600000000000000000000000076000000696f2f6e756c732f766f74
652f636f6e74726163742f504b01020a000a00000800001c841b4d0000000000000000000000001c
0000000000000000000000000aa000000696f2f6e756c732f766f74652f636f6e74726163742f657
6656e742f504b010214001400080808001c841b4d7c0491c885040000e10900002e000000000000
00000000000000e4000000696f2f6e756c732f766f74652f636f6e74726163742f6576656e742f4164
644974656d4576656e742e636c617373504b010214001400080808001c841b4d0d0cf3d60505000
0d40b000031000000000000000000000000c5050000696f2f6e756c732f766f74652f636f6e7472
6163742f6576656e742f566f7465437265617465457665 6e742e636c617373504b0102140014000
80808001c841b4d4a0e9208a50300003f0700002b000000000000000000000000290b0000696f
2f6e756c732f766f74652f636f6e74726163742f6576656e742f566f74654576656e742e636c617373
504b010214001400080808001c841b4d5258d8a46b030000d80600002f000000000000000000
000000270f0000696f2f6e756c732f766f74652f636f6e74726163742f6576656e742f566f7465496e6
97744576656e742e636c617373504b01020a000a00000800001c841b4d0000000000000000000000
0001b000000000000000000000000000ef120000696f2f6e756c732f766f74652f636f6e7472616374
2f66756e632f504b010214001400080808001c841b4d8d9e92e9580d00006c1d000029000000000
000000000000000028130000696f2f6e756c732f766f74652f636f6e74726163742f66756e632f426
17365566f74652e636c617373504b010214001400080808001c841b4d2c00d50e580100009e0200
002e000000000000000000000000d7200000696f2f6e756c732f766f74652f636f6e74726163742
f66756e632f566f7465496e746572666163652e636c617373504b010214001400080808001c841b4
d1b9fa402de000005201000002b000000000000000000000008b220000696f2f6e756c732f766
f74652f636f6e74726163742f66756e632f566f74655374617475732e636c617373504b01020a000a
00000800001c841b4d0000000000000000000001c0000000000000000000000000c2230000
696f2f6e756c732f766f74652f636f6e74726163742f6d6f64656c2f504b010214001400080808001c8
41b4d0d0cf59679050000f50b00002c0000000000000000000000000fc230000696f2f6e756c732f
766f74652f636f6e74726163742f6d6f64656c2f566f7465436f6e6669672e636c617373504b010214
001400080808001c841b4d28022e55910600000f1100002c0000000000000000000000000cf290
000696f2f6e756c732f766f74652f636f6e74726163742f6d6f64656c2f566f7465456e746974792e63
6c617373504b010214001400080808001c841b4dbd3eec285a030000bd0600002a000000000000

```
00000000000000ba300000696f2f6e756c732f766f74652f636f6e74726163742f6d6f64656c2f566f74654974656d2e636c617373504b0102140014000808001c841b4dc90dc6d686040000820a00002800000000000000000000000006c340000696f2f6e756c732f766f74652f636f6e74726163742f566f7465436f6e74726163742e636c617373504b05060000000012001200a6050000483900000000";

        String param = "{\"sender\": \"" + address + "\", \"gasLimit\": 80000, \"price\": 25, \"password\": \"" + password + "\", \"contractCode\": \"" + contractCode + "\", \"remark\": \"" + remark + "\", \"args\": [\"10000000000\"]}";

        String url = "http://" + IP + ":8001/api/contract/create";


        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(3L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private static void transfer() {
        String from = "Nsdz8mKKFMehRDVRZFyXNuuenugUYM7M";
        String to = "Nse3Uaj7Lesh6VNBVJ62bZRRZRpZ4DAG";
        String password = "";
        String remark = "test";
        String param = "{\"address\": \"" + from + "\", \"toAddress\": \"" + to + "\", \"password\": \"" + password + "\", \"amount\": 100000 , \"remark\": \"" + remark + "\"}";

        String url = "http://" + IP + ":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(100L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private static void callPay() {
        String address = "Nse3Uaj7Lesh6VNBVJ62bZRRZRpZ4DAG";
        String contractAddress = "NseMYnJhdt8inCg8LgveLZMgUAXMGFqH";
        String password = "";
        String remark = "test";
        String methodName = "multy";

        String param = "{\"sender\": \"" + address + "\", \"gasLimit\": 200000, \"price\": 1, \"password\":
\"" + password + "\", \"contractAddress\": \"" + contractAddress + "\", \"remark\": \"" + remark + "\",
\"methodName\": \"" + methodName + "\", \"value\" : 10000000000 , \"args\": []}";

        String url = "http://" + IP + ":8001/api/contract/call";


        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(800L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private static void createPay() {
        String address = "Nse3Uaj7Lesh6VNBVJ62bZRRZRpZ4DAG";
```

```
        String password = "";
        String remark = "test";
        String contractCode =
"504b030414000808080038b2d4d0000000000000000000000090004004d4554412d494e462f
feca00000300504b0708000000000200000000000000504b030414000808080038b2d4d000000
0000000000000000140000004d4554412d494e462f4d414e49464553542e4d46f34dcccb4c4b2
d2ed10d4b2d2acecccfb35230d433e0e5722e4a4d2c494dd175aa040958e819c41b9a982868f817
2526e7a42a38e71715e417259600d56bf272f1720100504b07089e7c765344000000045000000504
b03040a0000080000fb8a2d4d0000000000000000000001b000000746573746f6e74726163
742f6d756c74797472616e736665722f504b030414000808080fb8a2d4d0000000000000000000
000003200000074657374636f6e74726163742f6d756c74797472616e736665722f546573744d75
6c74795472616e736665722e636c6173739d56eb7313d715ffad257997454e8cea00f210aa262d91
658cb00c989a94c498180cb6f143d8d86d4256d25a5a4bda95b52b83d31749daa4e933e9234ddf8f
90a68fb40d2d96dd64265f3ac34cfb0ff463ff857eea97cc747acedd95ac48824e6a8ff69e7b9ebf73ef
3967f7efff79e73d00c7f08e8a53585590df85fb505050e4d5e48725a3a442c62a3fd6149479b51538
2cacc85857710dd75584b0a1e219d8323eabe073bcff7c00fcf7daab3758f005667d51c18d1af75905
cf0a5a1a1c8082e75cfa6f7fbda1e079155fc29715bcc0be5f64435ba5685f51f09282af32f36b8ce2eb
0abec19b6ff2e65b0a5ee6cd2b0abeade03b2abe8beff1e35505dfe7f535053f50f043267fa4e0c7327
e22e3a712fca656d425842657b5752d5ed0cc6c7cde291b66f694844e7ba398b20a12bac62cd3763
4d359d00a15d256327ada286a055b8234e123d82109b256284c982b1649af96b40d2d5520455fb4
6f81628c5919dadc3f6998fa74a598d2cb49571c9ab4d25a61412b1bbcf7987e276790e323938e6e
3b69cb74ca5ada89172b05678348d35ed1cbf12489a69893f43884363c57311da3a82f18b6418e46
4dd37234c720e412fa260d2b6e560a76bceed0cee4e35a5d273ee342e6ac1f354cc3394d1965756
75a9c4e4fb4afddf9ec228579ef8876137db67e2a94f7043948696490d6f924c98293dba5653265dd
b6af0e4a38d81ed4a8ab41fe0322e906a344033d44072ae4e356d9339170337a6f9f6e1645cdc9c5
cf18d909d3d1b37476ff974dcb79b4d76b7b707e4a62904b8696aba626616f7b635733e16a268466
17514394b2ebcb950cb99279474be7a7b49257479d5aa9a49b19090f445b3170592aae825e96a0c
e5b95725a1f37d8706f4b751d61fb20a6312e614f8b34884f81caa523990ce2319c0e62146724ec6b
8e79a6621432acade367748f9181c8e8e2e0f1939783f83966242088b3ece6f087297b19bf08e2977
83d88246eca7843c2817bdda584c8b49d71b2b3e746af5f99c91d1fb296af8f2eae2e94ccf3e6c5e16
2e5d8b525ce7225885fe1cd207e8ddf1050af884722c25a3fbe9c3766edc1d9c152e60967eee2b52
bf913d3f91396b554494c26bc001b99e4c2a29d9f4a0f9fcb2f3f93b2d712e3d6586eede2f8b5a9150
9fb3d3491b4e8bf484a8fe8c592b37144c66f83f81ddea2a0aa2d6e662412c4ef393d1d64a8aa5ef13
316455de761c41a7fc04dd6f8236b6435bba0af38cc7e1bb798fd276a4e6697ca465aa8ff19b7a843
555595d0ada6e848cf14ac74febc66e74622326e07b1c986556cb161da32cc94660bc36dbcced072
ba91cd398c21a8f2c0a1c9582cb1fc2fec5862b73b777f29b5aaa71d090fb6bd97316f2361a05d91b6
b0bc1aa2c6501cabd602fbdbba9eb2bdeef02e3e7a9719b8336ef6d7a65c6b1bf6b4e3937b710597
e86ac2d10b7735566af5cb6a779912d48ebd6de15d760c1ea872595fab1865eecee872fb76eeb4bd
660eacbb6fa90f1ca97b0ff73c52d9ab1d31be2f502bd5736a7f03b59aba1b74515684cb14af3cee24
66e4a8ce7826b5f1cd4e6bf546eaf5dac2c7e8e57f0a121e851f1d3c6df895cbb386788f13dd41fc108
f9dfa7e8c7e6785de6ee23d8171da9fa35d3f7f63f0ef96503b4fcf4ec18863023c838410177091d649
4f3a257834fde8c94e36e0137ac3b12d48b150c7267cb1907f1381587768139def425eda82125bbe
```

0d5f15bb422a3daad82db4829be8da097d90be5b804142ff085444d18d3eec450247318413f455b6
0369d88324e112663c18715a59168851a0b79bf2196e300ed48d67ebc6039eb12fe46f363dd560ea
ab9bce61bed9b43bf45693e9636d4d93f5a8c73c53751bf757d15dc59ee6e0630d1ed4ba87cb750f
05f2db41ebd177115a0a7d640b3d93fd3d78601bfbaad8ef9d7c284c275e8fe09efe4e9cfbc05f6ce7
c8fb79f450348eb7d7f5e9c55371080b581458af5064b774b6c9ce4feb950f44eead4526de01e24d1
deec1830dbc8f126f7aa007910f81d02d8b4b84638610ced17e9e8a23498571995a6051208eb958e
a88a73dc407318225a23a48fb109689f235d5f1a7f1192fa37f924ca175a6ff0eba07ee20180edc81e
c7f137e5fe8a16d3cdc7f98100f84fd5e2ee100e5322287e570e7ffcec42732e9a5b8a078fbf0240ee0
293c84ab543e4f531d68228b3937be9705534f8a2c987a8a743b04f5b4c882298d787e41a5a8b903
824a13af93bc3e2cb295911115dfb1e771a929f1b3b5229266c90907793f56cbe2e3228b4f5471c84
d610b8fd465d15a866d647d4216aba2bf4970580806aa38d224880bc1d11d4168b0519c10e2a16d
1c6b0d765cc84eb4061b768d5a839d14824f36041be1e76d74ed545b82ba0d58a1fbc9e22472342
30cacd2fff3c8e325eab73750c47b30f10f58f8174af837d61a7af47defee3af813c5ebd139af477b1b2
6611721e9af0dc28611a80acd0ac2586fe8c4de7a45f412aad3a21373c2c6f82f504b07087c257e1c
05070000b30e0000504b0102140014000808080038b2d4d0000000002000000000000000900040
0000000000000000000000000000004d4554412d494e462ffeca0000504b0102140014000808080
0038b2d4d9e7c765344000000450000001400000000000000000000000003d0000004d4554412
d494e462f4d414e49464553542e4d46504b01020a000a0000080000fb8a2d4d0000000000000000
000000001b00000000000000000000000c30000074657374636f6e74726163742f6d756c747
97472616e736665722f504b0102140014000808080fb8a2d4d7c257e1c05070000b30e00003200
00000000000000000000000fc00000074657374636f6e74726163742f6d756c74797472616e7366
65722f546573744d756c74795472616e736665722e636c617373504b0506000000000400040026 0
10000610800000000";

String param = "{\"sender\": \"" + address + "\", \"gasLimit\": 80000, \"price\": 1, \"password\": \"" + password + "\", \"contractCode\": \"" + contractCode + "\", \"remark\": \"" + remark + "\", \"args\": []}";

String url = "http://" + IP + ":8001/api/contract/create";

```
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(3000L);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
```

```
        }
    }
}
```

60:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\constant\ContractStorageConstant.java

```
 */

package io.nuls.contract.storage.constant;


/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/5/24
 */
public interface ContractStorageConstant {

    String DB_NAME_CONTRACT_LEDGER_TX_INDEX = "contract_ledger_tx_index";
    String DB_NAME_CONTRACT_LEDGER_UTXO = "contract_ledger_utxo";
    String DB_NAME_CONTRACT_ADDRESS = "contract_address";
    String DB_NAME_CONTRACT_SPECIAL_TX = "contract_special_tx";
    String DB_NAME_CONTRACT_EXECUTE_RESULT = "contract_execute_result";
    String DB_NAME_CONTRACT_COLLECTION = "contract_collection";

    String DB_NAME_CONTRACT_NRC20_TOKEN_TRANSFER =
"contract_nrc20_token_transfer";
}
```

61:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\po\ContractAddressInfoPo.java

```
package io.nuls.contract.storage.po;

import io.nuls.contract.util.ContractUtil;

import java.math.BigInteger;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/15
 */
```

```
public class ContractAddressInfoPo {

    private byte[] contractAddress;
    private byte[] sender;
    private byte[] createTxHash;
    private long createTime;
    private long blockHeight;
    private boolean acceptDirectTransfer;
    private boolean isNrc20;
    private String nrc20TokenName;
    private String nrc20TokenSymbol;
    private long decimals;
    private BigInteger totalSupply;

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

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

    public byte[] getSender() {
        return sender;
    }

    public void setSender(byte[] sender) {
        this.sender = sender;
    }

    public byte[] getCreateTxHash() {
        return createTxHash;
    }

    public void setCreateTxHash(byte[] createTxHash) {
        this.createTxHash = createTxHash;
    }

    public long getCreateTime() {
        return createTime;
    }
```

```
public void setCreateTime(long createTime) {
    this.createTime = createTime;
}

public long getBlockHeight() {
    return blockHeight;
}

public void setBlockHeight(long blockHeight) {
    this.blockHeight = blockHeight;
}

public boolean isAcceptDirectTransfer() {
    return acceptDirectTransfer;
}

public void setAcceptDirectTransfer(boolean acceptDirectTransfer) {
    this.acceptDirectTransfer = acceptDirectTransfer;
}

public boolean isNrc20() {
    return isNrc20;
}

public void setNrc20(boolean nrc20) {
    isNrc20 = nrc20;
}

public String getNrc20TokenName() {
    return nrc20TokenName;
}

public void setNrc20TokenName(String nrc20TokenName) {
    this.nrc20TokenName = nrc20TokenName;
}

public String getNrc20TokenSymbol() {
    return nrc20TokenSymbol;
}

public void setNrc20TokenSymbol(String nrc20TokenSymbol) {
    this.nrc20TokenSymbol = nrc20TokenSymbol;
```

```
    }

    public long getDecimals() {
        return decimals;
    }

    public void setDecimals(long decimals) {
        this.decimals = decimals;
    }

    public BigInteger getTotalSupply() {
        return totalSupply;
    }

    public void setTotalSupply(BigInteger totalSupply) {
        this.totalSupply = totalSupply;
    }

    public boolean isLock() {
        return ContractUtil.isLockContract(this.blockHeight);
    }

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

62:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
storage\src\main\java\io\nuls\contract\storage\po\ContractCollectionInfoPo.java
 */
package io.nuls.contract.storage.po;

import java.util.Map;
import java.util.Set;

/**
 * @desription:
```

```
 * @author: PierreLuo
 * @date: 2018/8/15
 */
public class ContractCollectionInfoPo {

    private String contractAddress;
    private byte[] creater;
    private Map<String, String> collectorMap;
    private long createTime;
    private long blockHeight;

    public String getContractAddress() {
        return contractAddress;
    }

    public void setContractAddress(String contractAddress) {
        this.contractAddress = contractAddress;
    }

    public byte[] getCreater() {
        return creater;
    }

    public void setCreater(byte[] creater) {
        this.creater = creater;
    }

    public Map<String, String> getCollectorMap() {
        return collectorMap;
    }

    public void setCollectorMap(Map<String, String> collectorMap) {
        this.collectorMap = collectorMap;
    }

    public long getCreateTime() {
        return createTime;
    }

    public void setCreateTime(long createTime) {
        this.createTime = createTime;
    }
```

```
    public long getBlockHeight() {
        return blockHeight;
    }

    public void setBlockHeight(long blockHeight) {
        this.blockHeight = blockHeight;
    }

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

63:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
storage\src\main\java\io\nuls\contract\storage\po\TransactionInfoPo.java
 */

package io.nuls.contract.storage.po;

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.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

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

/**
 * @author: PierreLuo
 * @date: 2018/7/23
 */
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();
    }

    /**
     * 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);
    // blockHeight
    size += SerializeUtils.sizeOfUint32();
    size += SerializeUtils.sizeOfUint48();
    size += SerializeUtils.sizeOfBytes(addresses);
    // txType
    size += SerializeUtils.sizeOfUint16();
    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;
}

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

```
        return 0;
    }
}
```

64:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\ContractAddressStorageService.java

```
package io.nuls.contract.storage.service;

import io.nuls.account.model.Account;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;

import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/5/24
 */
public interface ContractAddressStorageService {
    /**
     * hash
     *
     * @param account
     * @param hash
     * @return
     */
    Result saveContractAddress(byte[] contractAddressBytes, ContractAddressInfoPo info);

    /**
     *  - hash
     *
     * @param contractAddressBytes
     * @return
     */
    Result<ContractAddressInfoPo> getContractAddressInfo(byte[] contractAddressBytes);

    /**
     *
     *
```

```
     * @param contractAddressBytes
     * @return
     */
    Result deleteContractAddress(byte[] contractAddressBytes);

    /**
     *
     *
     * @param contractAddressBytes
     * @return
     */
    boolean isExistContractAddress(byte[] contractAddressBytes);

    /**
     *
     *
     * @return
     */
    Result<List<ContractAddressInfoPo>> getContractInfoList(byte[] creater);

    /**
     *
     *
     * @return
     */
    Result<List<ContractAddressInfoPo>> getAllContractInfoList();

    /**
     * Nrc20
     *
     * @return
     */
    Result<List<ContractAddressInfoPo>> getAllNrc20ContractInfoList();
}
```

65:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\ContractCollectionStorageService.java

```
package io.nuls.contract.storage.service;

import io.nuls.contract.storage.po.ContractCollectionInfoPo;
import io.nuls.kernel.model.Result;
```

```
import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/15
 */
public interface ContractCollectionStorageService {
    /**
     *
     *
     * @param contractAddressBytes
     * @param contractCollectionPo
     * @return
     */
    Result saveContractAddress(byte[] contractAddressBytes, ContractCollectionInfoPo
contractCollectionPo);

    /**
     *
     *
     * @param contractAddressBytes
     * @return
     */
    Result<ContractCollectionInfoPo> getContractAddress(byte[] contractAddressBytes);

    /**
     *
     *
     * @param contractAddressBytes
     * @return
     */
    Result deleteContractAddress(byte[] contractAddressBytes);

    /**
     *
     *
     * @return
     */
    Result<List<ContractCollectionInfoPo>> getContractAddressList();
}
```

66:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\ContractExecuteResultStorageService.java
```
package io.nuls.contract.storage.service;

import io.nuls.contract.dto.ContractResult;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/24
 */
public interface ContractExecuteResultStorageService {
    /**
     *
     *
     * @param hash
     * @param result
     * @return
     */
    Result saveContractExecuteResult(NulsDigestData hash, ContractResult result);

    /**
     *
     *
     * @param hash
     * @return
     */
    Result deleteContractExecuteResult(NulsDigestData hash);

    /**
     *
     *
     * @param hash
     * @return
     */
    boolean isExistContractExecuteResult(NulsDigestData hash);

    /**
```

```
     *
     *
     * @param hash
     * @return
     */
    public ContractResult getContractExecuteResult(NulsDigestData hash);

}
```

67:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\ContractTokenTransferStorageService.java

```
 */
package io.nuls.contract.storage.service;

import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/28
 */
public interface ContractTokenTransferStorageService {

    Result saveTokenTransferInfo(byte[] key, ContractTokenTransferInfoPo tx);

    Result deleteTokenTransferInfo(byte[] infoKey);

    Result<ContractTokenTransferInfoPo> getTokenTransferInfo(byte[] infoKey);

    List<ContractTokenTransferInfoPo> getTokenTransferInfoListByAddress(byte[] address);

    List<ContractTokenTransferInfoPo> getTokenTransferInfoListByAddress(byte[] address, byte[] txHash);
}
```

68:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\ContractTransactionInfoStorageService.java

```
 */
package io.nuls.contract.storage.service;
```

```
import io.nuls.contract.storage.po.TransactionInfoPo;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Result;

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

/**
 * author Facjas
 * date 2018/5/22.
 */
public interface ContractTransactionInfoStorageService {

    Result saveTransactionInfo(byte[] key, TransactionInfoPo tx) throws IOException;

    Result deleteTransactionInfo(byte[] infoKey);

    Result<byte[]> getTransactionInfo(byte[] infoKey);

    List<TransactionInfoPo> getTransactionInfoListByAddress(byte[] address) throws
NulsException;
}

69:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
storage\src\main\java\io\nuls\contract\storage\service\ContractTransferTransactionStorageService
.java
 */
package io.nuls.contract.storage.service;

import io.nuls.contract.entity.tx.ContractTransferTransaction;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;

import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/16
```

```
 */
public interface ContractTransferTransactionStorageService {

    Result saveContractTransferTx(NulsDigestData hash, Transaction tx);

    Result deleteContractTransferTx(NulsDigestData hash);

    Result<ContractTransferTransaction> getContractTransferTx(NulsDigestData hash);

    List<ContractTransferTransaction> loadAllContractTransferTxList() throws NulsException;

}
```

70:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\ContractUtxoStorageService.java

```
package io.nuls.contract.storage.service;

import io.nuls.db.model.Entry;
import io.nuls.db.service.BatchOperation;
import io.nuls.kernel.model.Result;

import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
public interface ContractUtxoStorageService {

    byte[] getUTXO(byte[] key);

    List<Entry<byte[], byte[]>> loadAllCoinList();

    Result batchSaveAndDeleteUTXO(List<Entry<byte[], byte[]>> utxosToSave, List<byte[]> utxosToDelete);

    BatchOperation createBatchOperation();
}
```

71:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\impl\ContractAddressStorageServiceImpl.ja

va
```
package io.nuls.contract.storage.service.impl;

import io.nuls.contract.storage.constant.ContractStorageConstant;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.model.Entry;
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.NulsDigestData;
import io.nuls.kernel.model.Result;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/5/24
 */
@Component
public class ContractAddressStorageServiceImpl implements ContractAddressStorageService,
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(ContractStorageConstant.DB_NAME_CONTRACT_ADDRESS);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveContractAddress(byte[] contractAddressBytes, ContractAddressInfoPo info) {
        if (contractAddressBytes == null || info == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        Result result =
dbService.putModel(ContractStorageConstant.DB_NAME_CONTRACT_ADDRESS,
contractAddressBytes, info);
        return result;
    }

    @Override
    public Result<ContractAddressInfoPo> getContractAddressInfo(byte[] contractAddressBytes) {
        if (contractAddressBytes == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        ContractAddressInfoPo infoPo =
dbService.getModel(ContractStorageConstant.DB_NAME_CONTRACT_ADDRESS,
contractAddressBytes, ContractAddressInfoPo.class);
        if(infoPo != null) {
            infoPo.setContractAddress(contractAddressBytes);
        }
        return Result.getSuccess().setData(infoPo);
    }

    @Override
    public Result deleteContractAddress(byte[] contractAddressBytes) {
        if (contractAddressBytes == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        Result result =
dbService.delete(ContractStorageConstant.DB_NAME_CONTRACT_ADDRESS,
```

```
contractAddressBytes);
        return result;
    }

    @Override
    public boolean isExistContractAddress(byte[] contractAddressBytes) {
        if (contractAddressBytes == null) {
            return false;
        }
        byte[] contract =
dbService.get(ContractStorageConstant.DB_NAME_CONTRACT_ADDRESS,
contractAddressBytes);
        if(contract == null) {
            return false;
        }
        return true;
    }

    @Override
    public Result<List<ContractAddressInfoPo>> getContractInfoList(byte[] creater) {
        List<Entry<byte[], ContractAddressInfoPo>> list =
dbService.entryList(ContractStorageConstant.DB_NAME_CONTRACT_ADDRESS,
ContractAddressInfoPo.class);
        if(list == null || list.size() ==0) {
            return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND);
        }
        List<ContractAddressInfoPo> resultList = new ArrayList<>();
        ContractAddressInfoPo po;
        for(Entry<byte[], ContractAddressInfoPo> entry : list) {
            po = entry.getValue();
            if(Arrays.equals(creater, po.getSender())) {
                po.setContractAddress(entry.getKey());
                resultList.add(po);
            }
        }
        Result<List<ContractAddressInfoPo>> result = Result.getSuccess();
        result.setData(resultList);
        return result;
    }

    @Override
    public Result<List<ContractAddressInfoPo>> getAllContractInfoList() {
```

```
        List<Entry<byte[], ContractAddressInfoPo>> list =
dbService.entryList(ContractStorageConstant.DB_NAME_CONTRACT_ADDRESS,
ContractAddressInfoPo.class);
        if(list == null || list.size() ==0) {
            return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND);
        }
        List<ContractAddressInfoPo> resultList = new ArrayList<>();
        ContractAddressInfoPo po;
        for(Entry<byte[], ContractAddressInfoPo> entry : list) {
            po = entry.getValue();
            po.setContractAddress(entry.getKey());
            resultList.add(po);
        }
        Result<List<ContractAddressInfoPo>> result = Result.getSuccess();
        result.setData(resultList);
        return result;
    }

    @Override
    public Result<List<ContractAddressInfoPo>> getAllNrc20ContractInfoList() {
        Result<List<ContractAddressInfoPo>> allContractInfoListResult = getAllContractInfoList();
        if(allContractInfoListResult.isFailed()) {
            return allContractInfoListResult;
        }
        List<ContractAddressInfoPo> resultList = new ArrayList<>();
        List<ContractAddressInfoPo> contractAddressInfoPoList =
allContractInfoListResult.getData();
        for(ContractAddressInfoPo po : contractAddressInfoPoList) {
            if(po.isNrc20()) {
                resultList.add(po);
            }
        }
        return Result.getSuccess().setData(resultList);
    }
}
```

72:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\impl\ContractCollectionStorageServiceImpl.java

```
package io.nuls.contract.storage.service.impl;

import io.nuls.contract.storage.constant.ContractStorageConstant;
```

```
import io.nuls.contract.storage.po.ContractCollectionInfoPo;
import io.nuls.contract.storage.service.ContractCollectionStorageService;
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.Result;

import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/5/24
 */
@Component
public class ContractCollectionStorageServiceImpl implements ContractCollectionStorageService,
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(ContractStorageConstant.DB_NAME_CONTRACT_COLLECTION);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }
```

```
    @Override
    public Result saveContractAddress(byte[] contractAddressBytes, ContractCollectionInfoPo
contractCollectionPo) {
        if (contractAddressBytes == null || contractCollectionPo == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        Result result =
dbService.putModel(ContractStorageConstant.DB_NAME_CONTRACT_COLLECTION,
contractAddressBytes, contractCollectionPo);
        return result;
    }

    @Override
    public Result<ContractCollectionInfoPo> getContractAddress(byte[] contractAddressBytes) {
        if (contractAddressBytes == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        ContractCollectionInfoPo po =
dbService.getModel(ContractStorageConstant.DB_NAME_CONTRACT_COLLECTION,
contractAddressBytes, ContractCollectionInfoPo.class);
        return Result.getSuccess().setData(po);
    }

    @Override
    public Result deleteContractAddress(byte[] contractAddressBytes) {
        if (contractAddressBytes == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        Result result =
dbService.delete(ContractStorageConstant.DB_NAME_CONTRACT_COLLECTION,
contractAddressBytes);
        return result;
    }

    @Override
    public Result<List<ContractCollectionInfoPo>> getContractAddressList() {
        List<ContractCollectionInfoPo> list =
dbService.values(ContractStorageConstant.DB_NAME_CONTRACT_COLLECTION,
ContractCollectionInfoPo.class);
        if(list == null || list.size() ==0) {
            return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND);
```

```
        }
        Result<List<ContractCollectionInfoPo>> result = Result.getSuccess();
        result.setData(list);
        return result;
    }
}
```

73:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\impl\ContractExecuteResultStorageServiceImpl.java

```
package io.nuls.contract.storage.service.impl;

import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.storage.constant.ContractStorageConstant;
import io.nuls.contract.storage.service.ContractExecuteResultStorageService;
import io.nuls.core.tools.log.Log;
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.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;

import java.io.IOException;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/24
 */
@Component
public class ContractExecuteResultStorageServiceImpl implements ContractExecuteResultStorageService, 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(ContractStorageConstant.DB_NAME_CONTRACT_EXECUTE_RESULT);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveContractExecuteResult(NulsDigestData hash, ContractResult executeResult)
{
        Result result;
        try {
            result =
dbService.putModel(ContractStorageConstant.DB_NAME_CONTRACT_EXECUTE_RESULT,
hash.getDigestBytes(), executeResult);
        } catch (Exception e) {
            Log.error("save contract execute result error", e);
            return Result.getFailed();
        }
        return result;
    }

    @Override
    public Result deleteContractExecuteResult(NulsDigestData hash) {
        try {
            return
dbService.delete(ContractStorageConstant.DB_NAME_CONTRACT_EXECUTE_RESULT,
hash.getDigestBytes());
        } catch (Exception e) {
            Log.error("delete contract execute result error", e);
            return Result.getFailed();
        }
    }
```

```
    @Override
    public boolean isExistContractExecuteResult(NulsDigestData hash) {
        if (hash == null) {
            return false;
        }
        byte[] contractExecuteResult = new byte[0];
        try {
            contractExecuteResult =
dbService.get(ContractStorageConstant.DB_NAME_CONTRACT_EXECUTE_RESULT,
hash.getDigestBytes());
        } catch (Exception e) {
            Log.error("check contract execute result error", e);
            return false;
        }
        if(contractExecuteResult == null) {
            return false;
        }
        return true;
    }

    @Override
    public ContractResult getContractExecuteResult(NulsDigestData hash) {
        if(hash == null) {
            return null;
        }
        try {
            return
dbService.getModel(ContractStorageConstant.DB_NAME_CONTRACT_EXECUTE_RESULT,
hash.getDigestBytes(), ContractResult.class);
        } catch (Exception e) {
            Log.error("get contract execute result error", e);
            return null;
        }
    }
}

74:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
storage\src\main\java\io\nuls\contract\storage\service\impl\ContractTokenTransferStorageServiceI
mpl.java
 */
package io.nuls.contract.storage.service.impl;
```

```
import io.nuls.contract.storage.constant.ContractStorageConstant;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.storage.service.ContractTokenTransferStorageService;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;
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.ArrayList;
import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
@Component
public class ContractTokenTransferStorageServiceImpl implements
ContractTokenTransferStorageService, InitializingBean {
    @Autowired
    private DBService dbService;
    private String area;

    @Override
    public void afterPropertiesSet() {
        this.area =
ContractStorageConstant.DB_NAME_CONTRACT_NRC20_TOKEN_TRANSFER;
        Result result = dbService.createArea(this.area);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveTokenTransferInfo(byte[] infoKey, ContractTokenTransferInfoPo infoPo) {
        return dbService.putModel(this.area, infoKey, infoPo);
    }
```

```
    @Override
    public List<ContractTokenTransferInfoPo> getTokenTransferInfoListByAddress(byte[] address)
{
        List<ContractTokenTransferInfoPo> infoPoList = new ArrayList<>();
        List<byte[]> keyList = dbService.keyList(this.area);
        if (keyList == null || keyList.isEmpty()) {
            return infoPoList;
        }

        ContractTokenTransferInfoPo tokenTransferInfoPo;
        for (byte[] key : keyList) {
            if (isAddressEquals(key, address)) {
                tokenTransferInfoPo = dbService.getModel(this.area, key,
ContractTokenTransferInfoPo.class);
                infoPoList.add(tokenTransferInfoPo);
            }
        }
        return infoPoList;
    }

    private boolean isAddressEquals(byte[] key, byte[] address) {
        int length = Address.ADDRESS_LENGTH;
        for(int i = 0; i < length; i++) {
            if(key[i] != address[i]) {
                return false;
            }
        }
        return true;
    }

    @Override
    public List<ContractTokenTransferInfoPo> getTokenTransferInfoListByAddress(byte[] address,
byte[] txHash) {
        List<ContractTokenTransferInfoPo> infoPoList = new ArrayList<>();
        List<byte[]> keyList = dbService.keyList(this.area);
        if (keyList == null || keyList.isEmpty()) {
            return infoPoList;
        }

        ContractTokenTransferInfoPo tokenTransferInfoPo;
        for (byte[] key : keyList) {
```

```
            if (isAddressAndHashEquals(key, address, txHash)) {
                tokenTransferInfoPo = dbService.getModel(this.area, key,
ContractTokenTransferInfoPo.class);
                infoPoList.add(tokenTransferInfoPo);
            }
        }
        return infoPoList;
    }

    private boolean isAddressAndHashEquals(byte[] key, byte[] address, byte[] txHash) {
        int length = Address.ADDRESS_LENGTH + txHash.length;
        for(int i = 0, k = 0; i < length; i++) {
            if(i < Address.ADDRESS_LENGTH) {
                if(key[i] != address[i]) {
                    return false;
                }
            } else {
                if(key[i] != txHash[k++]) {
                    return false;
                }
            }
        }
        return true;
    }

    @Override
    public Result deleteTokenTransferInfo(byte[] infoKey) {
        return dbService.delete(this.area, infoKey);
    }

    @Override
    public Result<ContractTokenTransferInfoPo> getTokenTransferInfo(byte[] infoKey) {
        ContractTokenTransferInfoPo tokenTransferInfoPo = dbService.getModel(this.area, infoKey,
ContractTokenTransferInfoPo.class);
        Result<ContractTokenTransferInfoPo> result = Result.getSuccess();
        result.setData(tokenTransferInfoPo);
        return result;
    }
}
```

75:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\impl\ContractTransactionInfoStorageService

```
Impl.java
 */
package io.nuls.contract.storage.service.impl;

import io.nuls.contract.storage.constant.ContractStorageConstant;
import io.nuls.contract.storage.po.TransactionInfoPo;
import io.nuls.contract.storage.service.ContractTransactionInfoStorageService;
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.annotation.Service;
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 io.nuls.kernel.utils.NulsByteBuffer;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
@Component
public class ContractTransactionInfoStorageServiceImpl implements
ContractTransactionInfoStorageService, InitializingBean {
    @Autowired
    private DBService dbService;

    @Override
    public void afterPropertiesSet() throws NulsException {
        Result result =
dbService.createArea(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_TX_INDEX);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
```

```
        }
    }

    @Override
    public Result saveTransactionInfo(byte[] infoKey, TransactionInfoPo infoPo) throws IOException
{
        return
dbService.put(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_TX_INDEX, infoKey,
infoPo.serialize());
    }

    @Override
    public List<TransactionInfoPo> getTransactionInfoListByAddress(byte[] address) throws
NulsException {
        List<TransactionInfoPo> infoPoList = new ArrayList<>();
        List<byte[]> keyList =
dbService.keyList(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_TX_INDEX);
        if (keyList == null || keyList.isEmpty()) {
            return infoPoList;
        }

        byte[] addressKey = new byte[Address.ADDRESS_LENGTH];
        TransactionInfoPo transactionInfoPo;
        byte[] values;
        for (byte[] key : keyList) {
            System.arraycopy(key, 0, addressKey, 0, Address.ADDRESS_LENGTH);
            if (Arrays.equals(addressKey, address)) {
                values =
dbService.get(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_TX_INDEX, key);
                transactionInfoPo = new TransactionInfoPo();
                transactionInfoPo.parse(values, 0);
                infoPoList.add(transactionInfoPo);
            }
        }
        return infoPoList;
    }

    @Override
    public Result deleteTransactionInfo(byte[] infoKey) {
        return
dbService.delete(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_TX_INDEX,
infoKey);
```

```
    }

    @Override
    public Result<byte[]> getTransactionInfo(byte[] infoKey) {
        byte[] txInfoBytes =
dbService.get(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_TX_INDEX, infoKey);
        Result<byte[]> result = Result.getSuccess();
        result.setData(txInfoBytes);
        return result;
    }
}

76:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
storage\src\main\java\io\nuls\contract\storage\service\impl\ContractTransferTransactionStorageIm
pl.java
 */
package io.nuls.contract.storage.service.impl;

import io.nuls.contract.entity.tx.ContractTransferTransaction;
import io.nuls.contract.storage.constant.ContractStorageConstant;
import io.nuls.contract.storage.service.ContractTransferTransactionStorageService;
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 java.util.ArrayList;
import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/11
 */
```

```
@Component
public class ContractTransferTransactionStorageImpl implements
ContractTransferTransactionStorageService, InitializingBean {

    @Autowired
    private DBService dbService;

    @Override
    public void afterPropertiesSet() throws NulsException {
        Result result =
dbService.createArea(ContractStorageConstant.DB_NAME_CONTRACT_SPECIAL_TX);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveContractTransferTx(NulsDigestData hash, Transaction tx) {
        Result result;
        try {
            result = dbService.put(ContractStorageConstant.DB_NAME_CONTRACT_SPECIAL_TX,
hash.serialize(), tx.serialize());
        } catch (Exception e) {
            Log.error("save contract transfer Tx error", e);
            return Result.getFailed();
        }
        return result;
    }

    @Override
    public Result deleteContractTransferTx(NulsDigestData hash) {
        try {
            return dbService.delete(ContractStorageConstant.DB_NAME_CONTRACT_SPECIAL_TX,
hash.serialize());
        } catch (Exception e) {
            Log.error("delete contract transfer Tx error", e);
            return Result.getFailed();
        }
    }

    @Override
    public Result<ContractTransferTransaction> getContractTransferTx(NulsDigestData hash) {
```

```
        try {
            byte[] txBytes =
dbService.get(ContractStorageConstant.DB_NAME_CONTRACT_SPECIAL_TX, hash.serialize());
            if (txBytes == null) {
                return Result.getSuccess();
            }
            ContractTransferTransaction contractTransferTransaction = new
ContractTransferTransaction();
            contractTransferTransaction.parse(txBytes, 0);
            return Result.getSuccess().setData(contractTransferTransaction);
        } catch (Exception e) {
            Log.error(e);
            return Result.getFailed();
        }
    }

    @Override
    public List<ContractTransferTransaction> loadAllContractTransferTxList() throws NulsException
{
        List<ContractTransferTransaction> txList = new ArrayList<>();
        List<Entry<byte[], byte[]>> entryList =
dbService.entryList(ContractStorageConstant.DB_NAME_CONTRACT_SPECIAL_TX);
        if (entryList == null || entryList.isEmpty()) {
            return txList;
        }

        ContractTransferTransaction tx;
        for (Entry<byte[], byte[]> entry : entryList) {
            tx = new ContractTransferTransaction();
            tx.parse(entry.getValue(), 0);
            txList.add(tx);
        }
        return txList;
    }
}
```

77:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-storage\src\main\java\io\nuls\contract\storage\service\impl\ContractUtxoStorageServiceImpl.java

```
package io.nuls.contract.storage.service.impl;

import io.nuls.contract.storage.constant.ContractStorageConstant;
import io.nuls.contract.storage.service.ContractUtxoStorageService;
```

```
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.Result;

import java.util.ArrayList;
import java.util.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/5
 */
@Component
public class ContractUtxoStorageServiceImpl implements ContractUtxoStorageService,
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(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_UTXO);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }
```

```
    @Override
    public List<Entry<byte[], byte[]>> loadAllCoinList() {
        List<Entry<byte[], byte[]>> coinList =
dbService.entryList(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_UTXO);
        return coinList;
    }

    @Override
    public byte[] getUTXO(byte[] key) {
        if(key == null) {
            return null;
        }
        return dbService.get(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_UTXO,
key);
    }

    @Override
    public Result<List<Entry<byte[], byte[]>>> batchSaveAndDeleteUTXO(List<Entry<byte[],
byte[]>> utxosToSave, List<byte[]> utxosToDelete) {
        BatchOperation batch =
dbService.createWriteBatch(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_UTXO)
;
        List<Entry<byte[], byte[]>> deleteUtxoEntryList = new ArrayList<>();
        byte[] deleteUtxo;
        if(utxosToDelete != null) {
            for (byte[] key : utxosToDelete) {
                /*deleteUtxo = getUTXO(key);
                // UTXO
                if(deleteUtxo != null) {
                    deleteUtxoEntryList.add(new Entry<byte[], byte[]>(key, deleteUtxo));
                }*/
                batch.delete(key);
            }
        }

        if(utxosToSave != null) {
            for(Entry<byte[], byte[]> entry : utxosToSave) {
                batch.put(entry.getKey(), entry.getValue());
            }
        }
        Result batchResult = batch.executeBatch();
```

```
        if (batchResult.isFailed()) {
            return batchResult;
        }
        return Result.getSuccess().setData(deleteUtxoEntryList);
    }

    @Override
    public BatchOperation createBatchOperation() {
        return
dbService.createWriteBatch(ContractStorageConstant.DB_NAME_CONTRACT_LEDGER_UTXO)
;
    }

}
```

78:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-tx\src\main\java\io\nuls\contract\entity\tx\processor\CallContractTxProcessor.java

```
package io.nuls.contract.entity.tx.processor;

import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.entity.tx.CallContractTransaction;
import io.nuls.contract.entity.tx.ContractTransferTransaction;
import io.nuls.contract.entity.txdata.CallContractData;
import io.nuls.contract.helper.VMHelper;
import io.nuls.contract.ledger.manager.ContractBalanceManager;
import io.nuls.contract.ledger.service.ContractUtxoService;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.contract.storage.service.ContractTokenTransferStorageService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.contract.vm.program.ProgramExecutor;
import io.nuls.contract.vm.program.ProgramStatus;
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.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.Transaction;
import io.nuls.kernel.processor.TransactionProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.VarInt;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;

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

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/8
 */
@Component
public class CallContractTxProcessor implements
TransactionProcessor<CallContractTransaction> {

    @Autowired
    private VMHelper vmHelper;

    @Autowired
    private ContractAddressStorageService contractAddressStorageService;

    @Autowired
    private ContractTokenTransferStorageService contractTokenTransferStorageService;

    @Autowired
    private ContractService contractService;

    @Autowired
    private ContractUtxoService contractUtxoService;

    @Autowired
```

```
    private LedgerService ledgerService;

    @Autowired
    private AccountLedgerService accountLedgerService;

    @Autowired
    private ContractBalanceManager contractBalanceManager;

    @Override
    public Result onRollback(CallContractTransaction tx, Object secondaryData) {
        try {
            //
            byte[] txHashBytes = null;
            try {
                txHashBytes = tx.getHash().serialize();
            } catch (IOException e) {
                Log.error(e);
            }

            CallContractData txData = tx.getTxData();
            byte[] senderContractAddressBytes = txData.getContractAddress();
            Result<ContractAddressInfoPo> senderContractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(senderContractAddressBytes);
            ContractAddressInfoPo po = senderContractAddressInfoResult.getData();
            if(po != null) {
                ContractResult contractResult = tx.getContractResult();
                if(contractResult == null) {
                    contractResult = contractService.getContractExecuteResult(tx.getHash());
                }
                if(contractResult != null) {
                    //  - transferEvent,
                    if(!contractResult.isSuccess()) {
                        if(ContractUtil.isTransferMethod(txData.getMethodName())) {
contractTokenTransferStorageService.deleteTokenTransferInfo(ArraysTool.concatenate(txData.ge
tSender(), txHashBytes, new VarInt(0).encode()));
                        }
                    }
                    List<String> events = contractResult.getEvents();
                    int size = events.size();
                    // Transfer
                    String event;
                    ContractAddressInfoPo contractAddressInfo;
```

```
            if(events != null && size > 0) {
                for(int i = 0; i < size; i++) {
                    event = events.get(i);
                    // NRC20TransferEvent-from-to,
                    ContractTokenTransferInfoPo tokenTransferInfoPo =
ContractUtil.convertJsonToTokenTransferInfoPo(event);
                    if(tokenTransferInfoPo == null) {
                        continue;
                    }
                    String contractAddress = tokenTransferInfoPo.getContractAddress();
                    if (StringUtils.isBlank(contractAddress)) {
                        continue;
                    }
                    if (!AddressTool.validAddress(contractAddress)) {
                        continue;
                    }
                    byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
                    if(ArraysTool.arrayEquals(senderContractAddressBytes, contractAddressBytes))
{
                        contractAddressInfo = po;
                    } else {
                        Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
                        contractAddressInfo = contractAddressInfoResult.getData();
                    }

                    if(contractAddressInfo == null) {
                        continue;
                    }
                    // NRC20
                    if(!contractAddressInfo.isNrc20()) {
                        continue;
                    }

                    // token
                    this.rollbackContractToken(tokenTransferInfoPo);
contractTokenTransferStorageService.deleteTokenTransferInfo(ArraysTool.concatenate(tokenTra
nsferInfoPo.getFrom(), txHashBytes, new VarInt(i).encode()));
contractTokenTransferStorageService.deleteTokenTransferInfo(ArraysTool.concatenate(tokenTra
nsferInfoPo.getTo(), txHashBytes, new VarInt(i).encode()));
                }
            }
```

```
            }
        }

        //
        //
        //
        Collection<ContractTransferTransaction> contractTransferTxs =
tx.getContractTransferTxs();
        if(contractTransferTxs != null && contractTransferTxs.size() > 0) {
            List<ContractTransferTransaction> contractTransferTxList = new
ArrayList<>(contractTransferTxs);
            Collections.reverse(contractTransferTxList);
            //
            List<Transaction> txList = new ArrayList<>();
            for(ContractTransferTransaction transferTx : contractTransferTxList) {
                try {
                    txList.add(transferTx);
                    Result result = ledgerService.rollbackTx(transferTx);
                    if(result.isFailed()) {
                        Log.error("rollback contract transfer tx from ledger error. msg: {}",
result.getMsg());
                        return result;
                    }
                    result = contractService.rollbackContractTransferTx(transferTx);
                    if(result.isFailed()) {
                        Log.error("rollback contract transfer tx from contract ledger error. msg: {}",
result.getMsg());
                        return Result.getFailed();
                    }
                } catch (Exception e) {
                    Log.error("rollback contract transfer tx error. msg: {}", e.getMessage());
                    return Result.getFailed();
                }
            }
            Result result = accountLedgerService.rollbackTransactions(txList);
            if(result.isFailed()) {
                Log.error("rollback contract transfer tx from account ledger error. msg: {}",
result.getMsg());
                return Result.getFailed();
            }
        }
```

```
            // UTXO
            contractUtxoService.deleteUtxoOfTransaction(tx);

            //
            contractService.deleteContractExecuteResult(tx.getHash());
        } catch (Exception e) {
            Log.error("rollback call contract tx error.", e);
            return Result.getFailed();
        }
        return Result.getSuccess();
    }

    @Override
    public Result onCommit(CallContractTransaction tx, Object secondaryData) {
        try {
            ContractResult contractResult = tx.getContractResult();

            // UTXO
            Result utxoResult = contractUtxoService.saveUtxoForContractAddress(tx);
            if (utxoResult.isFailed()) {
                Log.error("save confirmed contract utxo error, reason is {}.", utxoResult.getMsg());
                return utxoResult;
            }

            long blockHeight = tx.getBlockHeight();
            /**
             *
             */
            Collection<ContractTransferTransaction> contractTransferTxs =
tx.getContractTransferTxs();
            if(contractTransferTxs != null && contractTransferTxs.size() > 0) {

                for(ContractTransferTransaction transferTx : contractTransferTxs) {
                    try {
                        transferTx.setBlockHeight(blockHeight);

                        Result result = ledgerService.saveTx(transferTx);
                        if(result.isFailed()) {
                            Log.error("save contract transfer tx to ledger error. msg: {}", result.getMsg());
                            return result;
                        }
```

```
                result = contractService.saveContractTransferTx(transferTx);
                if(result.isFailed()) {
                    Log.error("save contract transfer tx to contract ledger error. msg: {}",
result.getMsg());
                    return result;
                }

                result = accountLedgerService.saveConfirmedTransaction(transferTx);
                if(result.isFailed()) {
                    Log.error("save contract transfer tx to account ledger error. msg: {}",
result.getMsg());
                    return result;
                }

            } catch (Exception e) {
                e.printStackTrace();
                Log.error("save contract transfer tx error. msg: {}", e.getMessage());
                return Result.getFailed();
            }
        }


    }

    //
    CallContractData callContractData = tx.getTxData();
    byte[] contractAddress = callContractData.getContractAddress();

    Result<ContractAddressInfoPo> contractAddressInfoPoResult =
contractAddressStorageService.getContractAddressInfo(contractAddress);
    if(contractAddressInfoPoResult.isFailed()) {
        return contractAddressInfoPoResult;
    }
    ContractAddressInfoPo contractAddressInfoPo = contractAddressInfoPoResult.getData();
    if(contractAddressInfoPo == null) {
        return Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST);
    }
    contractResult.setNrc20(contractAddressInfoPo.isNrc20());

    BlockHeader blockHeader = tx.getBlockHeader();
    byte[] newestStateRoot = blockHeader.getStateRoot();
```

```
        //
        ProgramStatus status = vmHelper.getContractStatus(newestStateRoot, contractAddress);
        boolean isTerminatedContract = ContractUtil.isTerminatedContract(status.ordinal());

        //  - transferEvent, ,
        if(isTerminatedContract || !contractResult.isSuccess()) {
            if(contractAddressInfoPo != null && contractAddressInfoPo.isNrc20() &&
ContractUtil.isTransferMethod(callContractData.getMethodName())) {
                byte[] txHashBytes = tx.getHash().serialize();
                byte[] infoKey = ArraysTool.concatenate(callContractData.getSender(), txHashBytes,
new VarInt(0).encode());
                Result<ContractTokenTransferInfoPo> infoResult =
contractTokenTransferStorageService.getTokenTransferInfo(infoKey);
                ContractTokenTransferInfoPo po = infoResult.getData();
                if(po != null) {
                    po.setStatus((byte) 2);
                    contractTokenTransferStorageService.saveTokenTransferInfo(infoKey, po);

                    // token
                    if(isTerminatedContract) {
                        // token
                        this.rollbackContractToken(po);
                        contractResult.setError(true);
                        contractResult.setErrorMessage("this contract has been terminated");
                    } else {

                        if(po.getFrom() != null) {
                            vmHelper.refreshTokenBalance(newestStateRoot, contractAddressInfoPo,
AddressTool.getStringAddressByBytes(po.getFrom()), po.getContractAddress());
                        }
                        if(po.getTo() != null) {
                            vmHelper.refreshTokenBalance(newestStateRoot, contractAddressInfoPo,
AddressTool.getStringAddressByBytes(po.getTo()), po.getContractAddress());
                        }
                    }
                }
            }
        }

        if(!isTerminatedContract) {
            //
```

```
            vmHelper.dealEvents(newestStateRoot, tx, contractResult, contractAddressInfoPo);
        }

        //
        contractService.saveContractExecuteResult(tx.getHash(), contractResult);

    } catch (Exception e) {
        Log.error("save call contract tx error.", e);
        return Result.getFailed();
    }
    return Result.getSuccess();
}

private void rollbackContractToken(ContractTokenTransferInfoPo po) {
    try {
        String contractAddressStr = po.getContractAddress();
        byte[] from = po.getFrom();
        byte[] to = po.getTo();
        BigInteger token = po.getValue();
        String fromStr = null;
        String toStr = null;
        if(from != null) {
            fromStr = AddressTool.getStringAddressByBytes(from);
        }
        if(to != null) {
            toStr = AddressTool.getStringAddressByBytes(to);
        }
        contractBalanceManager.addContractToken(fromStr, contractAddressStr, token);
        contractBalanceManager.subtractContractToken(toStr, contractAddressStr, token);
    } catch (Exception e) {
        // skip it
    }
}

@Override
public ValidateResult conflictDetect(List<Transaction> txList) {
    return ValidateResult.getSuccessResult();
}
}
```

79:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-tx\src\main\java\io\nuls\contract\entity\tx\processor\ContractTransferTxProcessor.java

```
package io.nuls.contract.entity.tx.processor;

import io.nuls.contract.entity.tx.ContractTransferTransaction;
import io.nuls.contract.entity.tx.CreateContractTransaction;
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.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/7
 */
@Component
public class ContractTransferTxProcessor implements
TransactionProcessor<ContractTransferTransaction> {
    @Override
    public Result onRollback(ContractTransferTransaction tx, Object secondaryData) {
        return Result.getSuccess();
    }

    @Override
    public Result onCommit(ContractTransferTransaction tx, Object secondaryData) {
        return Result.getSuccess();
    }

    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        return ValidateResult.getSuccessResult();
    }
}

80:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
tx\src\main\java\io\nuls\contract\entity\tx\processor\CreateContractTxProcessor.java
package io.nuls.contract.entity.tx.processor;

import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.entity.tx.CreateContractTransaction;
```

```
import io.nuls.contract.entity.txdata.CreateContractData;
import io.nuls.contract.helper.VMHelper;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.service.ContractTxService;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.contract.storage.service.ContractCollectionStorageService;
import io.nuls.contract.storage.service.ContractExecuteResultStorageService;
import io.nuls.contract.vm.program.ProgramExecutor;
import io.nuls.contract.vm.program.ProgramResult;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;
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.Component;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.processor.TransactionProcessor;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;

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

import static io.nuls.contract.constant.ContractConstant.*;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/7
 */
@Component
public class CreateContractTxProcessor implements
TransactionProcessor<CreateContractTransaction>, InitializingBean {

    @Autowired
```

```
private ContractAddressStorageService contractAddressStorageService;

@Autowired
private ContractTxService contractTxService;

@Autowired
private ContractService contractService;

@Autowired
private ContractExecuteResultStorageService contractExecuteResultStorageService;

@Autowired
private ContractCollectionStorageService contractCollectionStorageService;

@Autowired
private VMHelper vmHelper;

private ProgramExecutor programExecutor;

@Override
public void afterPropertiesSet() throws NulsException {
    programExecutor = vmHelper.getProgramExecutor();
}

@Override
public Result onRollback(CreateContractTransaction tx, Object secondaryData) {
    CreateContractData txData = tx.getTxData();
    byte[] contractAddress = txData.getContractAddress();
    contractCollectionStorageService.deleteContractAddress(contractAddress);
    contractAddressStorageService.deleteContractAddress(contractAddress);
    contractService.deleteContractExecuteResult(tx.getHash());
    return Result.getSuccess();
}

@Override
public Result onCommit(CreateContractTransaction tx, Object secondaryData) {
    ContractResult contractResult = tx.getContractResult();
    contractService.saveContractExecuteResult(tx.getHash(), contractResult);

    CreateContractData txData = tx.getTxData();
    byte[] contractAddress = txData.getContractAddress();
    byte[] sender = txData.getSender();
```

```
        String senderStr = AddressTool.getStringAddressByBytes(sender);
        String contractAddressStr = AddressTool.getStringAddressByBytes(contractAddress);
        //
        contractTxService.removeLocalUnconfirmedCreateContractTransaction(
                senderStr, contractAddressStr, contractResult);

        //
        if(!contractResult.isSuccess()) {
            return Result.getSuccess();
        }

        NulsDigestData hash = tx.getHash();
        long blockHeight = tx.getBlockHeight();
        long bestBlockHeight = NulsContext.getInstance().getBestHeight();
        ContractAddressInfoPo info = new ContractAddressInfoPo();
        info.setContractAddress(contractAddress);
        info.setSender(sender);
        try {
            info.setCreateTxHash(hash.serialize());
        } catch (IOException e) {
            throw new NulsRuntimeException(e);
        }
        info.setCreateTime(tx.getTime());
        info.setBlockHeight(blockHeight);

        //byte[] stateRoot = contractResult.getStateRoot();
        boolean isNrc20Contract = contractResult.isNrc20();
        boolean acceptDirectTransfer = contractResult.isAcceptDirectTransfer();
        info.setAcceptDirectTransfer(acceptDirectTransfer);
        info.setNrc20(isNrc20Contract);
        //  token tracker
        if(isNrc20Contract) {
            BlockHeader blockHeader = tx.getBlockHeader();
            byte[] newestStateRoot = blockHeader.getStateRoot();
            // NRC20 token
            ProgramResult programResult = vmHelper.invokeViewMethod(newestStateRoot,
bestBlockHeight, contractAddress, NRC20_METHOD_NAME, null, null);
            if(programResult.isSuccess()) {
                String tokenName = programResult.getResult();
                info.setNrc20TokenName(tokenName);
            }
            programResult = vmHelper.invokeViewMethod(newestStateRoot, bestBlockHeight,
```

```
contractAddress, NRC20_METHOD_SYMBOL, null, null);
            if(programResult.isSuccess()) {
                String symbol = programResult.getResult();
                info.setNrc20TokenSymbol(symbol);
            }
            programResult = vmHelper.invokeViewMethod(newestStateRoot, bestBlockHeight,
contractAddress, NRC20_METHOD_DECIMALS, null, null);
            if(programResult.isSuccess()) {
                String decimals = programResult.getResult();
                if(StringUtils.isNotBlank(decimals)) {
                    try {
                        info.setDecimals(new BigInteger(decimals).longValue());
                    } catch (Exception e) {
                        Log.error("Get nrc20 decimals error.", e);
                        // skip it
                    }
                }
            }
            programResult = vmHelper.invokeViewMethod(newestStateRoot, bestBlockHeight,
contractAddress, NRC20_METHOD_TOTAL_SUPPLY, null, null);
            if(programResult.isSuccess()) {
                String totalSupply = programResult.getResult();
                if(StringUtils.isNotBlank(totalSupply)) {
                    try {
                        info.setTotalSupply(new BigInteger(totalSupply));
                    } catch (Exception e) {
                        Log.error("Get nrc20 totalSupply error.", e);
                        // skip it
                    }
                }
            }
            // token
            vmHelper.refreshTokenBalance(newestStateRoot, info, senderStr, contractAddressStr);
            //
            vmHelper.dealEvents(newestStateRoot, tx, contractResult, info);
        }

        Result result = contractAddressStorageService.saveContractAddress(contractAddress, info);
        return result;
    }

    @Override
```

```
    public ValidateResult conflictDetect(List<Transaction> txList) {
        return ValidateResult.getSuccessResult();
    }

}
```

81:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-tx\src\main\java\io\nuls\contract\entity\tx\processor\DeleteContractTxProcessor.java

```
package io.nuls.contract.entity.tx.processor;

import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.entity.tx.DeleteContractTransaction;
import io.nuls.contract.service.ContractService;
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.List;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/6/8
 */
@Component
public class DeleteContractTxProcessor implements
TransactionProcessor<DeleteContractTransaction> {

    @Autowired
    private ContractService contractService;

    @Override
    public Result onRollback(DeleteContractTransaction tx, Object secondaryData) {
        contractService.deleteContractExecuteResult(tx.getHash());
        return Result.getSuccess();
    }

    @Override
    public Result onCommit(DeleteContractTransaction tx, Object secondaryData) {
```

```
        ContractResult contractResult = tx.getContractResult();
        contractService.saveContractExecuteResult(tx.getHash(), contractResult);
        return Result.getSuccess();
    }

    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        return ValidateResult.getSuccessResult();
    }
}
```

82:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-tx\src\main\java\io\nuls\contract\entity\tx\validator\CallContractTxValidator.java
*/

```
package io.nuls.contract.entity.tx.validator;

import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.entity.tx.CallContractTransaction;
import io.nuls.contract.entity.txdata.CallContractData;
import io.nuls.contract.ledger.util.ContractLedgerUtil;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolConstant;

import java.util.Set;

/**
 * @author: PierreLuo
 * @date: 2018/10/2
 */
@Component
public class CallContractTxValidator implements NulsDataValidator<CallContractTransaction> {
```

```
    @Override
    public ValidateResult validate(CallContractTransaction tx) throws NulsException {
        CallContractData txData = tx.getTxData();
        Na transferNa = Na.valueOf(txData.getValue());
        byte[] contractAddress = txData.getContractAddress();
        byte[] sender = txData.getSender();
        Set<String> addressSet = SignatureUtil.getAddressFromTX(tx);

        if(!ContractLedgerUtil.isExistContractAddress(contractAddress)) {
            Log.error("contract data error: The contract does not exist.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST);
        }

        if (!addressSet.contains(AddressTool.getStringAddressByBytes(sender))) {
            Log.error("contract data error: The contract caller is not the transaction creator.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }

        Na contractReceivedNa = Na.ZERO;
        for (Coin coin : tx.getCoinData().getTo()) {
            byte[] owner = coin.getOwner();
            if (owner.length > 23) {
                owner = coin.getAddress();
            }
            // Keep the change maybe a very small coin
            if (addressSet.contains(AddressTool.getStringAddressByBytes(owner))) {
                // When the receiver sign this tx,Allow it transfer small coin
                continue;
            }

            if (coin.getLockTime() != 0) {
                Log.error("contract data error: The amount of the transfer cannot be locked(UTXO
status error).");
                return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.UTXO_STATUS_CHANGE);
            }

            if (!ArraysTool.arrayEquals(owner, contractAddress)) {
                Log.error("contract data error: The receiver is not the contract address.");
```

```
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        } else {
            contractReceivedNa = contractReceivedNa.add(coin.getNa());
        }

        if (coin.getNa().isLessThan(ProtocolConstant.MININUM_TRANSFER_AMOUNT)) {
            Log.error("contract data error: The amount of the transfer is too small.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TOO_SMALL_AMOUNT);
        }
    }
    if (contractReceivedNa.isLessThan(transferNa)) {
        Log.error("contract data error: Insufficient amount to transfer to the contract address.");
        return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.INVALID_AMOUNT);
    }
    return ValidateResult.getSuccessResult();
  }
}
```

83:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-tx\src\main\java\io\nuls\contract\entity\tx\validator\ContractAcceptTransferredTxValidator.java

```
  public ValidateResult validate(Transaction tx) throws NulsException {
    if(tx.getCoinData() == null){
        return ValidateResult.getSuccessResult();
    }
    List<Coin> toList = tx.getCoinData().getTo();
    if(toList == null || toList.size() == 0){
        return ValidateResult.getSuccessResult();
    }
    int type = tx.getType();
    for (Coin coin : toList) {
        if(ContractUtil.isLegalContractAddress(coin.getOwner())) {
            if(type != NulsConstant.TX_TYPE_COINBASE && type !=
ContractConstant.TX_TYPE_CALL_CONTRACT) {
                Log.error("contract data error: The contract does not accept transfers of this type[{}] of
transaction.", type);
                return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
            }
        }
```

```
        }
        return ValidateResult.getSuccessResult();
    }
}
```

84:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-tx\src\main\java\io\nuls\contract\entity\tx\validator\CreateContractTxValidator.java
*/

```
package io.nuls.contract.entity.tx.validator;

import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.entity.tx.CreateContractTransaction;
import io.nuls.contract.entity.txdata.CreateContractData;
import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

import java.util.Set;

/**
 * @author: PierreLuo
 * @date: 2018/10/2
 */
@Component
public class CreateContractTxValidator implements
NulsDataValidator<CreateContractTransaction> {

    @Override
    public ValidateResult validate(CreateContractTransaction tx) throws NulsException {
        CreateContractData txData = tx.getTxData();
        byte[] sender = txData.getSender();
        byte[] contractAddress = txData.getContractAddress();
        if(!ContractUtil.isLegalContractAddress(contractAddress)) {
            Log.error("contract data error: Illegal contract address.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
```

```
ContractErrorCode.ILLEGAL_CONTRACT_ADDRESS);
        }
        Set<String> addressSet = SignatureUtil.getAddressFromTX(tx);

        if (!addressSet.contains(AddressTool.getStringAddressByBytes(sender))) {
            Log.error("contract data error: The contract creater is not the transaction creator.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }

        return ValidateResult.getSuccessResult();
    }
}

85:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
tx\src\main\java\io\nuls\contract\entity\tx\validator\DeleteContractTxValidator.java
 */

package io.nuls.contract.entity.tx.validator;

import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.entity.tx.DeleteContractTransaction;
import io.nuls.contract.entity.txdata.DeleteContractData;
import io.nuls.contract.ledger.module.ContractBalance;
import io.nuls.contract.ledger.service.ContractUtxoService;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.TransactionErrorCode;
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.Na;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

import java.util.Set;
```

```
/**
 * @author: PierreLuo
 * @date: 2018/10/2
 */
@Component
public class DeleteContractTxValidator implements
NulsDataValidator<DeleteContractTransaction> {

    @Autowired
    private ContractAddressStorageService contractAddressStorageService;

    @Autowired
    private ContractUtxoService contractUtxoService;

    @Override
    public ValidateResult validate(DeleteContractTransaction tx) throws NulsException {

        DeleteContractData txData = tx.getTxData();
        byte[] sender = txData.getSender();
        byte[] contractAddressBytes = txData.getContractAddress();
        Set<String> addressSet = SignatureUtil.getAddressFromTX(tx);

        if (!addressSet.contains(AddressTool.getStringAddressByBytes(sender))) {
            Log.error("contract data error: The contract deleter is not the transaction creator.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }

        Result<ContractAddressInfoPo> contractAddressInfoPoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
        if(contractAddressInfoPoResult.isFailed()) {
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
contractAddressInfoPoResult.getErrorCode());
        }
        ContractAddressInfoPo contractAddressInfoPo = contractAddressInfoPoResult.getData();
        if(contractAddressInfoPo == null) {
            Log.error("contract data error: The contract does not exist.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST);
        }
        if(!ArraysTool.arrayEquals(sender, contractAddressInfoPo.getSender())) {
            Log.error("contract data error: The contract deleter is not the contract creator.");
```

```
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }

        Result<ContractBalance> result = contractUtxoService.getBalance(contractAddressBytes);
        ContractBalance balance = (ContractBalance) result.getData();
        if(balance == null) {
            Log.error("contract data error: That balance of the contract is abnormal.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }

        Na totalBalance = balance.getBalance();
        if(totalBalance.compareTo(Na.ZERO) != 0) {
            Log.error("contract data error: The balance of the contract is not 0.");
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
ContractErrorCode.CONTRACT_DELETE_BALANCE);
        }


        return ValidateResult.getSuccessResult();
    }
}
```

86:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-tx\src\main\java\io\nuls\contract\service\ContractTxService.java

```
package io.nuls.contract.service;

import io.nuls.contract.dto.ContractResult;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.Result;

import java.util.LinkedList;
import java.util.Map;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/5/22
 */
public interface ContractTxService {
    /**
```

```
     *
     *
     * @param sender
     * @param gasLimit        gas
     * @param price
     * @param contractCode
     * @param args
     * @param password
     * @param remark
     * @return
     */
    Result contractCreateTx(String sender, Long gasLimit, Long price,
                            byte[] contractCode, String[][] args, String password, String remark);

    LinkedList<Map<String, String>> getLocalUnconfirmedCreateContractTransaction(String
sender);

    void removeLocalUnconfirmedCreateContractTransaction(String sender, String
contractAddress, ContractResult contractResult);

    void removeLocalUnconfirmedCreateContractTransaction(String sender, String
contractAddress);

    void removeLocalFailedUnconfirmedCreateContractTransaction(String sender, String
contractAddress);

    /**
     *
     *
     *
     * @param sender
     * @param gasLimit        gas
     * @param price
     * @param contractCode
     * @param args
     * @param password
     * @param remark
     * @return
     */
    Result contractPreCreateTx(String sender, Long gasLimit, Long price,
                               byte[] contractCode, String[][] args, String password, String remark);
```

```
    /**
     *
     *
     * @param sender
     * @param value
     * @param gasLimit        gas
     * @param price
     * @param contractAddress
     * @param methodName
     * @param methodDesc
     * @param args
     * @param password
     * @param remark
     * @return
     */
    Result contractCallTx(String sender, Na value, Long gasLimit, Long price, String
contractAddress,
                          String methodName, String methodDesc, String[][] args, String password, String
remark);

    /**
     *
     *
     * @param sender
     * @param value
     * @param gasLimit        gas
     * @param price
     * @param contractAddress
     * @param methodName
     * @param methodDesc
     * @param args
     * @param remark
     * @return
     */
    Result transferFee(String sender, Na value, Long gasLimit, Long price, String contractAddress,
                           String methodName, String methodDesc, String[][] args, String remark);

    /**
     *
     *
     * @param sender
     * @param contractAddress
```

```
     * @param password
     * @param remark
     * @return
     */
    Result contractDeleteTx(String sender, String contractAddress, String password, String
remark);
}

87:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
tx\src\main\java\io\nuls\contract\service\impl\ContractTxServiceImpl.java
package io.nuls.contract.service.impl;

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.service.AccountService;
import io.nuls.account.util.AccountTool;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.constant.ContractErrorCode;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.entity.tx.CallContractTransaction;
import io.nuls.contract.entity.tx.CreateContractTransaction;
import io.nuls.contract.entity.tx.DeleteContractTransaction;
import io.nuls.contract.entity.txdata.CallContractData;
import io.nuls.contract.entity.txdata.CreateContractData;
import io.nuls.contract.entity.txdata.DeleteContractData;
import io.nuls.contract.helper.VMHelper;
import io.nuls.contract.ledger.manager.ContractBalanceManager;
import io.nuls.contract.ledger.module.ContractBalance;
import io.nuls.contract.service.ContractTxService;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.contract.storage.service.ContractTokenTransferStorageService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.contract.util.VMContext;
import io.nuls.contract.vm.program.*;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.calc.LongUtils;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.log.Log;
```

```
import io.nuls.core.tools.map.MapUtil;
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.func.TimeService;
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.*;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.TransactionFeeCalculator;
import io.nuls.kernel.utils.VarInt;
import io.nuls.protocol.service.TransactionService;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.math.BigInteger;
import java.util.*;
import java.util.concurrent.locks.ReentrantLock;

import static io.nuls.contract.constant.ContractConstant.MAX_GASLIMIT;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/5/22
 */
@Component
public class ContractTxServiceImpl implements ContractTxService, InitializingBean {

    @Autowired
    private AccountService accountService;
    @Autowired
    private AccountLedgerService accountLedgerService;
    @Autowired
    private TransactionService transactionService;
    @Autowired
```

```
private ContractAddressStorageService contractAddressStorageService;
@Autowired
private ContractTokenTransferStorageService contractTokenTransferStorageService;
@Autowired
private VMHelper vmHelper;
@Autowired
private VMContext vmContext;
@Autowired
private ContractBalanceManager contractBalanceManager;

private ProgramExecutor programExecutor;

@Override
public void afterPropertiesSet() throws NulsException {
    programExecutor = vmHelper.getProgramExecutor();
}

/**
 *
 *
 *
 * @param sender
 * @param gasLimit     gas
 * @param price
 * @param contractCode
 * @param args
 * @param password
 * @param remark
 * @return
 */
@Override
public Result contractCreateTx(String sender, Long gasLimit, Long price,
                               byte[] contractCode, String[][] args,
                               String password, String remark) {
    try {
        AssertUtil.canNotEmpty(sender, "the sender address can not be empty");
        AssertUtil.canNotEmpty(contractCode, "the contractCode can not be empty");
        Na value = Na.ZERO;

        Result<Account> accountResult = accountService.getAccount(sender);
        if (accountResult.isFailed()) {
            return accountResult;
```

```
}

if(!ContractUtil.checkPrice(price.longValue())) {
    return Result.getFailed(ContractErrorCode.CONTRACT_MINIMUM_PRICE);
}

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);
    }
}

//
Address contractAddress = AccountTool.createContractAddress();

byte[] contractAddressBytes = contractAddress.getAddressBytes();
byte[] senderBytes = AddressTool.getAddress(sender);

CreateContractTransaction tx = new CreateContractTransaction();
if (StringUtils.isNotBlank(remark)) {
    try {
        tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
    } catch (UnsupportedEncodingException e) {
        Log.error(e);
        throw new RuntimeException(e);
    }
}
tx.setTime(TimeService.currentTimeMillis());


// CoinData
/*
 * Gas*Price
 * CoinData
 * tx
 * CoinBaseSender
 */
CoinData coinData = new CoinData();
//
```

```
if (!Na.ZERO.equals(value)) {
    Coin toCoin = new Coin(contractAddressBytes, value);
    coinData.getTo().add(toCoin);
}

BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
//
long blockHeight = blockHeader.getHeight();
//
byte[] prevStateRoot = ContractUtil.getStateRoot(blockHeader);
AssertUtil.canNotEmpty(prevStateRoot, "All features of the smart contract are locked.");

// VM
ProgramCreate programCreate = new ProgramCreate();
programCreate.setContractAddress(contractAddressBytes);
programCreate.setSender(senderBytes);
programCreate.setValue(BigInteger.valueOf(value.getValue()));
programCreate.setPrice(price.longValue());
programCreate.setGasLimit(gasLimit.longValue());
programCreate.setNumber(blockHeight);
programCreate.setContractCode(contractCode);
if (args != null) {
    programCreate.setArgs(args);
}
ProgramExecutor track = programExecutor.begin(prevStateRoot);
// Gas
long realGasLimit = programCreate.getGasLimit();
programCreate.setGasLimit(MAX_GASLIMIT);
ProgramResult programResult = track.create(programCreate);

// Gas
if(!programResult.isSuccess()) {
    Result result = Result.getFailed(ContractErrorCode.DATA_ERROR);
    result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
    return result;
} else {
    // Gas
    track = programExecutor.begin(prevStateRoot);
    programCreate.setGasLimit(realGasLimit);
    programResult = track.create(programCreate);
    if(!programResult.isSuccess()) {
        Result result = Result.getFailed(ContractErrorCode.DATA_ERROR);
```

```
                result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
                return result;
            }
        }
        long gasUsed = gasLimit.longValue();
        Na imputedNa = Na.valueOf(LongUtils.mul(gasUsed, price));
        //
        Na totalNa = imputedNa.add(value);

        // txData
        CreateContractData createContractData = new CreateContractData();
        createContractData.setSender(senderBytes);
        createContractData.setContractAddress(contractAddressBytes);
        createContractData.setValue(value.getValue());
        createContractData.setGasLimit(gasLimit);
        createContractData.setPrice(price);
        createContractData.setCodeLen(contractCode.length);
        createContractData.setCode(contractCode);
        if (args != null) {
            createContractData.setArgsCount((byte) args.length);
            if (args.length > 0) {
                createContractData.setArgs(args);
            }
        }
        tx.setTxData(createContractData);

        CoinDataResult coinDataResult = accountLedgerService.getCoinData(senderBytes,
totalNa, tx.size() + coinData.size(), TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
        if (!coinDataResult.isEnough()) {
            return Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE);
        }
        coinData.setFrom(coinDataResult.getCoinList());
        // UTXO
        if (coinDataResult.getChange() != null) {
            coinData.getTo().add(coinDataResult.getChange());
        }
        tx.setCoinData(coinData);
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));

        //
        List<ECKey> signEckeys = new ArrayList<>();
        List<ECKey> scriptEckeys = new ArrayList<>();
```

```
        ECKey eckey = account.getEcKey(password);
        //1
        if ((coinDataResult.getSignType() & 0x01) == 0x01) {
            signEckeys.add(eckey);
        }
        //1
        if ((coinDataResult.getSignType() & 0x02) == 0x02) {
            scriptEckeys.add(eckey);
        }
        SignatureUtil.createTransactionSignture(tx, scriptEckeys, signEckeys);

        //
        Result saveResult = accountLedgerService.verifyAndSaveUnconfirmedTransaction(tx);
        if (saveResult.isFailed()) {
            if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                //()
                Result rs = accountLedgerService.getMaxAmountOfOnce(senderBytes, 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()) {
            accountLedgerService.deleteTransaction(tx);
            return sendResult;
        }
        Map<String, String> resultMap = MapUtil.createHashMap(2);
        String txHash = tx.getHash().getDigestHex();
        String contractAddressStr =
AddressTool.getStringAddressByBytes(contractAddressBytes);
        resultMap.put("txHash", txHash);
        resultMap.put("contractAddress", contractAddressStr);
```

```
            //
            this.saveLocalUnconfirmedCreateContractTransaction(sender, resultMap, tx.getTime());
            return Result.getSuccess().setData(resultMap);
        } catch (IOException e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        } catch (NulsException e) {
            Log.error(e);
            return Result.getFailed(e.getErrorCode());
        } catch (Exception e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        }
    }

    /**
     * key: accountAddress
     * value(Map):
     * key: contractAddress
     * value(Map):
     * key: txHash / contractAddress / time/ success(optional)
     * value: txHash-V / contractAddress-V / time-V/ success-V(true,false)
     */
    private static final Map<String, Map<String, Map<String, String>>>
LOCAL_UNCONFIRMED_CREATE_CONTRACT_TRANSACTION =
MapUtil.createLinkedHashMap(4);
    private ReentrantLock lock = new ReentrantLock();

    private void saveLocalUnconfirmedCreateContractTransaction(String sender, Map<String,
String> resultMap, long time) {
        lock.lock();
        try {
            LinkedHashMap<String, String> map = MapUtil.createLinkedHashMap(3);
            map.putAll(resultMap);
            map.put("time", String.valueOf(time));
            String contractAddress = map.get("contractAddress");
            Map<String, Map<String, String>> unconfirmedOfAccountMap =
LOCAL_UNCONFIRMED_CREATE_CONTRACT_TRANSACTION.get(sender);
```

```
            if (unconfirmedOfAccountMap == null) {
                unconfirmedOfAccountMap = MapUtil.createLinkedHashMap(4);
                unconfirmedOfAccountMap.put(contractAddress, map);
                LOCAL_UNCONFIRMED_CREATE_CONTRACT_TRANSACTION.put(sender,
unconfirmedOfAccountMap);
            } else {
                unconfirmedOfAccountMap.put(contractAddress, map);
            }
        } finally {
            lock.unlock();
        }
    }

    @Override
    public LinkedList<Map<String, String>> getLocalUnconfirmedCreateContractTransaction(String
sender) {
        Map<String, Map<String, String>> unconfirmedOfAccountMap =
LOCAL_UNCONFIRMED_CREATE_CONTRACT_TRANSACTION.get(sender);
        if (unconfirmedOfAccountMap == null) {
            return null;
        }
        return new LinkedList<>(unconfirmedOfAccountMap.values());
    }

    @Override
    public void removeLocalUnconfirmedCreateContractTransaction(String sender, String
contractAddress, ContractResult contractResult) {
        lock.lock();
        try {
            Map<String, Map<String, String>> unconfirmedOfAccountMap =
LOCAL_UNCONFIRMED_CREATE_CONTRACT_TRANSACTION.get(sender);
            if (unconfirmedOfAccountMap == null) {
                return;
            }
            //
            if (contractResult.isSuccess()) {
                unconfirmedOfAccountMap.remove(contractAddress);
            } else {
                //
                Map<String, String> dataMap = unconfirmedOfAccountMap.get(contractAddress);
                if (dataMap != null) {
                    dataMap.put("success", "false");
```

```
                dataMap.put("msg", contractResult.getErrorMessage());
            }
        }
    } finally {
        lock.unlock();
    }
}

@Override
public void removeLocalUnconfirmedCreateContractTransaction(String sender, String
contractAddress) {
    lock.lock();
    try {
        Map<String, Map<String, String>> unconfirmedOfAccountMap =
LOCAL_UNCONFIRMED_CREATE_CONTRACT_TRANSACTION.get(sender);
        if (unconfirmedOfAccountMap == null) {
            return;
        }
        unconfirmedOfAccountMap.remove(contractAddress);
    } finally {
        lock.unlock();
    }
}

@Override
public void removeLocalFailedUnconfirmedCreateContractTransaction(String sender, String
contractAddress) {
    lock.lock();
    try {
        Map<String, Map<String, String>> unconfirmedOfAccountMap =
LOCAL_UNCONFIRMED_CREATE_CONTRACT_TRANSACTION.get(sender);
        if (unconfirmedOfAccountMap == null) {
            return;
        }
        Map<String, String> dataMap = unconfirmedOfAccountMap.get(contractAddress);
        if (dataMap != null) {
            String success = dataMap.get("success");
            if ("false".equals(success)) {
                unconfirmedOfAccountMap.remove(contractAddress);
            }
        }
    } finally {
```

```
        lock.unlock();
    }
}

/**
 *
 *
 *
 * @param sender
 * @param gasLimit     gas
 * @param price
 * @param contractCode
 * @param args
 * @param password
 * @param remark
 * @return
 */
@Override
public Result contractPreCreateTx(String sender, Long gasLimit, Long price,
                                  byte[] contractCode, String[][] args,
                                  String password, String remark) {
    try {
        AssertUtil.canNotEmpty(sender, "the sender address can not be empty");
        AssertUtil.canNotEmpty(contractCode, "the contractCode can not be empty");
        Na value = Na.ZERO;

        Result<Account> accountResult = accountService.getAccount(sender);
        if (accountResult.isFailed()) {
            return accountResult;
        }

        //
        Address contractAddress = AccountTool.createContractAddress();

        byte[] contractAddressBytes = contractAddress.getAddressBytes();
        byte[] senderBytes = AddressTool.getAddress(sender);

        CreateContractTransaction tx = new CreateContractTransaction();
        if (StringUtils.isNotBlank(remark)) {
            try {
                tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
            } catch (UnsupportedEncodingException e) {
```

```
        Log.error(e);
        throw new RuntimeException(e);
    }
}
tx.setTime(TimeService.currentTimeMillis());


// CoinData
/*
 * Gas*Price
 * CoinData
 * tx
 * CoinBaseSender
 */
CoinData coinData = new CoinData();
//
if (!Na.ZERO.equals(value)) {
    Coin toCoin = new Coin(contractAddressBytes, value);
    coinData.getTo().add(toCoin);
}

BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
//
long blockHeight = blockHeader.getHeight();
//
byte[] prevStateRoot = ContractUtil.getStateRoot(blockHeader);
AssertUtil.canNotEmpty(prevStateRoot, "All features of the smart contract are locked.");

// VM
ProgramCreate programCreate = new ProgramCreate();
programCreate.setContractAddress(contractAddressBytes);
programCreate.setSender(senderBytes);
programCreate.setValue(BigInteger.valueOf(value.getValue()));
programCreate.setPrice(price.longValue());
programCreate.setGasLimit(gasLimit.longValue());
programCreate.setNumber(blockHeight);
programCreate.setContractCode(contractCode);
if (args != null) {
    programCreate.setArgs(args);
}
ProgramExecutor track = programExecutor.begin(prevStateRoot);
// Gas
```

```
long realGasLimit = programCreate.getGasLimit();
programCreate.setGasLimit(MAX_GASLIMIT);
ProgramResult programResult = track.create(programCreate);

// Gas
if(!programResult.isSuccess()) {
    Result result = Result.getFailed(ContractErrorCode.DATA_ERROR);
    result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
    return result;
} else {
    // Gas
    track = programExecutor.begin(prevStateRoot);
    programCreate.setGasLimit(realGasLimit);
    programResult = track.create(programCreate);
    if(!programResult.isSuccess()) {
        Result result = Result.getFailed(ContractErrorCode.DATA_ERROR);
        result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
        return result;
    }
}
long gasUsed = gasLimit.longValue();
Na imputedNa = Na.valueOf(LongUtils.mul(gasUsed, price));
//
Na totalNa = imputedNa.add(value);

// txData
CreateContractData createContractData = new CreateContractData();
createContractData.setSender(senderBytes);
createContractData.setContractAddress(contractAddressBytes);
createContractData.setValue(value.getValue());
createContractData.setGasLimit(gasLimit);
createContractData.setPrice(price);
createContractData.setCodeLen(contractCode.length);
createContractData.setCode(contractCode);
if (args != null) {
    createContractData.setArgsCount((byte) args.length);
    if (args.length > 0) {
        createContractData.setArgs(args);
    }
}
tx.setTxData(createContractData);
```

```
            CoinDataResult coinDataResult = accountLedgerService.getCoinData(senderBytes,
totalNa, tx.size(), TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
            if (!coinDataResult.isEnough()) {
                return Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE);
            }
            return Result.getSuccess();
        } catch (NulsException e) {
            Log.error(e);
            return Result.getFailed(e.getErrorCode());
        } catch (Exception e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        }
    }

    /**
     *
     *
     * @param sender
     * @param value
     * @param gasLimit        gas
     * @param price
     * @param contractAddress
     * @param methodName
     * @param methodDesc
     * @param args
     * @param password
     * @param remark
     * @return
     */
    @Override
    public Result contractCallTx(String sender, Na value, Long gasLimit, Long price, String
contractAddress,
                                 String methodName, String methodDesc, String[][] args,
                                 String password, String remark) {
        try {
            AssertUtil.canNotEmpty(sender, "the sender address can not be empty");
            AssertUtil.canNotEmpty(contractAddress, "the contractAddress can not be empty");
            AssertUtil.canNotEmpty(methodName, "the methodName can not be empty");
            if (value == null) {
```

```
    value = Na.ZERO;
}

if(!ContractUtil.checkPrice(price.longValue())) {
    return Result.getFailed(ContractErrorCode.CONTRACT_MINIMUM_PRICE);
}

Result<Account> accountResult = accountService.getAccount(sender);
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[] senderBytes = AddressTool.getAddress(sender);
byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);

BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
//
long blockHeight = blockHeader.getHeight();
//
byte[] prevStateRoot = ContractUtil.getStateRoot(blockHeader);
AssertUtil.canNotEmpty(prevStateRoot, "All features of the smart contract are locked.");

// VM
ProgramCall programCall = new ProgramCall();
programCall.setContractAddress(contractAddressBytes);
programCall.setSender(senderBytes);
programCall.setNumber(blockHeight);
programCall.setMethodName(methodName);
programCall.setMethodDesc(methodDesc);
programCall.setArgs(args);

//
if (vmHelper.checkIsViewMethod(methodName, methodDesc, contractAddressBytes)) {
```

```
    programCall.setValue(BigInteger.ZERO);
    programCall.setGasLimit(ContractConstant.CONTRACT_CONSTANT_GASLIMIT);
    programCall.setPrice(ContractConstant.CONTRACT_CONSTANT_PRICE);

    ProgramExecutor track = programExecutor.begin(prevStateRoot);
    ProgramResult programResult = track.call(programCall);
    Result result;
    if (!programResult.isSuccess()) {
        result = Result.getFailed(ContractErrorCode.DATA_ERROR);
        result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
    } else {
        result = Result.getSuccess();
        result.setData(programResult.getResult());
    }
    return result;
}


//
programCall.setValue(BigInteger.valueOf(value.getValue()));
programCall.setPrice(price.longValue());
programCall.setGasLimit(gasLimit.longValue());

CallContractTransaction tx = new CallContractTransaction();
if (StringUtils.isNotBlank(remark)) {
    try {
        tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
    } catch (UnsupportedEncodingException e) {
        Log.error(e);
        throw new RuntimeException(e);
    }
}
tx.setTime(TimeService.currentTimeMillis());

// CoinData
/*
 * Gas*Price
 * CoinData
 * tx
 * CoinBaseSender
 */
CoinData coinData = new CoinData();
```

```
//
if (!Na.ZERO.equals(value)) {
    Coin toCoin = new Coin(contractAddressBytes, value);
    coinData.getTo().add(toCoin);
}

// VM
ProgramExecutor track = programExecutor.begin(prevStateRoot);
// Gas
long realGasLimit = programCall.getGasLimit();
programCall.setGasLimit(MAX_GASLIMIT);
ProgramResult programResult = track.call(programCall);

// Gas
if(!programResult.isSuccess()) {
    Result result = Result.getFailed(ContractErrorCode.DATA_ERROR);
    result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
    return result;
} else {
    // Gas
    track = programExecutor.begin(prevStateRoot);
    programCall.setGasLimit(realGasLimit);
    programResult = track.call(programCall);
    if(!programResult.isSuccess()) {
        Result result = Result.getFailed(ContractErrorCode.DATA_ERROR);
        result.setMsg(ContractUtil.simplifyErrorMsg(programResult.getErrorMessage()));
        return result;
    }
}
long gasUsed = gasLimit.longValue();
Na imputedNa = Na.valueOf(LongUtils.mul(gasUsed, price));
//
Na totalNa = imputedNa.add(value);

// txData
CallContractData callContractData = new CallContractData();
callContractData.setContractAddress(contractAddressBytes);
callContractData.setSender(senderBytes);
callContractData.setValue(value.getValue());
callContractData.setPrice(price.longValue());
callContractData.setGasLimit(gasLimit.longValue());
callContractData.setMethodName(methodName);
```

```
        callContractData.setMethodDesc(methodDesc);
        if (args != null) {
            callContractData.setArgsCount((byte) args.length);
            callContractData.setArgs(args);
        }
        tx.setTxData(callContractData);

        CoinDataResult coinDataResult = accountLedgerService.getCoinData(senderBytes,
totalNa, tx.size() + coinData.size(), TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
        if (!coinDataResult.isEnough()) {
            return Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE);
        }
        coinData.setFrom(coinDataResult.getCoinList());
        // UTXO
        if (coinDataResult.getChange() != null) {
            coinData.getTo().add(coinDataResult.getChange());
        }
        tx.setCoinData(coinData);

        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));

        //
        List<ECKey> signEckeys = new ArrayList<>();
        List<ECKey> scriptEckeys = new ArrayList<>();
        ECKey eckey = account.getEcKey(password);
        //1
        if ((coinDataResult.getSignType() & 0x01) == 0x01) {
            signEckeys.add(eckey);
        }
        //1
        if ((coinDataResult.getSignType() & 0x02) == 0x02) {
            scriptEckeys.add(eckey);
        }
        SignatureUtil.createTransactionSignture(tx, scriptEckeys, signEckeys);

        // Token
        Result<byte[]> unConfirmedTokenTransferResult =
this.saveUnConfirmedTokenTransfer(tx, sender, contractAddress, methodName, args);
        if(unConfirmedTokenTransferResult.isFailed()) {
            return unConfirmedTokenTransferResult;
        }
        byte[] infoKey = unConfirmedTokenTransferResult.getData();
```

```
            //
            Result saveResult = accountLedgerService.verifyAndSaveUnconfirmedTransaction(tx);
            if (saveResult.isFailed()) {
                if (infoKey != null) {
                    contractTokenTransferStorageService.deleteTokenTransferInfo(infoKey);
                }
                if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                    //()
                    Result rs = accountLedgerService.getMaxAmountOfOnce(senderBytes, 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()) {
                //
                accountLedgerService.deleteTransaction(tx);
                if (infoKey != null) {
                    contractTokenTransferStorageService.deleteTokenTransferInfo(infoKey);
                }
                return sendResult;
            }

            return Result.getSuccess().setData(tx.getHash().getDigestHex());
        } catch (IOException e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        } catch (NulsException e) {
            Log.error(e);
```

```
            return Result.getFailed(e.getErrorCode());
        } catch (Exception e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        }
    }

    private Result<byte[]> saveUnConfirmedTokenTransfer(CallContractTransaction tx, String
sender, String contractAddress, String methodName, String[][] args) {
        try {
            byte[] senderBytes = AddressTool.getAddress(sender);
            byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
            Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
            ContractAddressInfoPo po = contractAddressInfoResult.getData();
            if(po != null && po.isNrc20() && ContractUtil.isTransferMethod(methodName)) {
                byte[] txHashBytes = tx.getHash().serialize();
                byte[] infoKey = ArraysTool.concatenate(senderBytes, txHashBytes, new
VarInt(0).encode());
                ContractTokenTransferInfoPo tokenTransferInfoPo = new
ContractTokenTransferInfoPo();
                if(ContractConstant.NRC20_METHOD_TRANSFER.equals(methodName)) {
                    String to = args[0][0];
                    String tokenValue = args[1][0];
                    BigInteger token = new BigInteger(tokenValue);
                    Result result = contractBalanceManager.subtractContractToken(sender,
contractAddress, token);
                    if(result.isFailed()) {
                        return result;
                    }
                    contractBalanceManager.addContractToken(to, contractAddress, token);
                    tokenTransferInfoPo.setFrom(senderBytes);
                    tokenTransferInfoPo.setTo(AddressTool.getAddress(to));
                    tokenTransferInfoPo.setValue(token);
                } else {
                    String from = args[0][0];
                    // token
                    if(!sender.equals(from)) {
                        return Result.getSuccess();
                    }
```

```
                String to = args[1][0];
                String tokenValue = args[2][0];
                BigInteger token = new BigInteger(tokenValue);
                Result result = contractBalanceManager.subtractContractToken(from,
contractAddress, token);
                if(result.isFailed()) {
                    return result;
                }
                contractBalanceManager.addContractToken(to, contractAddress, token);
                tokenTransferInfoPo.setFrom(AddressTool.getAddress(from));
                tokenTransferInfoPo.setTo(AddressTool.getAddress(to));
                tokenTransferInfoPo.setValue(token);
            }

            tokenTransferInfoPo.setName(po.getNrc20TokenName());
            tokenTransferInfoPo.setSymbol(po.getNrc20TokenSymbol());
            tokenTransferInfoPo.setDecimals(po.getDecimals());
            tokenTransferInfoPo.setTime(tx.getTime());
            tokenTransferInfoPo.setContractAddress(contractAddress);
            tokenTransferInfoPo.setBlockHeight(tx.getBlockHeight());
            tokenTransferInfoPo.setTxHash(txHashBytes);
            tokenTransferInfoPo.setStatus((byte) 0);
            Result result = contractTokenTransferStorageService.saveTokenTransferInfo(infoKey,
tokenTransferInfoPo);
            if(result.isFailed()) {
                return result;
            }
            return Result.getSuccess().setData(infoKey);
        }
        return Result.getSuccess();
    } catch (Exception e) {
        Log.error(e);
        Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
        result.setMsg(e.getMessage());
        return result;
    }
}

@Override
public Result transferFee(String sender, Na value, Long gasLimit, Long price, String
contractAddress,
                          String methodName, String methodDesc, String[][] args, String remark) {
```

```
try {
    AssertUtil.canNotEmpty(sender, "the sender address can not be empty");
    AssertUtil.canNotEmpty(contractAddress, "the contractAddress can not be empty");
    AssertUtil.canNotEmpty(methodName, "the methodName can not be empty");
    if (value == null) {
        value = Na.ZERO;
    }

    Result<Account> accountResult = accountService.getAccount(sender);
    if (accountResult.isFailed()) {
        return accountResult;
    }

    byte[] senderBytes = AddressTool.getAddress(sender);
    byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);

    BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
    //
    byte[] prevStateRoot = ContractUtil.getStateRoot(blockHeader);
    AssertUtil.canNotEmpty(prevStateRoot, "All features of the smart contract are locked.");

    CallContractTransaction tx = new CallContractTransaction();
    if (StringUtils.isNotBlank(remark)) {
        try {
            tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
        } catch (UnsupportedEncodingException e) {
            Log.error(e);
            throw new RuntimeException(e);
        }
    }
    tx.setTime(TimeService.currentTimeMillis());

    CoinData coinData = new CoinData();
    //
    if (!Na.ZERO.equals(value)) {
        Coin toCoin = new Coin(contractAddressBytes, value);
        coinData.getTo().add(toCoin);
    }

    long gasUsed = gasLimit.longValue();
    Na imputedGasUsedNa = Na.valueOf(LongUtils.mul(gasUsed, price));
    //
```

```
            Na totalNa = imputedGasUsedNa.add(value);

            // txData
            CallContractData callContractData = new CallContractData();
            callContractData.setContractAddress(contractAddressBytes);
            callContractData.setSender(senderBytes);
            callContractData.setValue(value.getValue());
            callContractData.setPrice(price.longValue());
            callContractData.setGasLimit(gasLimit.longValue());
            callContractData.setMethodName(methodName);
            callContractData.setMethodDesc(methodDesc);
            if (args != null) {
                callContractData.setArgsCount((byte) args.length);
                callContractData.setArgs(args);
            }
            tx.setTxData(callContractData);

            Na fee = accountLedgerService.getTxFee(senderBytes, totalNa, tx.size() +
coinData.size(), TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
            fee = fee.add(imputedGasUsedNa);
            return Result.getSuccess().setData(new Object[]{fee, tx});
        } catch (Exception e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.SYS_UNKOWN_EXCEPTION);
            result.setMsg(e.getMessage());
            return result;
        }
    }

    /**
     *
     *
     * @param sender
     * @param contractAddress
     * @param password
     * @param remark
     * @return
     */
    @Override
    public Result contractDeleteTx(String sender, String contractAddress,
                                   String password, String remark) {
        try {
```

```
        AssertUtil.canNotEmpty(sender, "the sender address can not be empty");
        AssertUtil.canNotEmpty(contractAddress, "the contractAddress can not be empty");

        byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);

        Result<ContractAddressInfoPo> contractAddressInfoPoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
        if(contractAddressInfoPoResult.isFailed()) {
            return contractAddressInfoPoResult;
        }
        ContractAddressInfoPo contractAddressInfoPo = contractAddressInfoPoResult.getData();
        if(contractAddressInfoPo == null) {
            return Result.getFailed(ContractErrorCode.CONTRACT_ADDRESS_NOT_EXIST);
        }

        BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
        //
        byte[] stateRoot = ContractUtil.getStateRoot(blockHeader);
        //
        ProgramStatus status = vmHelper.getContractStatus(stateRoot, contractAddressBytes);
        boolean isTerminatedContract = ContractUtil.isTerminatedContract(status.ordinal());
        if(isTerminatedContract) {
            return Result.getFailed(ContractErrorCode.CONTRACT_DELETED);
        }

        byte[] senderBytes = AddressTool.getAddress(sender);
        if(!ArraysTool.arrayEquals(senderBytes, contractAddressInfoPo.getSender())) {
            return Result.getFailed(ContractErrorCode.CONTRACT_DELETE_CREATER);
        }

        Result<ContractBalance> result =
contractBalanceManager.getBalance(contractAddressBytes);
        ContractBalance balance = (ContractBalance) result.getData();
        if(balance == null) {
            return result;
        }

        Na totalBalance = balance.getBalance();
        if(totalBalance.compareTo(Na.ZERO) != 0) {
            return Result.getFailed(ContractErrorCode.CONTRACT_DELETE_BALANCE);
        }
```

```
        Result<Account> accountResult = accountService.getAccount(sender);
        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);
            }
        }

        DeleteContractTransaction tx = new DeleteContractTransaction();
        if (StringUtils.isNotBlank(remark)) {
            try {
                tx.setRemark(remark.getBytes(NulsConfig.DEFAULT_ENCODING));
            } catch (UnsupportedEncodingException e) {
                Log.error(e);
                throw new RuntimeException(e);
            }
        }
        tx.setTime(TimeService.currentTimeMillis());

        // txData
        DeleteContractData deleteContractData = new DeleteContractData();
        deleteContractData.setContractAddress(contractAddressBytes);
        deleteContractData.setSender(senderBytes);

        tx.setTxData(deleteContractData);

        // CoinData
        /*
         * Gas
         */
        CoinData coinData = new CoinData();

        //
        CoinDataResult coinDataResult = accountLedgerService.getCoinData(senderBytes,
Na.ZERO, tx.size() + coinData.size(),
TransactionFeeCalculator.MIN_PRECE_PRE_1024_BYTES);
```

```
        if (!coinDataResult.isEnough()) {
            return Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE);
        }
        coinData.setFrom(coinDataResult.getCoinList());
        // UTXO
        if (coinDataResult.getChange() != null) {
            coinData.getTo().add(coinDataResult.getChange());
        }
        tx.setCoinData(coinData);
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));

        //
        List<ECKey> signEckeys = new ArrayList<>();
        List<ECKey> scriptEckeys = new ArrayList<>();
        ECKey eckey = account.getEcKey(password);
        //1
        if ((coinDataResult.getSignType() & 0x01) == 0x01) {
            signEckeys.add(eckey);
        }
        //1
        if ((coinDataResult.getSignType() & 0x02) == 0x02) {
            scriptEckeys.add(eckey);
        }
        SignatureUtil.createTransactionSignture(tx, scriptEckeys, signEckeys);

        //
        Result saveResult = accountLedgerService.verifyAndSaveUnconfirmedTransaction(tx);
        if (saveResult.isFailed()) {
            if
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
                //()
                Result rs = accountLedgerService.getMaxAmountOfOnce(senderBytes, 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()) {
                //
                accountLedgerService.deleteTransaction(tx);
                return sendResult;
            }
            return Result.getSuccess().setData(tx.getHash().getDigestHex());
        } catch (IOException e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        } catch (NulsException e) {
            Log.error(e);
            return Result.getFailed(e.getErrorCode());
        } catch (Exception e) {
            Log.error(e);
            Result result = Result.getFailed(ContractErrorCode.CONTRACT_TX_CREATE_ERROR);
            result.setMsg(e.getMessage());
            return result;
        }
    }

}

88:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\entity\BlockHeaderDto.java
 */
package io.nuls.contract.entity;


import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.model.BlockHeader;

import java.io.IOException;
import java.io.Serializable;

/**
 * @Desription:
```

```
 * @Author: PierreLuo
 * @Date: 2018/5/2
 */
public class BlockHeaderDto implements Serializable {

    private String hash;
    private String preHash;
    private long time;
    private long height;
    private long txCount;
    //23 bytes
    private byte[] packingAddress;
    private byte[] stateRoot;

    public BlockHeaderDto() {}

    public BlockHeaderDto(BlockHeader header) {
        this.hash = (header.getHash() == null ? null : header.getHash().getDigestHex());
        this.preHash = (header.getPreHash() == null ? null : header.getPreHash().getDigestHex());
        this.time = header.getTime();
        this.height = header.getHeight();
        this.txCount = header.getTxCount();
        this.packingAddress = header.getPackingAddress();
        this.stateRoot = ContractUtil.getStateRoot(header);
    }

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

    public String getPreHash() {
        return preHash;
    }

    public void setPreHash(String preHash) {
        this.preHash = preHash;
    }
```

```
    public long getTime() {
        return time;
    }

    public void setTime(long time) {
        this.time = time;
    }

    public long getHeight() {
        return height;
    }

    public void setHeight(long height) {
        this.height = height;
    }

    public long getTxCount() {
        return txCount;
    }

    public void setTxCount(long txCount) {
        this.txCount = txCount;
    }

    public byte[] getPackingAddress() {
        return packingAddress;
    }

    public void setPackingAddress(byte[] packingAddress) {
        this.packingAddress = packingAddress;
    }

    public byte[] getStateRoot() {
        return stateRoot;
    }

    public void setStateRoot(byte[] stateRoot) {
        this.stateRoot = stateRoot;
    }
}
```

89:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-

vm\src\main\java\io\nuls\contract\entity\ContractInfoDto.java

```
package io.nuls.contract.entity;

import io.nuls.contract.util.ContractUtil;
import io.nuls.contract.vm.program.ProgramMethod;

import java.math.BigInteger;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/15
 */
public class ContractInfoDto {

    private ProgramMethod constructor;
    private boolean isNrc20;

    public ProgramMethod getConstructor() {
        return constructor;
    }

    public void setConstructor(ProgramMethod constructor) {
        this.constructor = constructor;
    }

    public boolean isNrc20() {
        return isNrc20;
    }

    public void setNrc20(boolean nrc20) {
        isNrc20 = nrc20;
    }
}
```

90:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\helper\VMHelper.java

```
package io.nuls.contract.helper;

import io.nuls.account.service.AccountService;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.constant.ContractErrorCode;
```

```
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTokenTransferInfoPo;
import io.nuls.contract.entity.ContractInfoDto;
import io.nuls.contract.entity.tx.CreateContractTransaction;
import io.nuls.contract.entity.txdata.CreateContractData;
import io.nuls.contract.ledger.manager.ContractBalanceManager;
import io.nuls.contract.storage.po.ContractAddressInfoPo;
import io.nuls.contract.storage.service.ContractAddressStorageService;
import io.nuls.contract.storage.service.ContractTokenTransferStorageService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.contract.util.VMContext;
import io.nuls.contract.vm.program.*;
import io.nuls.contract.vm.program.impl.ProgramExecutorImpl;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.map.MapUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.db.service.DBService;
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.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.VarInt;

import java.math.BigInteger;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

import static io.nuls.contract.constant.ContractConstant.*;

@Component
public class VMHelper implements InitializingBean {

    @Autowired
    private VMContext vmContext;
```

```
    @Autowired
    private DBService dbService;
    @Autowired
    private AccountService accountService;
    @Autowired
    private ContractBalanceManager contractBalanceManager;
    @Autowired
    private ContractAddressStorageService contractAddressStorageService;
    @Autowired
    private ContractTokenTransferStorageService contractTokenTransferStorageService;

    private ProgramExecutor programExecutor;

    private ConcurrentHashMap<String, Long> accountLastedPriceMap =
MapUtil.createConcurrentHashMap(4);

    private static final BigInteger MAXIMUM_DECIMALS = BigInteger.valueOf(18L);
    private static final BigInteger MAXIMUM_TOTAL_SUPPLY =
BigInteger.valueOf(2L).pow(256).subtract(BigInteger.ONE);

    @Override
    public void afterPropertiesSet() throws NulsException {
        programExecutor = new ProgramExecutorImpl(vmContext, dbService);
    }

    public ProgramExecutor getProgramExecutor() {
        return programExecutor;
    }

    public boolean checkIsViewMethod(String methodName, String methodDesc, byte[]
contractAddressBytes) {
        ProgramMethod method = this.getMethodInfoByContractAddress(methodName,
methodDesc, contractAddressBytes);
        if(method == null) {
            return false;
        } else {
            return method.isView();
        }
    }

    public ProgramMethod getMethodInfoByCode(String methodName, String methodDesc, byte[]
code) {
```

```
        if(StringUtils.isBlank(methodName) || code == null) {
            return null;
        }
        List<ProgramMethod> methods = this.getAllMethods(code);
        return this.getMethodInfo(methodName, methodDesc, methods);
    }

    private ProgramMethod getMethodInfo(String methodName, String methodDesc,
List<ProgramMethod> methods) {
        if(methods != null && methods.size() > 0) {
            boolean emptyDesc = StringUtils.isBlank(methodDesc);
            for(ProgramMethod method : methods) {
                if(methodName.equals(method.getName())) {
                    if(emptyDesc) {
                        return method;
                    } else if(methodDesc.equals(method.getDesc())) {
                        return method;
                    }
                }
            }
        }
        return null;
    }

    public ProgramMethod getMethodInfoByContractAddress(String methodName, String
methodDesc, byte[] contractAddressBytes) {
        if(StringUtils.isBlank(methodName)) {
            return null;
        }
        BlockHeader header = NulsContext.getInstance().getBestBlock().getHeader();
        //
        byte[] currentStateRoot = ContractUtil.getStateRoot(header);

        ProgramExecutor track = programExecutor.begin(currentStateRoot);
        List<ProgramMethod> methods = track.method(contractAddressBytes);

        return this.getMethodInfo(methodName, methodDesc, methods);
    }

    private List<ProgramMethod> getAllMethods(byte[] contractCode) {
        return programExecutor.jarMethod(contractCode);
    }
```

```
    public ContractInfoDto getConstructor(byte[] contractCode) {
        try {
            ContractInfoDto dto = new ContractInfoDto();
            List<ProgramMethod> programMethods = this.getAllMethods(contractCode);
            if(programMethods == null || programMethods.size() == 0) {
                return null;
            }
            for(ProgramMethod method : programMethods) {
                if(ContractConstant.CONTRACT_CONSTRUCTOR.equals(method.getName())) {
                    dto.setConstructor(method);
                    break;
                }
            }
            dto.setNrc20(this.checkNrc20Contract(programMethods));
            return dto;
        } catch (Exception e) {
            Log.error(e);
            return null;
        }
    }

    public ProgramResult invokeViewMethod(byte[] contractAddressBytes, String methodName,
String methodDesc, Object... args) {
        return this.invokeViewMethod(contractAddressBytes, methodName, methodDesc,
ContractUtil.twoDimensionalArray(args));
    }

    public ProgramResult invokeViewMethod(byte[] contractAddressBytes, String methodName,
String methodDesc, String[][] args) {
        //
        BlockHeader blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
        long blockHeight = blockHeader.getHeight();
        //
        byte[] currentStateRoot = ContractUtil.getStateRoot(blockHeader);

        return this.invokeViewMethod(null, currentStateRoot, blockHeight, contractAddressBytes,
methodName, methodDesc, args);
    }

    private ProgramResult invokeViewMethod(ProgramExecutor executor, byte[] stateRoot, long
blockHeight, byte[] contractAddressBytes, String methodName, String methodDesc, Object... args)
```

```
{
        return this.invokeViewMethod(executor, stateRoot, blockHeight, contractAddressBytes,
methodName, methodDesc, ContractUtil.twoDimensionalArray(args));
    }

    public ProgramResult invokeViewMethod(byte[] stateRoot, long blockHeight, byte[]
contractAddressBytes, String methodName, String methodDesc, String[][] args) {
        return this.invokeViewMethod(null, stateRoot, blockHeight, contractAddressBytes,
methodName, methodDesc, args);
    }

    public ProgramResult invokeViewMethod(ProgramExecutor executor, byte[] stateRoot, long
blockHeight, byte[] contractAddressBytes, String methodName, String methodDesc, String[][] args)
{

        ProgramCall programCall = new ProgramCall();
        programCall.setContractAddress(contractAddressBytes);
        programCall.setValue(BigInteger.ZERO);
        programCall.setGasLimit(ContractConstant.CONTRACT_CONSTANT_GASLIMIT);
        programCall.setPrice(ContractConstant.CONTRACT_CONSTANT_PRICE);
        programCall.setNumber(blockHeight);
        programCall.setMethodName(methodName);
        programCall.setMethodDesc(methodDesc);
        programCall.setArgs(args);

        ProgramExecutor track;
        if(executor == null) {
            track = programExecutor.begin(stateRoot);
        } else {
            track = executor.startTracking();
        }
        ProgramResult programResult = track.call(programCall);

        return programResult;
    }

    public void updateLastedPriceForAccount(byte[] sender, long price) {
        if(price <= 0) {
            return;
        }
        String address = AddressTool.getStringAddressByBytes(sender);
        accountLastedPriceMap.put(address, price);
```

```
    }

    public long getLastedPriceForAccount(byte[] sender) {
        String address = AddressTool.getStringAddressByBytes(sender);
        Long price = accountLastedPriceMap.get(address);
        if(price == null) {
            price = ContractConstant.CONTRACT_MINIMUM_PRICE;
        }
        price = price < ContractConstant.CONTRACT_MINIMUM_PRICE ?
ContractConstant.CONTRACT_MINIMUM_PRICE : price;
        accountLastedPriceMap.put(address, price);
        return price;
    }

    public void dealEvents(byte[] newestStateRoot, Transaction tx, ContractResult contractResult,
ContractAddressInfoPo po) {
        if(po == null) {
            return;
        }
        try {
            List<String> events = contractResult.getEvents();
            int size = events.size();
            // Transfer, token
            String event;
            ContractAddressInfoPo contractAddressInfo;
            if(events != null && size > 0) {
                for(int i = 0; i < size; i++) {
                    event = events.get(i);
                    // NRC20TransferEvent-from-to,
                    ContractTokenTransferInfoPo tokenTransferInfoPo =
ContractUtil.convertJsonToTokenTransferInfoPo(event);
                    if(tokenTransferInfoPo == null) {
                        continue;
                    }
                    String contractAddress = tokenTransferInfoPo.getContractAddress();
                    if (StringUtils.isBlank(contractAddress)) {
                        continue;
                    }
                    if (!AddressTool.validAddress(contractAddress)) {
                        continue;
                    }
                    byte[] contractAddressBytes = AddressTool.getAddress(contractAddress);
```

```
                if(ArraysTool.arrayEquals(po.getContractAddress(), contractAddressBytes)) {
                    contractAddressInfo = po;
                } else {
                    Result<ContractAddressInfoPo> contractAddressInfoResult =
contractAddressStorageService.getContractAddressInfo(contractAddressBytes);
                    contractAddressInfo = contractAddressInfoResult.getData();
                }

                if(contractAddressInfo == null) {
                    continue;
                }
                // NRC20
                if(!contractAddressInfo.isNrc20()) {
                    continue;
                }
                byte[] txHashBytes;
                byte[] from = tokenTransferInfoPo.getFrom();
                byte[] to = tokenTransferInfoPo.getTo();
                tokenTransferInfoPo.setName(contractAddressInfo.getNrc20TokenName());
                tokenTransferInfoPo.setSymbol(contractAddressInfo.getNrc20TokenSymbol());
                tokenTransferInfoPo.setDecimals(contractAddressInfo.getDecimals());
                tokenTransferInfoPo.setTime(tx.getTime());
                tokenTransferInfoPo.setBlockHeight(tx.getBlockHeight());
                txHashBytes = tx.getHash().serialize();
                tokenTransferInfoPo.setTxHash(txHashBytes);
                tokenTransferInfoPo.setStatus((byte) (contractResult.isSuccess() ? 1 : 2));

                if(from != null) {
                    this.refreshTokenBalance(newestStateRoot, contractAddressInfo,
AddressTool.getStringAddressByBytes(from), contractAddress);
                    this.saveTokenTransferInfo(from, txHashBytes, new VarInt(i).encode(),
tokenTransferInfoPo);
                }
                if(to != null) {
                    this.refreshTokenBalance(newestStateRoot, contractAddressInfo,
AddressTool.getStringAddressByBytes(to), contractAddress);
                    this.saveTokenTransferInfo(to, txHashBytes, new VarInt(i).encode(),
tokenTransferInfoPo);
                }
            }
        }
    } catch (Exception e) {
```

```
            Log.warn("contract event parse error.", e);
        }
    }

    private void saveTokenTransferInfo(byte[] address, byte[] txHashBytes, byte[] index,
ContractTokenTransferInfoPo tokenTransferInfoPo) {
contractTokenTransferStorageService.saveTokenTransferInfo(ArraysTool.concatenate(address, tx
HashBytes, index), tokenTransferInfoPo);
    }

    public void refreshTokenBalance(byte[] stateRoot, ContractAddressInfoPo po, String address,
String contractAddress) {
        this.refreshTokenBalance(null, stateRoot, po, address, contractAddress);
    }

    private void refreshTokenBalance(ProgramExecutor executor, byte[] stateRoot,
ContractAddressInfoPo po, String address, String contractAddress) {
        long bestBlockHeight = NulsContext.getInstance().getBestHeight();
        byte[] contractAddressBytes = po.getContractAddress();
        ProgramResult programResult = this.invokeViewMethod(executor, stateRoot,
bestBlockHeight, contractAddressBytes, NRC20_METHOD_BALANCE_OF, null, address);
        if(!programResult.isSuccess()) {
            return;
        } else {
            contractBalanceManager.refreshContractToken(address, contractAddress, po, new
BigInteger(programResult.getResult()));
        }

    }

    private boolean checkNrc20Contract(List<ProgramMethod> methods) {
        if(methods == null || methods.size() == 0) {
            return false;
        }
        Map<String, ProgramMethod> contractMethodsMap =
MapUtil.createHashMap(methods.size());
        for(ProgramMethod method : methods) {
            contractMethodsMap.put(method.getName(), method);
        }

        Set<Map.Entry<String, ProgramMethod>> entries =
VMContext.getNrc20Methods().entrySet();
```

```
        String methodName;
        ProgramMethod standardMethod;
        ProgramMethod mappingMethod;
        for(Map.Entry<String, ProgramMethod> entry : entries) {
            methodName = entry.getKey();
            standardMethod = entry.getValue();
            mappingMethod = contractMethodsMap.get(methodName);

            if(mappingMethod == null) {
                return false;
            }
            if(!standardMethod.equalsNrc20Method(mappingMethod)) {
                return false;
            }
        }

        return true;
    }

    //private boolean checkNrc20Contract(byte[] contractCode) {
    //    List<ProgramMethod> methods = programExecutor.jarMethod(contractCode);
    //    if(methods == null || methods.size() == 0) {
    //        return false;
    //    }
    //    return checkNrc20Contract(methods);
    //}

    private boolean checkAcceptDirectTransfer(List<ProgramMethod> methods) {
        if(methods == null || methods.size() == 0) {
            return false;
        }
        for(ProgramMethod method : methods) {
            if(ContractConstant.BALANCE_TRIGGER_METHOD_NAME.equals(method.getName())) {
                return method.isPayable();
            }
        }
        return false;
    }

    public Result validateNrc20Contract(ProgramExecutor track, CreateContractTransaction tx,
ContractResult contractResult) {
        if(contractResult == null) {
```

```
            return Result.getFailed(ContractErrorCode.NULL_PARAMETER);
        }
        CreateContractData createContractData = tx.getTxData();
        byte[] stateRoot = contractResult.getStateRoot();
        byte[] contractAddress = contractResult.getContractAddress();
        long bestBlockHeight = NulsContext.getInstance().getBestHeight();
        List<ProgramMethod> methods = this.getAllMethods(createContractData.getCode());
        boolean isNrc20 = this.checkNrc20Contract(methods);
        boolean isAcceptDirectTransfer = this.checkAcceptDirectTransfer(methods);
        contractResult.setNrc20(isNrc20);
        contractResult.setAcceptDirectTransfer(isAcceptDirectTransfer);
        if(isNrc20) {
            // NRC20 tokenName
            ProgramResult programResult = this.invokeViewMethod(track, stateRoot,
bestBlockHeight, contractAddress, NRC20_METHOD_NAME, null, null);
            if(programResult.isSuccess()) {
                String tokenName = programResult.getResult();
                if(StringUtils.isNotBlank(tokenName)) {
                    if(!StringUtils.validTokenNameOrSymbol(tokenName)) {
                        return
Result.getFailed(ContractErrorCode.CONTRACT_NAME_FORMAT_INCORRECT);
                    }
                }
            }
            // NRC20 tokenSymbol
            programResult = this.invokeViewMethod(track, stateRoot, bestBlockHeight,
contractAddress, NRC20_METHOD_SYMBOL, null, null);
            if(programResult.isSuccess()) {
                String symbol = programResult.getResult();
                if(StringUtils.isNotBlank(symbol)) {
                    if(!StringUtils.validTokenNameOrSymbol(symbol)) {
                        return
Result.getFailed(ContractErrorCode.CONTRACT_NRC20_SYMBOL_FORMAT_INCORRECT);
                    }
                }
            }

            programResult = this.invokeViewMethod(track, stateRoot, bestBlockHeight,
contractAddress, NRC20_METHOD_DECIMALS, null, null);
            BigInteger decimalsBig = BigInteger.ZERO;
            if(programResult.isSuccess()) {
                String decimals = programResult.getResult();
```

```
            if(StringUtils.isNotBlank(decimals)) {
                try {
                    decimalsBig = new BigInteger(decimals);
                    if(decimalsBig.compareTo(BigInteger.ZERO) < 0 ||
decimalsBig.compareTo(MAXIMUM_DECIMALS) > 0) {
                        return
Result.getFailed(ContractErrorCode.CONTRACT_NRC20_MAXIMUM_DECIMALS);
                    }
                } catch (Exception e) {
                    Log.error("Get nrc20 decimals error.", e);
                    // skip it
                }
            }
        }
        programResult = this.invokeViewMethod(track, stateRoot, bestBlockHeight,
contractAddress, NRC20_METHOD_TOTAL_SUPPLY, null, null);
        if(programResult.isSuccess()) {
            String totalSupply = programResult.getResult();
            if(StringUtils.isNotBlank(totalSupply)) {
                try {
                    BigInteger totalSupplyBig = new BigInteger(totalSupply);
                    if(totalSupplyBig.compareTo(BigInteger.ZERO) <= 0 ||
totalSupplyBig.compareTo(MAXIMUM_TOTAL_SUPPLY.multiply(BigInteger.TEN.pow(decimalsBi
g.intValue()))) > 0) {
                        return
Result.getFailed(ContractErrorCode.CONTRACT_NRC20_MAXIMUM_TOTAL_SUPPLY);
                    }
                } catch (Exception e) {
                    Log.error("Get nrc20 totalSupply error.", e);
                    // skip it
                }
            }
        }
      }
      return Result.getSuccess();
   }

   public ProgramStatus getContractStatus(byte[] stateRoot, byte[] contractAddress) {
      ProgramExecutor track = programExecutor.begin(stateRoot);
      return track.status(contractAddress);
   }
}
```

91:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\util\VMContext.java
 */
```
package io.nuls.contract.util;


import io.nuls.contract.entity.BlockHeaderDto;
import io.nuls.contract.ledger.module.ContractBalance;
import io.nuls.contract.ledger.service.ContractUtxoService;
import io.nuls.contract.vm.program.ProgramMethod;
import io.nuls.core.tools.str.StringUtils;
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.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.protocol.service.BlockService;

import java.io.IOException;
import java.math.BigInteger;
import java.util.Map;

/**
 * @Desription:
 * @Author: PierreLuo
 * @Date: 2018/5/2
 */
@Component
public class VMContext {

    @Autowired
    private BlockService blockService;

    @Autowired
    private ContractUtxoService contractUtxoService;

    private ThreadLocal<BlockHeader> currentBlockHeader = new ThreadLocal<>();

    public static Map<String, ProgramMethod> NRC20_METHODS = null;
```

```
/**
 * @param hash
 * @return
 * @throws NulsException
 * @throws IOException
 */
public BlockHeaderDto getBlockHeader(String hash) throws NulsException, IOException {
    if(StringUtils.isBlank(hash)) {
        return null;
    }
    NulsDigestData nulsDigestData = NulsDigestData.fromDigestHex(hash);
    Result<BlockHeader> blockHeaderResult = blockService.getBlockHeader(nulsDigestData);
    if(blockHeaderResult == null || blockHeaderResult.getData() == null) {
        return null;
    }
    BlockHeaderDto header = new BlockHeaderDto(blockHeaderResult.getData());
    return header;
}

/**
 * @param height
 * @return
 * @throws NulsException
 * @throws IOException
 */
public BlockHeaderDto getBlockHeader(long height) throws NulsException, IOException {
    if(height < 0L) {
        return null;
    }
    Result<BlockHeader> blockHeaderResult = blockService.getBlockHeader(height);
    if(blockHeaderResult == null || blockHeaderResult.getData() == null) {
        return null;
    }
    BlockHeaderDto header = new BlockHeaderDto(blockHeaderResult.getData());
    return header;
}

/**
 * get the newest block header
 * @return
 * @throws IOException
```

```
 */
public BlockHeaderDto getNewestBlockHeader()  {
    return new BlockHeaderDto(NulsContext.getInstance().getBestBlock().getHeader());
}

/**
 * get the current block header
 * @return
 * @throws IOException
 */
public BlockHeaderDto getCurrentBlockHeader()  {
    BlockHeader blockHeader = currentBlockHeader.get();
    if(blockHeader == null) {
        blockHeader = NulsContext.getInstance().getBestBlock().getHeader();
    }
    return new BlockHeaderDto(blockHeader);
}

/**
 *
 * @param address
 * @param blockHeight , ,
 */
public BigInteger getBalance(byte[] address, Long blockHeight) {
    Result<ContractBalance> result = contractUtxoService.getBalance(address, blockHeight);
    if(result.isSuccess()) {
        ContractBalance balance = result.getData();
        // pierre test comment out
        return BigInteger.valueOf(balance.getRealUsable().getValue());
    }
    return BigInteger.ZERO;
}

/**
 *
 * @param address
 * @param blockHeight , ,
 */
public BigInteger getTotalBalance(byte[] address, Long blockHeight) {
    Result<ContractBalance> result = contractUtxoService.getBalance(address, blockHeight);
    if(result.isSuccess()) {
        ContractBalance balance = result.getData();
```

```
            return BigInteger.valueOf(balance.getBalance().getValue());
        }
        return BigInteger.ZERO;
    }

    public static Map<String, ProgramMethod> getNrc20Methods() {
        return NRC20_METHODS;
    }

    public static void setNrc20Methods(Map<String, ProgramMethod> nrc20Methods) {
        NRC20_METHODS = nrc20Methods;
    }

    public void createCurrentBlockHeader(BlockHeader tempHeader) {
        currentBlockHeader.remove();
        currentBlockHeader.set(tempHeader);
    }

    public void removeCurrentBlockHeader() {
        currentBlockHeader.remove();
    }
}

92:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\code\ClassCode.java
 */
package io.nuls.contract.vm.code;

import io.nuls.contract.vm.util.Constants;
import org.apache.commons.collections4.ListUtils;
import org.apache.commons.lang3.StringUtils;
import org.objectweb.asm.Attribute;
import org.objectweb.asm.Opcodes;
import org.objectweb.asm.tree.*;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import static io.nuls.contract.vm.util.Utils.arrayListInitialCapacity;
import static io.nuls.contract.vm.util.Utils.hashMapInitialCapacity;
```

```
public class ClassCode {

    /**
     * The class version. The minor version is stored in the 16 most significant bits, and the major
     * version in the 16 least significant bits.
     */
    public final int version;

    /**
     * The class's access flags (see {@link org.objectweb.asm.Opcodes}). This field also indicates if
     * the class is deprecated.
     */
    public final int access;

    /**
     * The internal name of this class (see {@link org.objectweb.asm.Type#getInternalName}).
     */
    public final String name;

    /**
     * The signature of this class. May be <tt>null</tt>.
     */
    public final String signature;

    /**
     * The internal of name of the super class (see {@link
org.objectweb.asm.Type#getInternalName}).
     * For interfaces, the super class is {@link Object}. May be <tt>null</tt>, but only for the
     * {@link Object} class.
     */
    public final String superName;

    /**
     * The internal names of the interfaces directly implemented by this class (see {@link
     * org.objectweb.asm.Type#getInternalName}).
     */
    public final List<String> interfaces;

    /**
     * The name of the source file from which this class was compiled. May be <tt>null</tt>.
     */
```

```
    public final String sourceFile;

    /**
     * The correspondence between source and compiled elements of this class. May be
<tt>null</tt>.
     */
    public final String sourceDebug;

    /**
     * The module stored in this class. May be <tt>null</tt>.
     */
    public final ModuleNode module;

    /**
     * The internal name of the enclosing class of this class. May be <tt>null</tt>.
     */
    public final String outerClass;

    /**
     * The name of the method that contains this class, or <tt>null</tt> if this class is not enclosed
     * in a method.
     */
    public final String outerMethod;

    /**
     * The descriptor of the method that contains this class, or <tt>null</tt> if this class is not
     * enclosed in a method.
     */
    public final String outerMethodDesc;

    /**
     * The runtime visible annotations of this class. May be <tt>null</tt>.
     */
    public final List<AnnotationNode> visibleAnnotations;

    /**
     * The runtime invisible annotations of this class. May be <tt>null</tt>.
     */
    public final List<AnnotationNode> invisibleAnnotations;

    /**
     * The runtime visible type annotations of this class. May be <tt>null</tt>.
```

```
     */
    public final List<TypeAnnotationNode> visibleTypeAnnotations;

    /**
     * The runtime invisible type annotations of this class. May be <tt>null</tt>.
     */
    public final List<TypeAnnotationNode> invisibleTypeAnnotations;

    /**
     * The non standard attributes of this class. May be <tt>null</tt>.
     */
    public final List<Attribute> attrs;

    /**
     * The inner classes of this class.
     */
    public final List<InnerClassNode> innerClasses;

    /**
     * <b>Experimental, use at your own risk. This field will be renamed when it becomes stable, this
     * will break existing code using it</b>. The internal name of the nest host class of this class.
     * May be <tt>null</tt>.
     */
    public final String nestHostClassExperimental;

    /**
     * <b>Experimental, use at your own risk. This field will be renamed when it becomes stable, this
     * will break existing code using it</b>. The internal names of the nest members of this class.
     * May be <tt>null</tt>.
     */
    public final List<String> nestMembersExperimental;

    /**
     * The fields of this class.
     */
    //public final List<FieldNode> fields;
    public final Map<String, FieldCode> fields;

    /**
     * The methods of this class.
```

```
 */
//public final List<MethodNode> methods;
public final List<MethodCode> methods;
private final Map<String, MethodCode> methodMap;

public final VariableType variableType;

public final boolean isInterface;

public final boolean isSuper;

public final boolean isAbstract;

public final boolean isV1_6;

public final boolean isV1_8;

public final String simpleName;

public ClassCode(ClassNode classNode) {
    version = classNode.version;
    access = classNode.access;
    name = classNode.name;
    signature = classNode.signature;
    superName = classNode.superName;
    interfaces = ListUtils.emptyIfNull(classNode.interfaces);
    sourceFile = classNode.sourceFile;
    sourceDebug = classNode.sourceDebug;
    module = classNode.module;
    outerClass = classNode.outerClass;
    outerMethod = classNode.outerMethod;
    outerMethodDesc = classNode.outerMethodDesc;
    visibleAnnotations = ListUtils.emptyIfNull(classNode.visibleAnnotations);
    invisibleAnnotations = ListUtils.emptyIfNull(classNode.invisibleAnnotations);
    visibleTypeAnnotations = ListUtils.emptyIfNull(classNode.visibleTypeAnnotations);
    invisibleTypeAnnotations = ListUtils.emptyIfNull(classNode.invisibleTypeAnnotations);
    attrs = ListUtils.emptyIfNull(classNode.attrs);
    innerClasses = ListUtils.emptyIfNull(classNode.innerClasses);
    nestHostClassExperimental = classNode.nestHostClassExperimental;
    nestMembersExperimental = ListUtils.emptyIfNull(classNode.nestMembersExperimental);
    //fields = ListUtils.emptyIfNull(classNode.fields);
    //methods = ListUtils.emptyIfNull(classNode.methods);
```

```
    final List<FieldNode> fieldNodes = ListUtils.emptyIfNull(classNode.fields);
    fields = new LinkedHashMap<>(hashMapInitialCapacity(fieldNodes.size()));
    for (FieldNode fieldNode : fieldNodes) {
        final FieldCode fieldCode = new FieldCode(fieldNode);
        fields.put(fieldCode.name, fieldCode);
    }
    final List<MethodNode> methodNodes = ListUtils.emptyIfNull(classNode.methods);
    methods = new ArrayList<>(arrayListInitialCapacity(methodNodes.size()));
    methodMap = new LinkedHashMap<>(hashMapInitialCapacity(methodNodes.size() * 2));
    for (MethodNode methodNode : methodNodes) {
        final MethodCode methodCode = new MethodCode(this, methodNode);
        methods.add(methodCode);
        methodMap.put(methodCode.nameDesc, methodCode);
        if (!methodMap.containsKey(methodCode.name)) {
            methodMap.put(methodCode.name, methodCode);
        }
    }
    variableType = VariableType.valueOf(name);
    isInterface = (access & Opcodes.ACC_INTERFACE) != 0;
    isSuper = (access & Opcodes.ACC_SUPER) != 0;
    isAbstract = (access & Opcodes.ACC_ABSTRACT) != 0;
    isV1_6 = (version & Opcodes.V1_6) != 0;
    isV1_8 = (version & Opcodes.V1_8) != 0;
    simpleName = getSimpleName();
}

public MethodCode getMethodCode(String methodName, String methodDesc) {
    if (StringUtils.isEmpty(methodDesc)) {
        return this.methodMap.get(methodName);
    } else {
        return this.methodMap.get(methodName + methodDesc);
    }
}

private String getSimpleName() {
    int i = this.name.lastIndexOf(Constants.DOLLAR);
    if (i > 0) {
        return this.name.substring(i + 1);
    } else {
        i = this.name.lastIndexOf(Constants.CLASS_SEPARATOR);
        if (i > 0) {
            return this.name.substring(i + 1);
```

```
            } else {
                return this.name;
            }
        }
    }

    public boolean isSyntheticField(String fieldName) {
        FieldCode fieldCode = fields.get(fieldName);
        return fieldCode != null && fieldCode.isSynthetic;
    }

}

93:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\code\ClassCodeCacheKey.java
 */
package io.nuls.contract.vm.code;

import org.apache.commons.codec.digest.DigestUtils;

public class ClassCodeCacheKey {

    private final byte[] bytes;
    private final String key;

    public ClassCodeCacheKey(byte[] bytes) {
        this.bytes = bytes;
        this.key = DigestUtils.sha1Hex(bytes);
    }

    public byte[] getBytes() {
        return bytes;
    }

    public String getKey() {
        return key;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
```

```
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        ClassCodeCacheKey that = (ClassCodeCacheKey) o;

        return key != null ? key.equals(that.key) : that.key == null;
    }

    @Override
    public int hashCode() {
        return key != null ? key.hashCode() : 0;
    }

}

94:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\code\ClassCodeLoader.java
 */
package io.nuls.contract.vm.code;

import com.google.common.cache.CacheBuilder;
import com.google.common.cache.CacheLoader;
import com.google.common.cache.LoadingCache;
import io.nuls.contract.vm.util.Constants;
import org.apache.commons.io.FileUtils;
import org.apache.commons.io.IOUtils;
import org.apache.commons.lang3.StringUtils;
import org.objectweb.asm.ClassReader;
import org.objectweb.asm.tree.ClassNode;

import javax.annotation.Nonnull;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
```

```
import java.util.function.Function;
import java.util.jar.JarEntry;
import java.util.jar.JarInputStream;

public class ClassCodeLoader {

    private static final Map<String, ClassCode> RESOURCE_CLASS_CODES;

    private static final LoadingCache<ClassCodeCacheKey, Map<String, ClassCode>> CACHE;

    static {
        CACHE = CacheBuilder.newBuilder()
                .initialCapacity(100)
                .maximumSize(1024)
                .expireAfterAccess(10 * 60, TimeUnit.SECONDS)
                .build(new CacheLoader<ClassCodeCacheKey, Map<String, ClassCode>>() {
                    @Override
                    public Map<String, ClassCode> load(@Nonnull final ClassCodeCacheKey cacheKey)
{
                        return ClassCodeLoader.loadJar(cacheKey.getBytes());
                    }
                });
        RESOURCE_CLASS_CODES = loadFromResource();
    }

    public static ClassCode load(String className) {
        try {
            ClassReader classReader = new ClassReader(className);
            return load(classReader);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static ClassCode loadFromResource(String className) {
        ClassCode classCode = RESOURCE_CLASS_CODES.get(className);
        if (classCode == null) {
            throw new RuntimeException("can't load class " + className);
        } else {
            return classCode;
        }
    }
```

```
    public static ClassCode getFromResource(String className) {
        return RESOURCE_CLASS_CODES.get(className);
    }

    public static ClassCode loadFromResourceOrTmp(String className) {
        ClassCode classCode = RESOURCE_CLASS_CODES.get(className);
        if (classCode == null) {
            try {
                File file = new File("/tmp/classes/" + className + ".class");
                if (file.exists()) {
                    byte[] bytes = FileUtils.readFileToByteArray(file);
                    return load(bytes);
                } else {
                    throw new RuntimeException("can't load class " + className);
                }
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        } else {
            return classCode;
        }
    }

    public static void load(Map<String, ClassCode> classCodes, String className,
Function<String, ClassCode> loader) {
        if (!classCodes.containsKey(className)) {
            ClassCode classCode = loader.apply(className);
            classCodes.put(className, classCode);
            if (StringUtils.isNotEmpty(classCode.superName)) {
                load(classCodes, classCode.superName, loader);
            }
            for (String interfaceName : classCode.interfaces) {
                load(classCodes, interfaceName, loader);
            }
            for (MethodCode methodCode : classCode.methods) {
                if (isSupport(methodCode.returnVariableType)) {
                    load(classCodes, methodCode.returnVariableType.getType(), loader);
                }
                for (VariableType variableType : methodCode.argsVariableType) {
                    if (isSupport(variableType)) {
                        load(classCodes, variableType.getType(), loader);
```

```
                }
            }
        }
    }
}

public static Map<String, ClassCode> loadAll(String className, Function<String, ClassCode>
loader) {
    Map<String, ClassCode> classCodes = new LinkedHashMap<>(100);
    load(classCodes, className, loader);
    return classCodes;
}

public static Map<String, ClassCode> loadJarCache(byte[] bytes) {
    try {
        return CACHE.get(new ClassCodeCacheKey(bytes));
    } catch (ExecutionException e) {
        throw new RuntimeException(e);
    }
}

private static boolean isSupport(VariableType variableType) {
    if (variableType.isPrimitiveType()) {
        return false;
    } else if (variableType.isVoid()) {
        return false;
    } else {
        return true;
    }
}

private static ClassCode load(byte[] bytes) {
    return load(new ClassReader(bytes));
}

private static ClassCode load(ClassReader classReader) {
    ClassNode classNode = new ClassNode();
    classReader.accept(classNode, 0);
    ClassCode classCode = new ClassCode(classNode);
    return classCode;
}
```

```
    private static Map<String, ClassCode> loadFromResource() {
        InputStream inputStream = ClassCodeLoader.class.getResourceAsStream("/used_classes");
        if (inputStream == null) {
            return new HashMap<>();
        } else {
            return loadJar(inputStream);
        }
    }

    private static Map<String, ClassCode> loadJar(byte[] bytes) {
        InputStream inputStream = new ByteArrayInputStream(bytes);
        return loadJar(inputStream);
    }

    private static Map<String, ClassCode> loadJar(InputStream inputStream) {
        try {
            JarInputStream jarInputStream = new JarInputStream(inputStream);
            return loadJar(jarInputStream);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private static Map<String, ClassCode> loadJar(JarInputStream jarInputStream) {
        Map<String, ClassCode> map = new HashMap<>(100);
        try {
            JarEntry jarEntry;
            while ((jarEntry = jarInputStream.getNextJarEntry()) != null) {
                if (!jarEntry.isDirectory() && jarEntry.getName().endsWith(Constants.CLASS_SUFFIX)) {
                    byte[] bytes = IOUtils.toByteArray(jarInputStream);
                    ClassCode classCode = load(bytes);
                    map.put(classCode.name, classCode);
                }
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        return map;
    }

}
```

95:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\code\ClassCodes.java
*/

```
package io.nuls.contract.vm.code;

import java.util.Map;

public class ClassCodes {

    private final Map<String, ClassCode> classCodeMap;

    public ClassCodes(Map<String, ClassCode> classCodeMap) {
        this.classCodeMap = classCodeMap;
    }

    public boolean instanceOf(final ClassCode classCode, final String interfaceName) {
        if (classCode.interfaces.contains(interfaceName)) {
            return true;
        } else {
            if (classCode.superName != null) {
                final ClassCode superClassCode = classCodeMap.get(classCode.superName);
                if (superClassCode != null) {
                    return instanceOf(superClassCode, interfaceName);
                } else {
                    return false;
                }
            } else {
                return false;
            }
        }
    }

}
```

96:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-vm\src\main\java\io\nuls\contract\vm\code\Descriptors.java
*/

```
package io.nuls.contract.vm.code;

import com.google.common.collect.BiMap;
import com.google.common.collect.HashBiMap;
```

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

public class Descriptors {

    public static final BiMap<String, String> DESCRIPTORS;

    public static final String VOID = "void";
    public static final String BYTE = "byte";
    public static final String CHAR = "char";
    public static final String DOUBLE = "double";
    public static final String FLOAT = "float";
    public static final String INT = "int";
    public static final String LONG = "long";
    public static final String SHORT = "short";
    public static final String BOOLEAN = "boolean";

    public static final String DOUBLE_DESC = "D";
    public static final String LONG_DESC = "J";

    static {
        DESCRIPTORS = HashBiMap.create();
        DESCRIPTORS.put(VOID, "V");
        DESCRIPTORS.put(BYTE, "B");
        DESCRIPTORS.put(CHAR, "C");
        DESCRIPTORS.put(DOUBLE, DOUBLE_DESC);
        DESCRIPTORS.put(FLOAT, "F");
        DESCRIPTORS.put(INT, "I");
        DESCRIPTORS.put(LONG, LONG_DESC);
        DESCRIPTORS.put(SHORT, "S");
        DESCRIPTORS.put(BOOLEAN, "Z");
    }

    public static List<String> parse(String desc) {
        return parse(desc, false);
    }

    public static List<String> parse(String desc, boolean includeReturn) {
        boolean isL = false;
        boolean isEnd = false;
        StringBuilder sb = new StringBuilder();
        List<String> descList = new ArrayList<>();
```

```
for (char c : desc.toCharArray()) {
    if ('[' == c) {
        sb.append(c);
    } else if ('(' == c) {
        //
    } else if (')' == c) {
        isEnd = true;
    } else if (';' == c) {
        isEnd = true;
        sb.append(c);
    } else if (isL) {
        sb.append(c);
    } else if ('L' == c) {
        isL = true;
        sb.append(c);
    } else if (DESCRIPTORS.inverse().containsKey(String.valueOf(c))) {
        isEnd = true;
        sb.append(c);
    } else {
        throw new IllegalArgumentException("unknown desc");
    }

    if (isEnd) {
        if (sb.length() > 0) {
            descList.add(sb.toString());
            sb = new StringBuilder();
        }
        isL = false;
        isEnd = false;
    }

    if (')' == c) {
        if (includeReturn) {
            //
        } else {
            break;
        }
    }

}
```

```
        return descList;
    }

}

97:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\code\FieldCode.java
 */
package io.nuls.contract.vm.code;

import org.apache.commons.collections4.ListUtils;
import org.objectweb.asm.Attribute;
import org.objectweb.asm.Opcodes;
import org.objectweb.asm.tree.AnnotationNode;
import org.objectweb.asm.tree.FieldNode;
import org.objectweb.asm.tree.TypeAnnotationNode;

import java.util.List;

public class FieldCode {

    /**
     * The field's access flags (see {@link org.objectweb.asm.Opcodes}). This field also indicates if
     * the field is synthetic and/or deprecated.
     */
    public final int access;

    /**
     * The field's name.
     */
    public final String name;

    /**
     * The field's descriptor (see {@link org.objectweb.asm.Type}).
     */
    public final String desc;

    /**
     * The field's signature. May be <tt>null</tt>.
     */
    public final String signature;
```

```
/**
 * The field's initial value. This field, which may be <tt>null</tt> if the field does not have an
 * initial value, must be an {@link Integer}, a {@link Float}, a {@link Long}, a {@link Double} or
 * a {@link String}.
 */
public final Object value;

/**
 * The runtime visible annotations of this field. May be <tt>null</tt>.
 */
public final List<AnnotationNode> visibleAnnotations;

/**
 * The runtime invisible annotations of this field. May be <tt>null</tt>.
 */
public final List<AnnotationNode> invisibleAnnotations;

/**
 * The runtime visible type annotations of this field. May be <tt>null</tt>.
 */
public final List<TypeAnnotationNode> visibleTypeAnnotations;

/**
 * The runtime invisible type annotations of this field. May be <tt>null</tt>.
 */
public final List<TypeAnnotationNode> invisibleTypeAnnotations;

/**
 * The non standard attributes of this field. * May be <tt>null</tt>.
 */
public final List<Attribute> attrs;

public final VariableType variableType;

public final boolean isStatic;

public final boolean isFinal;

public final boolean isSynthetic;

public FieldCode(FieldNode fieldNode) {
    access = fieldNode.access;
```

```
        name = fieldNode.name;
        desc = fieldNode.desc;
        signature = fieldNode.signature;
        value = fieldNode.value;
        visibleAnnotations = ListUtils.emptyIfNull(fieldNode.visibleAnnotations);
        invisibleAnnotations = ListUtils.emptyIfNull(fieldNode.invisibleAnnotations);
        visibleTypeAnnotations = ListUtils.emptyIfNull(fieldNode.visibleTypeAnnotations);
        invisibleTypeAnnotations = ListUtils.emptyIfNull(fieldNode.invisibleTypeAnnotations);
        attrs = ListUtils.emptyIfNull(fieldNode.attrs);
        //
        variableType = VariableType.valueOf(desc);
        isStatic = (access & Opcodes.ACC_STATIC) != 0;
        isFinal = (access & Opcodes.ACC_FINAL) != 0;
        isSynthetic = (access & Opcodes.ACC_SYNTHETIC) != 0;
    }

}

98:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\code\LocalVariableCode.java
 */
package io.nuls.contract.vm.code;

import org.objectweb.asm.tree.LabelNode;
import org.objectweb.asm.tree.LocalVariableNode;

public class LocalVariableCode {

    /**
     * The name of a local variable.
     */
    public final String name;

    /**
     * The type descriptor of this local variable.
     */
    public final String desc;

    /**
     * The signature of this local variable. May be <tt>null</tt>.
     */
    public final String signature;
```

```
    /**
     * The first instruction corresponding to the scope of this local variable (inclusive).
     */
    public final LabelNode start;

    /**
     * The last instruction corresponding to the scope of this local variable (exclusive).
     */
    public final LabelNode end;

    /**
     * The local variable's index.
     */
    public final int index;

    public final VariableType variableType;

    public LocalVariableCode(LocalVariableNode localVariableNode) {
        name = localVariableNode.name;
        desc = localVariableNode.desc;
        signature = localVariableNode.signature;
        start = localVariableNode.start;
        end = localVariableNode.end;
        index = localVariableNode.index;
        //
        variableType = VariableType.valueOf(desc);
    }

}

99:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\code\MethodCode.java
 */
package io.nuls.contract.vm.code;

import com.google.common.base.Joiner;
import io.nuls.contract.vm.program.ProgramMethodArg;
import io.nuls.contract.vm.program.impl.ProgramDescriptors;
import io.nuls.contract.vm.util.Constants;
import org.apache.commons.collections4.ListUtils;
import org.objectweb.asm.Attribute;
```

```
import org.objectweb.asm.Opcodes;
import org.objectweb.asm.Type;
import org.objectweb.asm.tree.*;

import java.util.ArrayList;
import java.util.List;

import static io.nuls.contract.vm.util.Utils.arrayListInitialCapacity;

public class MethodCode {

    public static final String VIEW_ANNOTATION_DESC = "Lio/nuls/contract/sdk/annotation/View;";
    public static final String PAYABLE_ANNOTATION_DESC =
"Lio/nuls/contract/sdk/annotation/Payable;";
    public static final String REQUIRED_ANNOTATION_DESC =
"Lio/nuls/contract/sdk/annotation/Required;";

    /**
     * The method's access flags (see {@link Opcodes}). This field also indicates if the method is
     * synthetic and/or deprecated.
     */
    public final int access;

    /**
     * The method's name.
     */
    public final String name;

    /**
     * The method's descriptor (see {@link Type}).
     */
    public final String desc;

    /**
     * The method's signature. May be <tt>null</tt>.
     */
    public final String signature;

    /**
     * The internal names of the method's exception classes (see {@link Type#getInternalName()}).
     */
    public final List<String> exceptions;
```

```
/**
 * The method parameter info (access flags and name)
 */
public final List<ParameterNode> parameters;

/**
 * The runtime visible annotations of this method. May be <tt>null</tt>.
 */
public final List<AnnotationNode> visibleAnnotations;

/**
 * The runtime invisible annotations of this method. May be <tt>null</tt>.
 */
public final List<AnnotationNode> invisibleAnnotations;

/**
 * The runtime visible type annotations of this method. May be <tt>null</tt>.
 */
public final List<TypeAnnotationNode> visibleTypeAnnotations;

/**
 * The runtime invisible type annotations of this method. May be <tt>null</tt>.
 */
public final List<TypeAnnotationNode> invisibleTypeAnnotations;

/**
 * The non standard attributes of this method. May be <tt>null</tt>.
 */
public final List<Attribute> attrs;

/**
 * The default value of this annotation interface method. This field must be a {@link Byte},
 * {@link Boolean}, {@link Character}, {@link Short}, {@link Integer}, {@link Long}, {@link
 * Float}, {@link Double}, {@link String} or {@link Type}, or an two elements String array (for
 * enumeration values), a {@link AnnotationNode}, or a {@link List} of values of one of the
 * preceding types. May be <tt>null</tt>.
 */
public final Object annotationDefault;

/**
 * The number of method parameters than can have runtime visible annotations. This number
```

```
must be
     * less or equal than the number of parameter types in the method descriptor (the default value
0
     * indicates that all the parameters described in the method descriptor can have annotations). It
     * can be strictly less when a method has synthetic parameters and when these parameters are
     * ignored when computing parameter indices for the purpose of parameter annotations (see
     * https://docs.oracle.com/javase/specs/jvms/se9/html/jvms-4.html#jvms-4.7.18).
     */
    public final int visibleAnnotableParameterCount;

    /**
     * The runtime visible parameter annotations of this method. These lists are lists of {@link
     * AnnotationNode} objects. May be <tt>null</tt>.
     */
    public final List<AnnotationNode>[] visibleParameterAnnotations;

    /**
     * The number of method parameters than can have runtime invisible annotations. This number
must
     * be less or equal than the number of parameter types in the method descriptor (the default
value
     * 0 indicates that all the parameters described in the method descriptor can have annotations).
     * It can be strictly less when a method has synthetic parameters and when these parameters
are
     * ignored when computing parameter indices for the purpose of parameter annotations (see
     * https://docs.oracle.com/javase/specs/jvms/se9/html/jvms-4.html#jvms-4.7.18).
     */
    public final int invisibleAnnotableParameterCount;

    /**
     * The runtime invisible parameter annotations of this method. These lists are lists of {@link
     * AnnotationNode} objects. May be <tt>null</tt>.
     */
    public final List<AnnotationNode>[] invisibleParameterAnnotations;

    /**
     * The instructions of this method.
     */
    public final InsnList instructions;

    /**
     * The try catch blocks of this method.
```

```
 */
public final List<TryCatchBlockNode> tryCatchBlocks;

/**
 * The maximum stack size of this method.
 */
public final int maxStack;

/**
 * The maximum number of local variables of this method.
 */
public final int maxLocals;

/**
 * The local variables of this method. May be <tt>null</tt>
 */
//public final List<LocalVariableNode> localVariables;
public final List<LocalVariableCode> localVariables;

/**
 * The visible local variable annotations of this method. May be <tt>null</tt>
 */
public final List<LocalVariableAnnotationNode> visibleLocalVariableAnnotations;

/**
 * The invisible local variable annotations of this method. May be <tt>null</tt>
 */
public final List<LocalVariableAnnotationNode> invisibleLocalVariableAnnotations;

/**
 * Whether the accept method has been called on this object.
 */
//private final boolean visited;

public final ClassCode classCode;

public final String className;

public final String nameDesc;

public final String fullName;
```

```
public final boolean isPublic;

public final boolean isStatic;

public final boolean isAbstract;

public final boolean isNative;

public final boolean isClinit;

public final boolean isConstructor;

public final VariableType returnVariableType;

public final List<VariableType> argsVariableType;

//contract

public final String returnArg;

public final List<ProgramMethodArg> args;

public final String normalDesc;

public MethodCode(ClassCode classCode, MethodNode methodNode) {
    access = methodNode.access;
    name = methodNode.name;
    desc = methodNode.desc;
    signature = methodNode.signature;
    exceptions = ListUtils.emptyIfNull(methodNode.exceptions);
    parameters = ListUtils.emptyIfNull(methodNode.parameters);
    visibleAnnotations = ListUtils.emptyIfNull(methodNode.visibleAnnotations);
    invisibleAnnotations = ListUtils.emptyIfNull(methodNode.invisibleAnnotations);
    visibleTypeAnnotations = ListUtils.emptyIfNull(methodNode.visibleTypeAnnotations);
    invisibleTypeAnnotations = ListUtils.emptyIfNull(methodNode.invisibleTypeAnnotations);
    attrs = ListUtils.emptyIfNull(methodNode.attrs);
    annotationDefault = methodNode.annotationDefault;
    visibleAnnotableParameterCount = methodNode.visibleAnnotableParameterCount;
    visibleParameterAnnotations = methodNode.visibleParameterAnnotations;
    invisibleAnnotableParameterCount = methodNode.invisibleAnnotableParameterCount;
    invisibleParameterAnnotations = methodNode.invisibleParameterAnnotations;
    instructions = methodNode.instructions;
```

```
        tryCatchBlocks = ListUtils.emptyIfNull(methodNode.tryCatchBlocks);
        maxStack = methodNode.maxStack;
        maxLocals = methodNode.maxLocals;
        //localVariables = ListUtils.emptyIfNull(methodNode.localVariables);
        visibleLocalVariableAnnotations =
ListUtils.emptyIfNull(methodNode.visibleLocalVariableAnnotations);
        invisibleLocalVariableAnnotations =
ListUtils.emptyIfNull(methodNode.invisibleLocalVariableAnnotations);
        //
        this.classCode = classCode;
        className = classCode.name;
        nameDesc = name + desc;
        fullName = className + "." + nameDesc;
        isPublic = (access & Opcodes.ACC_PUBLIC) != 0;
        isStatic = (access & Opcodes.ACC_STATIC) != 0;
        isAbstract = (access & Opcodes.ACC_ABSTRACT) != 0;
        isNative = (access & Opcodes.ACC_NATIVE) != 0;
        isClinit = Constants.CLINIT_NAME.equals(name);
        isConstructor = Constants.CONSTRUCTOR_NAME.equals(name);
        //
        final List<VariableType> variableTypes = VariableType.parseAll(desc);
        final int last = variableTypes.size() - 1;
        returnVariableType = variableTypes.get(last);
        argsVariableType = variableTypes.subList(0, last);

        final List<LocalVariableNode> localVariableNodes =
ListUtils.emptyIfNull(methodNode.localVariables);
        localVariables = new ArrayList<>(arrayListInitialCapacity(localVariableNodes.size()));
        for (LocalVariableNode localVariableNode : localVariableNodes) {
            localVariables.add(new LocalVariableCode(localVariableNode));
        }

        //contract
        returnArg = ProgramDescriptors.getNormalDesc(returnVariableType);
        args = new ArrayList<>(arrayListInitialCapacity(argsVariableType.size()));
        final List<String> stringArgs = new
ArrayList<>(arrayListInitialCapacity(argsVariableType.size()));
        int index = 0;
        if (!isStatic) {
            index += 1;
        }
        for (int i = 0; i < argsVariableType.size(); i++) {
```

```
            final VariableType variableType = argsVariableType.get(i);
            if (i > 0) {
                final VariableType previousVariableType = argsVariableType.get(i - 1);
                if (previousVariableType.isLong() || previousVariableType.isDouble()) {
                    index += 1;
                }
            }
            String name = "var" + (i + 1);

            LocalVariableCode localVariableCode = getLocalVariableCode(index);
            index++;
            if (localVariableCode != null) {
                name = localVariableCode.name;
                if (isConstructor && classCode.isSyntheticField(name)) {
                    continue;
                }
//                if (!variableType.equals(localVariableCode.variableType)) {
//                    System.out.println();
//                }
            }

            final String normalDesc = ProgramDescriptors.getNormalDesc(variableType);
            final String stringArg = normalDesc + " " + name;
            stringArgs.add(stringArg);
            final ProgramMethodArg arg = new ProgramMethodArg(normalDesc, name,
hasRequiredAnnotation(i));
            args.add(arg);
        }

        StringBuilder sb = new StringBuilder();
        sb.append("(");
        sb.append(Joiner.on(", ").join(stringArgs));
        sb.append(") return ");
        sb.append(returnArg);
        normalDesc = sb.toString();

//        String desc = ProgramDescriptors.parseDesc(normalDesc);
//        if (!desc.equals(desc)) {
//            System.out.println();
//        }
    }
```

```
    public boolean hasViewAnnotation() {
        return hasAnnotation(VIEW_ANNOTATION_DESC);
    }

    public boolean hasPayableAnnotation() {
        return hasAnnotation(PAYABLE_ANNOTATION_DESC);
    }

    public boolean hasAnnotation(String annotation) {
        return visibleAnnotations.stream()
                .anyMatch(annotationNode -> annotation.equals(annotationNode.desc));
    }

    private boolean hasRequiredAnnotation(int i) {
        if (!(visibleParameterAnnotations != null && visibleParameterAnnotations.length > 0 &&
visibleParameterAnnotations.length > i)) {
            return false;
        }
        List<AnnotationNode> list = visibleParameterAnnotations[i];
        if (list == null) {
            return false;
        }
        return list.stream().anyMatch(annotationNode ->
REQUIRED_ANNOTATION_DESC.equals(annotationNode.desc));
    }

    public LocalVariableCode getLocalVariableCode(int index) {
        return localVariables.stream().filter(localVariableCode -> localVariableCode.index == index)
                .findFirst().orElse(null);
    }

    public boolean isClass(String className) {
        return this.className.equals(className);
    }

    public boolean isMethod(String name, String desc) {
        return this.name.equals(name) && this.desc.equals(desc);
    }

    public boolean isMethod(String className, String name, String desc) {
        return this.className.equals(className) && this.name.equals(name) &&
this.desc.equals(desc);
```

```
    }

}

100:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\code\VariableType.java
 */
package io.nuls.contract.vm.code;

import com.google.common.cache.CacheBuilder;
import com.google.common.cache.CacheLoader;
import com.google.common.cache.LoadingCache;
import com.google.common.collect.BiMap;
import com.google.common.collect.HashBiMap;
import org.apache.commons.lang3.ArrayUtils;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;

import static io.nuls.contract.vm.util.Utils.arrayListInitialCapacity;

public class VariableType {

    private static final LoadingCache<String, VariableType> CACHE;
    private static final LoadingCache<String, List<VariableType>> CACHE_LIST;

    static {
        CACHE = CacheBuilder.newBuilder()
                .initialCapacity(1024)
                .maximumSize(102400)
                .expireAfterAccess(10, TimeUnit.MINUTES)
                .build(new CacheLoader<String, VariableType>() {
                    @Override
                    public VariableType load(String desc) {
                        return new VariableType(desc);
                    }
                });
        CACHE_LIST = CacheBuilder.newBuilder()
                .initialCapacity(1024)
                .maximumSize(10240)
```

```
            .expireAfterAccess(10, TimeUnit.MINUTES)
            .build(new CacheLoader<String, List<VariableType>>() {
                @Override
                public List<VariableType> load(String desc) {
                    return parseList(desc);
                }
            });
    }

    public static final VariableType INT_TYPE = valueOf("I");
    public static final VariableType LONG_TYPE = valueOf("J");
    public static final VariableType FLOAT_TYPE = valueOf("F");
    public static final VariableType DOUBLE_TYPE = valueOf("D");
    public static final VariableType BOOLEAN_TYPE = valueOf("Z");
    public static final VariableType BYTE_TYPE = valueOf("B");
    public static final VariableType CHAR_TYPE = valueOf("C");
    public static final VariableType SHORT_TYPE = valueOf("S");
    public static final VariableType INT_WRAPPER_TYPE = valueOf("Ljava/lang/Integer;");
    public static final VariableType LONG_WRAPPER_TYPE = valueOf("Ljava/lang/Long;");
    public static final VariableType FLOAT_WRAPPER_TYPE = valueOf("Ljava/lang/Float;");
    public static final VariableType DOUBLE_WRAPPER_TYPE = valueOf("Ljava/lang/Double;");
    public static final VariableType BOOLEAN_WRAPPER_TYPE = valueOf("Ljava/lang/Boolean;");
    public static final VariableType BYTE_WRAPPER_TYPE = valueOf("Ljava/lang/Byte;");
    public static final VariableType CHAR_WRAPPER_TYPE = valueOf("Ljava/lang/Character;");
    public static final VariableType SHORT_WRAPPER_TYPE = valueOf("Ljava/lang/Short;");
    public static final VariableType OBJECT_TYPE = valueOf("Ljava/lang/Object;");
    public static final VariableType STRING_TYPE = valueOf("Ljava/lang/String;");
    public static final VariableType RUNTIME_EXCEPTION_TYPE =
valueOf("Ljava/lang/RuntimeException;");
    public static final VariableType NUMBER_FORMAT_EXCEPTION_TYPE =
valueOf("Ljava/lang/NumberFormatException;");
    public static final VariableType NULL_POINTER_EXCEPTION_TYPE =
valueOf("Ljava/lang/NullPointerException;");
    public static final VariableType ARRAY_INDEX_OUT_OF_BOUNDS_EXCEPTION_TYPE =
valueOf("Ljava/lang/ArrayIndexOutOfBoundsException;");
    public static final VariableType NEGATIVE_ARRAY_SIZE_EXCEPTION_TYPE =
valueOf("Ljava/lang/NegativeArraySizeException;");
    public static final VariableType CLASS_CAST_EXCEPTION_TYPE =
valueOf("Ljava/lang/ClassCastException;");
    public static final VariableType STACK_OVERFLOW_ERROR_TYPE =
valueOf("Ljava/lang/StackOverflowError;");
    public static final VariableType BIGINTEGER_TYPE = valueOf("Ljava/math/BigInteger;");
```

```
    public static final VariableType STRINGBUILDER_TYPE = valueOf("Ljava/lang/StringBuilder;");
    public static final VariableType ADDRESS_TYPE = valueOf("Lio/nuls/contract/sdk/Address;");
    public static final VariableType BLOCK_HEADER_TYPE =
valueOf("Lio/nuls/contract/sdk/BlockHeader;");
    public static final VariableType INT_ARRAY_TYPE = valueOf("[I");
    public static final VariableType LONG_ARRAY_TYPE = valueOf("[J");
    public static final VariableType FLOAT_ARRAY_TYPE = valueOf("[F");
    public static final VariableType DOUBLE_ARRAY_TYPE = valueOf("[D");
    public static final VariableType BOOLEAN_ARRAY_TYPE = valueOf("[Z");
    public static final VariableType BYTE_ARRAY_TYPE = valueOf("[B");
    public static final VariableType CHAR_ARRAY_TYPE = valueOf("[C");
    public static final VariableType SHORT_ARRAY_TYPE = valueOf("[S");
    public static final VariableType STRING_ARRAY_TYPE = valueOf("[Ljava/lang/String;");
    public static final VariableType STACK_TRACE_ELEMENT_TYPE =
valueOf("Ljava/lang/StackTraceElement;");
    public static final VariableType STACK_TRACE_ELEMENT_ARRAY_TYPE =
valueOf("[Ljava/lang/StackTraceElement;");

    public static final VariableType[] WRAPPER_TYPE = new VariableType[]{
            INT_WRAPPER_TYPE,
            LONG_WRAPPER_TYPE,
            FLOAT_WRAPPER_TYPE,
            DOUBLE_WRAPPER_TYPE,
            BOOLEAN_WRAPPER_TYPE,
            BYTE_WRAPPER_TYPE,
            CHAR_WRAPPER_TYPE,
            SHORT_WRAPPER_TYPE
    };

    public static final BiMap<String, String> DESCRIPTORS;

    static {
        DESCRIPTORS = HashBiMap.create();
        DESCRIPTORS.put("z", "Ljava/lang/Boolean;");
        DESCRIPTORS.put("[z", "[Ljava/lang/Boolean;");
        DESCRIPTORS.put("b", "Ljava/lang/Byte;");
        DESCRIPTORS.put("[b", "[Ljava/lang/Byte;");
        DESCRIPTORS.put("s", "Ljava/lang/Short;");
        DESCRIPTORS.put("[s", "[Ljava/lang/Short;");
        DESCRIPTORS.put("c", "Ljava/lang/Character;");
        DESCRIPTORS.put("[c", "[Ljava/lang/Character;");
        DESCRIPTORS.put("i", "Ljava/lang/Integer;");
```

```
    DESCRIPTORS.put("[i", "[Ljava/lang/Integer;");
    DESCRIPTORS.put("l", "Ljava/lang/Long;");
    DESCRIPTORS.put("[l", "[Ljava/lang/Long;");
    DESCRIPTORS.put("f", "Ljava/lang/Float;");
    DESCRIPTORS.put("[f", "[Ljava/lang/Float;");
    DESCRIPTORS.put("d", "Ljava/lang/Double;");
    DESCRIPTORS.put("[d", "[Ljava/lang/Double;");
    DESCRIPTORS.put("r", "Ljava/lang/String;");
    DESCRIPTORS.put("[r", "[Ljava/lang/String;");
    DESCRIPTORS.put("e", "Ljava/math/BigInteger;");
    DESCRIPTORS.put("[e", "[Ljava/math/BigInteger;");
    DESCRIPTORS.put("a", "Lio/nuls/contract/sdk/Address;");
    DESCRIPTORS.put("[a", "[Lio/nuls/contract/sdk/Address;");
    DESCRIPTORS.put("m", "Ljava/util/HashMap;");
    DESCRIPTORS.put("n", "Ljava/util/HashMap$Node;");
    DESCRIPTORS.put("[n", "[Ljava/util/HashMap$Node;");
    DESCRIPTORS.put("g", "Ljava/util/ArrayList;");
    DESCRIPTORS.put("o", "Ljava/lang/Object;");
    DESCRIPTORS.put("[o", "[Ljava/lang/Object;");
}

private String desc;

private String type;

private VariableType componentType;

private boolean primitiveType;

private boolean primitive;

private boolean array;

private int dimensions;

private Object defaultValue;

private VariableType(String desc) {
    if (Descriptors.DESCRIPTORS.containsKey(desc)) {
        this.desc = Descriptors.DESCRIPTORS.get(desc);
    } else {
        this.desc = desc;
```

```
    }
    this.type = this.desc;
    if (this.type.contains("[")) {
        this.array = true;
        this.dimensions = this.type.lastIndexOf("[") + 1;
        this.type = this.desc.replace("[", "");
        this.componentType = valueOf(this.desc.replaceFirst("\\[", ""));
    }
    if (Descriptors.DESCRIPTORS.inverse().containsKey(this.type)) {
        this.type = Descriptors.DESCRIPTORS.inverse().get(this.type);
        this.primitiveType = isNotVoid();
    } else if (this.type.startsWith("L") && this.type.endsWith(";")) {
        this.type = this.type.substring(1, this.type.length() - 1);
    } else {
        this.desc = "L" + this.type + ";";
    }
    if (!this.array && this.primitiveType) {
        this.primitive = true;
        this.defaultValue = this.defaultValue();
    }
}

public static VariableType valueOf(String desc) {
    try {
        VariableType variableType = CACHE.get(desc);
        if (!variableType.getDesc().equals(desc)) {
            variableType = CACHE.get(variableType.getDesc());
            CACHE.put(desc, variableType);
        }
        return variableType;
    } catch (ExecutionException e) {
        throw new RuntimeException(e);
    }
}

public static List<VariableType> parseArgs(String desc) {
    List<VariableType> args = parseAll(desc);
    return args.subList(0, args.size() - 1);
}

public static List<VariableType> parseAll(String desc) {
    try {
```

```
            return CACHE_LIST.get(desc);
        } catch (ExecutionException e) {
            throw new RuntimeException(e);
        }
    }

    private static List<VariableType> parseList(String desc) {
        List<String> list = Descriptors.parse(desc, true);
        List<VariableType> args = new ArrayList<>(arrayListInitialCapacity(list.size()));
        for (String type : list) {
            args.add(valueOf(type));
        }
        return args;
    }

    public boolean isVoid() {
        return Descriptors.VOID.equals(this.type);
    }

    public boolean isNotVoid() {
        return !isVoid();
    }

    public boolean isByte() {
        return primitive && Descriptors.BYTE.equals(this.type);
    }

    public boolean isChar() {
        return primitive && Descriptors.CHAR.equals(this.type);
    }

    public boolean isDouble() {
        return primitive && Descriptors.DOUBLE.equals(this.type);
    }

    public boolean isFloat() {
        return primitive && Descriptors.FLOAT.equals(this.type);
    }

    public boolean isInt() {
        return primitive && Descriptors.INT.equals(this.type);
    }
```

```
public boolean isLong() {
    return primitive && Descriptors.LONG.equals(this.type);
}

public boolean isShort() {
    return primitive && Descriptors.SHORT.equals(this.type);
}

public boolean isBoolean() {
    return primitive && Descriptors.BOOLEAN.equals(this.type);
}

public Object defaultValue() {
    Object defaultValue = null;
    switch (this.type) {
        case Descriptors.INT:
            defaultValue = 0;
            break;
        case Descriptors.LONG:
            defaultValue = 0L;
            break;
        case Descriptors.FLOAT:
            defaultValue = 0.0F;
            break;
        case Descriptors.DOUBLE:
            defaultValue = 0.0D;
            break;
        case Descriptors.BOOLEAN:
            defaultValue = false;
            break;
        case Descriptors.BYTE:
            defaultValue = (byte) 0;
            break;
        case Descriptors.CHAR:
            defaultValue = '\u0000';
            break;
        case Descriptors.SHORT:
            defaultValue = (short) 0;
            break;
        default:
            break;
```

```
    }
    return defaultValue;
}

public Class getPrimitiveTypeClass() {
    Class clazz = null;
    if (!this.primitiveType) {
        return clazz;
    }
    switch (this.type) {
        case Descriptors.INT:
            clazz = Integer.TYPE;
            break;
        case Descriptors.LONG:
            clazz = Long.TYPE;
            break;
        case Descriptors.FLOAT:
            clazz = Float.TYPE;
            break;
        case Descriptors.DOUBLE:
            clazz = Double.TYPE;
            break;
        case Descriptors.BOOLEAN:
            clazz = Boolean.TYPE;
            break;
        case Descriptors.BYTE:
            clazz = Byte.TYPE;
            break;
        case Descriptors.CHAR:
            clazz = Character.TYPE;
            break;
        case Descriptors.SHORT:
            clazz = Short.TYPE;
            break;
        default:
            break;
    }
    return clazz;
}

public Object getPrimitiveValue(Object value) {
    if (this.primitive) {
```

```
    if (value == null || "".equals(value.toString())) {
        value = defaultValue();
    } else {
        String s = value.toString();
        switch (this.type) {
            case Descriptors.INT:
                value = Integer.valueOf(s).intValue();
                break;
            case Descriptors.LONG:
                value = Long.valueOf(s).longValue();
                break;
            case Descriptors.FLOAT:
                value = Float.valueOf(s).floatValue();
                break;
            case Descriptors.DOUBLE:
                value = Double.valueOf(s).doubleValue();
                break;
            case Descriptors.BOOLEAN:
                if ("true".equalsIgnoreCase(s) || "1".equals(s)) {
                    value = true;
                } else {
                    value = false;
                }
                break;
            case Descriptors.BYTE:
                value = Byte.valueOf(s).byteValue();
                break;
            case Descriptors.CHAR:
                if (value instanceof Integer) {
                    value = (char) ((Integer) value).intValue();
                } else {
                    value = s.charAt(0);
                }
                break;
            case Descriptors.SHORT:
                value = Short.valueOf(s).shortValue();
                break;
            default:
                break;
        }
    }
}
```

```
    return value;
}

public String getDesc() {
    return desc;
}

public String getType() {
    return type;
}

public VariableType getComponentType() {
    return componentType;
}

public boolean isPrimitiveType() {
    return primitiveType;
}

public boolean isWrapperType() {
    return ArrayUtils.contains(WRAPPER_TYPE, this);
}

public boolean isStringType() {
    return STRING_TYPE.equals(this);
}

public boolean isPrimitive() {
    return primitive;
}

public boolean isArray() {
    return array;
}

public int getDimensions() {
    return dimensions;
}

public Object getDefaultValue() {
    return defaultValue;
}
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        VariableType that = (VariableType) o;

        if (primitiveType != that.primitiveType) {
            return false;
        }
        if (primitive != that.primitive) {
            return false;
        }
        if (array != that.array) {
            return false;
        }
        if (dimensions != that.dimensions) {
            return false;
        }
        if (desc != null ? !desc.equals(that.desc) : that.desc != null) {
            return false;
        }
        if (type != null ? !type.equals(that.type) : that.type != null) {
            return false;
        }
        if (componentType != null ? !componentType.equals(that.componentType) :
that.componentType != null) {
            return false;
        }
        return defaultValue != null ? defaultValue.equals(that.defaultValue) : that.defaultValue ==
null;
    }

    @Override
    public int hashCode() {
        int result = desc != null ? desc.hashCode() : 0;
        result = 31 * result + (type != null ? type.hashCode() : 0);
```

```
        result = 31 * result + (componentType != null ? componentType.hashCode() : 0);
        result = 31 * result + (primitiveType ? 1 : 0);
        result = 31 * result + (primitive ? 1 : 0);
        result = 31 * result + (array ? 1 : 0);
        result = 31 * result + dimensions;
        result = 31 * result + (defaultValue != null ? defaultValue.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "VariableType{" +
                "desc=" + desc +
                ", type=" + type +
                ", componentType=" + componentType +
                ", primitiveType=" + primitiveType +
                ", primitive=" + primitive +
                ", array=" + array +
                ", dimensions=" + dimensions +
                ", defaultValue=" + defaultValue +
                '}';
    }

}

101:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\exception\ErrorException.java
 */
package io.nuls.contract.vm.exception;

public class ErrorException extends RuntimeException {

    private long gasUsed;

    private String stackTraceMessage;

    public ErrorException(String message, long gasUsed, String stackTraceMessage) {
        super(message);
        this.gasUsed = gasUsed;
        this.stackTraceMessage = stackTraceMessage;
    }
```

```
    public long getGasUsed() {
        return gasUsed;
    }

    public String getStackTraceMessage() {
        return stackTraceMessage;
    }

}

102:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\exception\RevertException.java
 */
package io.nuls.contract.vm.exception;

public class RevertException extends RuntimeException {

    private String stackTraceMessage;

    public RevertException(String message, String stackTraceMessage) {
        super(message);
        this.stackTraceMessage = stackTraceMessage;
    }

    public String getStackTraceMessage() {
        return stackTraceMessage;
    }

}

103:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\Frame.java
 */
package io.nuls.contract.vm;

import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.instructions.references.Athrow;
import org.objectweb.asm.tree.*;

public class Frame {
```

```
public final VM vm;

public final Heap heap;

public final MethodArea methodArea;

public final MethodCode methodCode;

public final int maxStack;

public final int maxLocals;

public final OperandStack operandStack;

public final LocalVariables localVariables;

public final Result result;

private AbstractInsnNode currentInsnNode;

private OpCode currentOpCode;

public boolean addGas = true;

public Frame(VM vm, MethodCode methodCode, Object[] args) {
    this.vm = vm;
    this.heap = vm.heap;
    this.methodArea = vm.methodArea;
    this.methodCode = methodCode;
    this.maxStack = this.methodCode.maxStack;
    this.maxLocals = this.methodCode.maxLocals;
    this.operandStack = new OperandStack(this.maxStack);
    this.localVariables = new LocalVariables(this.maxLocals, args);
    this.result = new Result(this.methodCode.returnVariableType);
    this.currentInsnNode = this.methodCode.instructions.getFirst();
}

public void step() {
    if (this.currentInsnNode != null) {
        this.currentInsnNode = this.currentInsnNode.getNext();
    }
}
```

```
public void jump() {
    this.currentInsnNode = jumpInsnNode().label;
}

public void jump(LabelNode label) {
    this.currentInsnNode = label;
}

public OpCode currentOpCode() {
    if (this.currentInsnNode != null) {
        this.currentOpCode = OpCode.valueOf(this.currentInsnNode.getOpcode());
    } else {
        this.currentOpCode = null;
    }
    return this.currentOpCode;
}

public int getLine(LabelNode labelNode) {
    AbstractInsnNode abstractInsnNode = labelNode;
    while (!(abstractInsnNode instanceof LineNumberNode)) {
        abstractInsnNode = abstractInsnNode.getNext();
    }
    return ((LineNumberNode) abstractInsnNode).line;
}

public int getLine() {
    AbstractInsnNode abstractInsnNode = this.currentInsnNode;
    while (!(abstractInsnNode instanceof LineNumberNode)) {
        abstractInsnNode = abstractInsnNode.getPrevious();
    }
    return ((LineNumberNode) abstractInsnNode).line;
}

public boolean checkArray(ObjectRef arrayRef, int index) {
    if (arrayRef == null) {
        throwNullPointerException();
        return false;
    }
    int length = arrayRef.getDimensions()[0];
    if (index < 0 || index >= length) {
        throwArrayIndexOutOfBoundsException(index);
```

```
            return false;
        }
        return true;
    }

    private void throwException(ObjectRef objectRef) {
        this.operandStack.pushRef(objectRef);
        Athrow.athrow(this);
    }

    public void throwRuntimeException(String message) {
        ObjectRef objectRef =
this.heap.runNewObject(VariableType.RUNTIME_EXCEPTION_TYPE, message);
        throwException(objectRef);
    }

    public void throwNumberFormatException(String message) {
        ObjectRef objectRef =
this.heap.runNewObject(VariableType.NUMBER_FORMAT_EXCEPTION_TYPE, message);
        throwException(objectRef);
    }

    public void throwNullPointerException() {
        ObjectRef objectRef =
this.heap.runNewObject(VariableType.NULL_POINTER_EXCEPTION_TYPE);
        throwException(objectRef);
    }

    public void throwArrayIndexOutOfBoundsException(int index) {
        ObjectRef objectRef =
this.heap.runNewObject(VariableType.ARRAY_INDEX_OUT_OF_BOUNDS_EXCEPTION_TYPE)
;
        throwException(objectRef);
    }

    public void throwNegativeArraySizeException() {
        ObjectRef objectRef =
this.heap.runNewObject(VariableType.NEGATIVE_ARRAY_SIZE_EXCEPTION_TYPE);
        throwException(objectRef);
    }

    public void throwClassCastException() {
```

```
        ObjectRef objectRef =
this.heap.runNewObject(VariableType.CLASS_CAST_EXCEPTION_TYPE);
        throwException(objectRef);
    }

    private void throwError(ObjectRef objectRef) {
        this.vm.getResult().error(objectRef);
    }

    public void throwStackOverflowError() {
        ObjectRef objectRef =
this.heap.runNewObject(VariableType.STACK_OVERFLOW_ERROR_TYPE);
        throwError(objectRef);
    }

    public void nonsupportOpCode() {
        int line = getLine();
        throw new RuntimeException(String.format("nonsupport opcodeclass(%s), line(%d)",
methodCode.className, line));
    }

    public void nonsupportMethod(MethodCode methodCode) {
        throw new RuntimeException("nonsupport method: " + methodCode.fullName);
    }

    public InsnNode insnNode() {
        return (InsnNode) this.currentInsnNode;
    }

    public IntInsnNode intInsnNode() {
        return (IntInsnNode) this.currentInsnNode;
    }

    public VarInsnNode varInsnNode() {
        return (VarInsnNode) this.currentInsnNode;
    }

    public TypeInsnNode typeInsnNode() {
        return (TypeInsnNode) this.currentInsnNode;
    }

    public FieldInsnNode fieldInsnNode() {
```

```
    return (FieldInsnNode) this.currentInsnNode;
}

public MethodInsnNode methodInsnNode() {
    return (MethodInsnNode) this.currentInsnNode;
}

public InvokeDynamicInsnNode invokeDynamicInsnNode() {
    return (InvokeDynamicInsnNode) this.currentInsnNode;
}

public JumpInsnNode jumpInsnNode() {
    return (JumpInsnNode) this.currentInsnNode;
}

public LabelNode labelNode() {
    return (LabelNode) this.currentInsnNode;
}

public LdcInsnNode ldcInsnNode() {
    return (LdcInsnNode) this.currentInsnNode;
}

public IincInsnNode iincInsnNode() {
    return (IincInsnNode) this.currentInsnNode;
}

public TableSwitchInsnNode tableSwitchInsnNode() {
    return (TableSwitchInsnNode) this.currentInsnNode;
}

public LookupSwitchInsnNode lookupSwitchInsnNode() {
    return (LookupSwitchInsnNode) this.currentInsnNode;
}

public MultiANewArrayInsnNode multiANewArrayInsnNode() {
    return (MultiANewArrayInsnNode) this.currentInsnNode;
}

public FrameNode frameNode() {
    return (FrameNode) this.currentInsnNode;
}
```

```
    public LineNumberNode lineNumberNode() {
        return (LineNumberNode) this.currentInsnNode;
    }

    public AbstractInsnNode getCurrentInsnNode() {
        return currentInsnNode;
    }

    public void setAddGas(boolean addGas) {
        this.addGas = addGas;
    }

    public OpCode getCurrentOpCode() {
        return currentOpCode;
    }

}

104:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\GasCost.java
 */
package io.nuls.contract.vm;

public class GasCost {

    public static final int COMPARISON = 1;//
    public static final int CONSTANT = 1;//
    public static final int LDC = 1;// * LDC
    public static final int CONTROL = 5;//
    public static final int TABLESWITCH = 2;//switch * TABLESWITCH
    public static final int LOOKUPSWITCH = 2;//switch * LOOKUPSWITCH
    public static final int CONVERSION = 1;//
    public static final int EXTENDED = 1;//null
    public static final int MULTIANEWARRAY = 1;// * MULTIANEWARRAY
    public static final int LOAD = 1;//
    public static final int ARRAYLOAD = 5;//
    public static final int MATH = 1;//
    public static final int REFERENCE = 10;//
    public static final int NEWARRAY = 1;// * NEWARRAY
    public static final int STACK = 2;//
    public static final int STORE = 1;//
```

```
    public static final int ARRAYSTORE = 5;//
    public static final int TRANSFER = 1000;//

}

105:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\Heap.java
 */
package io.nuls.contract.vm;

import com.google.common.collect.BiMap;
import com.google.common.collect.HashBiMap;
import io.nuls.contract.vm.code.ClassCode;
import io.nuls.contract.vm.code.FieldCode;
import io.nuls.contract.vm.code.MethodCode;
import io.nuls.contract.vm.code.VariableType;
import io.nuls.contract.vm.natives.io.nuls.contract.sdk.NativeAddress;
import io.nuls.contract.vm.util.CloneUtils;
import io.nuls.contract.vm.util.Constants;
import io.nuls.contract.vm.util.JsonUtils;
import org.apache.commons.lang3.StringUtils;
import org.ethereum.core.Repository;
import org.ethereum.vm.DataWord;

import java.lang.reflect.Array;
import java.math.BigInteger;
import java.util.*;

public class Heap {

    public static final Map<ObjectRef, Map<String, Object>> INIT_OBJECTS = new
HashMap<>(1024);

    public static final Map<String, Object> INIT_ARRAYS = new HashMap<>(1024);

    private VM vm;

    public final Map<ObjectRef, Map<String, Object>> objects = new HashMap<>(1024);

    public final Map<String, Object> arrays = new HashMap<>(1024);

    private final Set<ObjectRef> changes = new HashSet<>(1024);
```

```
private final BiMap<String, String> classNames = HashBiMap.create(1024);

private ObjectRef contract;

private byte[] address;

private Repository repository;

private BigInteger objectRefCount;

private static final DataWord OBJECT_REF_COUNT = new DataWord("objectRefCount");

public Heap(BigInteger objectRefCount) {
    this.objectRefCount = new BigInteger(objectRefCount.toString());
}

public void setVm(VM vm) {
    this.vm = vm;
}

public void loadClassCodes(Map<String, ClassCode> classCodes) {
    if (classCodes != null) {
        int i = 0;
        for (ClassCode classCode : classCodes.values()) {
            this.classNames.put(String.valueOf(i++), classCode.variableType.getDesc());
        }
        this.classNames.putAll(VariableType.DESCRIPTORS);
    }
}

public ObjectRef newObjectRef(String ref, String desc, int... dimensions) {
    if (StringUtils.isEmpty(ref)) {
        objectRefCount = objectRefCount.add(BigInteger.ONE);
        ref = objectRefCount.toString();
    }
    ObjectRef objectRef = new ObjectRef(ref, desc, dimensions);
    objects.put(objectRef, new LinkedHashMap<>());
    change(objectRef);
    return objectRef;
}
```

```
public ObjectRef newObjectRef(String desc, int... dimensions) {
    return newObjectRef(null, desc, dimensions);
}

public ObjectRef newObject(String ref, ClassCode classCode) {
    ObjectRef objectRef = newObjectRef(ref, classCode.variableType.getDesc());
    initFields(classCode, objectRef);
    return objectRef;
}

public ObjectRef newObject(ClassCode classCode) {
    return newObject(null, classCode);
}

public ObjectRef newObject(String className) {
    ClassCode classCode = this.vm.methodArea.loadClass(className);
    return newObject(classCode);
}

public ObjectRef newObject(VariableType variableType) {
    ClassCode classCode = this.vm.methodArea.loadClass(variableType.getType());
    return newObject(classCode);
}

public Map<String, Object> getFieldsInit(ObjectRef objectRef) {
    Map<String, Object> fields = objects.get(objectRef);
    if (fields == null) {
        fields = INIT_OBJECTS.get(objectRef);
    }
    return fields;
}

public Map<String, Object> getFields(ObjectRef objectRef) {
    Map<String, Object> fields = getFieldsInit(objectRef);
    if (fields == null) {
        fields = getFieldsFromState(objectRef);
        if (fields != null) {
            objects.put(objectRef, fields);
        }
    }
    return fields;
}
```

```
    public void putFields(ObjectRef objectRef, Map<String, Object> fields) {
        objects.put(objectRef, fields);
        change(objectRef);
    }

    public Map<String, Object> putFields(ObjectRef objectRef) {
        Map<String, Object> fields = objects.get(objectRef);
        if (fields == null) {
            fields = INIT_OBJECTS.get(objectRef);
            if (fields != null) {
                fields = CloneUtils.clone(fields);
                objects.put(objectRef, fields);
            } else {
                fields = getFieldsFromState(objectRef);
                if (fields != null) {
                    objects.put(objectRef, fields);
                }
            }
        }
        return fields;
    }

    public Map<String, Object> getFieldsFromState(ObjectRef objectRef) {
        if (this.repository == null) {
            return null;
        }
        String key = JsonUtils.encode(objectRef, classNames);
        DataWord dataWord = this.repository.getStorageValue(this.address, new DataWord(key));
        if (dataWord == null) {
            return null;
        }
        byte[] value = dataWord.getNoLeadZeroesData();
        Map<String, Object> map = (Map<String, Object>) JsonUtils.decode(new String(value),
classNames);
        return map;
    }

    public Object getField(ObjectRef objectRef, String fieldName) {
        return getFields(objectRef).get(fieldName);
    }
```

```
public void putField(ObjectRef objectRef, String fieldName, Object value) {
    putFields(objectRef).put(fieldName, value);
    change(objectRef);
}

public Object getStatic(String className, String fieldName) {
    ObjectRef objectRef = getStaticObjectRef(className);
    return getField(objectRef, fieldName);
}

public void putStatic(String className, String fieldName, Object value) {
    ObjectRef objectRef = getStaticObjectRef(className);
    putField(objectRef, fieldName, value);
}

private ObjectRef getStaticObjectRef(String className) {
    ClassCode classCode = this.vm.methodArea.loadClass(className);
    ObjectRef objectRef = new ObjectRef(classCode.name, classCode.variableType.getDesc());
    Map<String, Object> map = getFieldsInit(objectRef);
    if (map == null) {
        objectRef = newObjectRef(classCode.name, classCode.variableType.getDesc());
    }
    return objectRef;
}

public ObjectRef newArray(VariableType type, int... dimensions) {
    ObjectRef objectRef = newObjectRef(type.getDesc(), dimensions);
    return objectRef;
}

public Object getArrayInit(ObjectRef arrayRef, Integer key) {
    if (key == 0) {
        return getField(arrayRef, key.toString());
    }
    String arrayKey = arrayRef.getRef() + "_" + key;
    Object object = arrays.get(arrayKey);
    if (object == null) {
        object = INIT_ARRAYS.get(arrayKey);
    }
    return object;
}
```

```
    public Object putArrayInit(ObjectRef arrayRef, Integer key) {
        if (key == 0) {
            return putFields(arrayRef).get(key.toString());
        }
        String arrayKey = arrayRef.getRef() + "_" + key;
        Object object = arrays.get(arrayKey);
        if (object == null) {
            object = INIT_ARRAYS.get(arrayKey);
            if (object != null) {
                object = CloneUtils.cloneObject(object);
                arrays.put(arrayKey, object);
            }
        }
        return object;
    }

    public Object getArrayChunk(ObjectRef arrayRef, int chunkNum, boolean write) {
        getFields(arrayRef);
        String key = Integer.toString(chunkNum);
        String arrayKey = arrayRef.getRef() + "_" + key;
        Object value = null;
        if (write) {
            value = putArrayInit(arrayRef, chunkNum);
        } else {
            value = getArrayInit(arrayRef, chunkNum);
        }
        if (value == null) {
            value = getArrayChunkFromState(arrayRef, arrayKey);
            if (value != null) {
                arrays.put(arrayKey, value);
            }
        }
        if (value == null) {
            int arrayLength = getArrayLength(arrayRef);
            int chunkLength = (chunkNum + 1) * 1024 <= arrayLength ? 1024 : arrayLength % 1024;
            if (arrayRef.getDimensions().length == 1 && arrayRef.getVariableType().isPrimitiveType())
{
                Class componentType = arrayRef.getVariableType().getPrimitiveTypeClass();
                value = Array.newInstance(componentType, chunkLength);
            } else {
                value = new ObjectRef[chunkLength];
            }
```

```
            if (chunkNum == 0) {
                putField(arrayRef, key, value);
            } else {
                this.arrays.put(arrayKey, value);
                putField(arrayRef, key, key);
            }
        }
        value = getArrayInit(arrayRef, chunkNum);
        return value;
    }

    public Object getArrayChunkFromState(ObjectRef arrayRef, String arrayKey) {
        if (this.repository == null) {
            return null;
        }
        DataWord dataWord = this.repository.getStorageValue(this.address, new
DataWord(arrayKey));
        if (dataWord == null) {
            return null;
        }
        byte[] value = dataWord.getNoLeadZeroesData();
        Class clazz = arrayRef.getVariableType().getPrimitiveTypeClass();
        if (!arrayRef.getVariableType().getComponentType().isPrimitive()) {
            clazz = ObjectRef.class;
        }
        Object object = JsonUtils.decodeArray(new String(value), clazz, classNames);
        return object;
    }

    public Object getArray(ObjectRef arrayRef, int index) {
        int chunkNum = index / 1024;
        int chunkIndex = index % 1024;
        Object arrayChunk = getArrayChunk(arrayRef, chunkNum, false);
        Object value = Array.get(arrayChunk, chunkIndex);
        if (value == null && arrayRef.getDimensions().length > 1) {
            int[] dimensions = new int[arrayRef.getDimensions().length - 1];
            System.arraycopy(arrayRef.getDimensions(), 1, dimensions, 0,
arrayRef.getDimensions().length - 1);
            VariableType variableType =
VariableType.valueOf(arrayRef.getVariableType().getDesc().substring(1));
            value = newArray(variableType, dimensions);
            putArray(arrayRef, index, value);
```

```
        }
        return value;
    }

    public void putArray(ObjectRef arrayRef, int index, Object value) {
        int chunkNum = index / 1024;
        int chunkIndex = index % 1024;
        Object arrayChunk = getArrayChunk(arrayRef, chunkNum, true);
        Array.set(arrayChunk, chunkIndex, value);
        change(arrayRef);
    }

    public void arraycopy(Object src, int srcPos, Object dest, int destPos, int length) {
        if (length < 1) {
            return;
        }
        checkArray(src, srcPos);
        checkArray(src, srcPos + length - 1);
        checkArray(dest, destPos);
        checkArray(dest, destPos + length - 1);

        while (length > 0) {
            int srcChunk = srcPos / 1024;
            int srcIndex = srcPos % 1024;
            int destChunk = destPos / 1024;
            int destIndex = destPos % 1024;
            int index = Math.max(srcIndex, destIndex);
            int copyLength = 1024 - index;
            copyLength = Math.min(copyLength, length);
            arrayChunkCopy(src, srcChunk, srcIndex, dest, destChunk, destIndex, copyLength);
            srcPos += copyLength;
            destPos += copyLength;
            length -= copyLength;
        }
    }

    public void arrayChunkCopy(Object src, int srcChunk, int srcPos, Object dest, int destChunk, int
destPos, int length) {
        Object srcArray = src;
        if (src instanceof ObjectRef) {
            srcArray = getArrayChunk((ObjectRef) src, srcChunk, false);
        } else {
```

```
        srcPos = srcChunk * 1024 + srcPos;
    }
    Object destArray = dest;
    if (dest instanceof ObjectRef) {
        ObjectRef destObjectRef = (ObjectRef) dest;
        destArray = getArrayChunk(destObjectRef, destChunk, true);
        change(destObjectRef);
    } else {
        destPos = destChunk * 1024 + destPos;
    }
    System.arraycopy(srcArray, srcPos, destArray, destPos, length);
}

public ObjectRef newArray(char[] chars) {
    if (chars == null) {
        return null;
    }
    ObjectRef objectRef = newArray(VariableType.CHAR_ARRAY_TYPE, chars.length);
    arraycopy(chars, 0, objectRef, 0, chars.length);
    return objectRef;
}

public ObjectRef newArray(byte[] bytes) {
    if (bytes == null) {
        return null;
    }
    ObjectRef objectRef = newArray(VariableType.BYTE_ARRAY_TYPE, bytes.length);
    arraycopy(bytes, 0, objectRef, 0, bytes.length);
    return objectRef;
}

public ObjectRef newArray(Object array, VariableType variableType, int length) {
    if (array == null) {
        return null;
    }
    ObjectRef objectRef = newArray(variableType, length);
    arraycopy(array, 0, objectRef, 0, length);
    return objectRef;
}

public ObjectRef newString(String str) {
    if (str == null) {
```

```
            return null;
        }
        ObjectRef objectRef = newObjectRef(VariableType.STRING_TYPE.getDesc());
        putField(objectRef, Constants.HASH, str.hashCode());
        putField(objectRef, Constants.VALUE, newArray(str.toCharArray()));
        return objectRef;
    }

    public ObjectRef newBigInteger(String value) {
        ObjectRef objectRef = runNewObject(VariableType.BIGINTEGER_TYPE, value);
        return objectRef;
    }

    public ObjectRef newAddress(String value) {
        ObjectRef objectRef = runNewObject(VariableType.ADDRESS_TYPE, value);
        return objectRef;
    }

    public ObjectRef newCharacter(char value) {
        return runNewObjectWithArgs(VariableType.CHAR_WRAPPER_TYPE,
Constants.CHAR_CONSTRUCTOR_DESC, value);
    }

    public ObjectRef runNewObject(VariableType variableType, byte[] bytes) {
        ObjectRef ref = newArray(bytes);
        return runNewObjectWithArgs(variableType, Constants.BYTES_CONSTRUCTOR_DESC,
ref);
    }

    public ObjectRef runNewObject(VariableType variableType) {
        return runNewObjectWithArgs(variableType, Constants.CONSTRUCTOR_DESC);
    }

    public ObjectRef runNewObject(VariableType variableType, String str) {
        ObjectRef strRef = newString(str);
        return runNewObjectWithArgs(variableType, Constants.CONSTRUCTOR_STRING_DESC,
strRef);
    }

    public ObjectRef runNewObjectWithArgs(VariableType variableType, String methodDesc,
Object... args) {
        ClassCode classCode = this.vm.methodArea.loadClass(variableType.getType());
```

```
        ObjectRef objectRef = newObject(classCode);
        MethodCode methodCode =
this.vm.methodArea.loadMethod(objectRef.getVariableType().getType(),
Constants.CONSTRUCTOR_NAME, methodDesc);
        if (methodCode == null) {
            throw new RuntimeException(String.format("can't new %s", variableType.getType()));
        }
        Object[] runArgs = new Object[args.length + 1];
        runArgs[0] = objectRef;
        for (int i = 1; i < runArgs.length; i++) {
            runArgs[i] = args[i - 1];
        }
        this.vm.run(methodCode, runArgs, false);
        return objectRef;
    }

    public ObjectRef getClassRef(String desc) {
        ObjectRef objectRef = new ObjectRef(desc, Constants.CLASS_DESC);
        Object object = getFields(objectRef);
        if (object == null) {
            ClassCode classCode = this.vm.methodArea.loadClass(Constants.CLASS_NAME);
            objectRef = newObject(desc, classCode);
        }
        return objectRef;
    }

    public Object getObject(ObjectRef objectRef) {
        if (objectRef == null) {
            return null;
        } else if (objectRef.isArray() && objectRef.getVariableType().isPrimitiveType() &&
objectRef.getDimensions().length == 1) {
            Class componentType = objectRef.getVariableType().getPrimitiveTypeClass();
            Object array = Array.newInstance(componentType, objectRef.getDimensions());
            int length = getArrayLength(objectRef);
            arraycopy(objectRef, 0, array, 0, length);
            return array;
        } else if (VariableType.STRING_ARRAY_TYPE.equals(objectRef.getVariableType())) {
            int length = getArrayLength(objectRef);
            String[] strings = new String[length];
            for (int i = 0; i < strings.length; i++) {
                ObjectRef ref = (ObjectRef) getArray(objectRef, i);
                String str = runToString(ref);
```

```
                strings[i] = str;
            }
            return strings;
        } else if (VariableType.STRING_TYPE.equals(objectRef.getVariableType())) {
            ObjectRef charsRef = (ObjectRef) getField(objectRef, Constants.VALUE);
            char[] chars = (char[]) getObject(charsRef);
            String str = new String(chars);
            return str;
        } else if (VariableType.BIGINTEGER_TYPE.equals(objectRef.getVariableType())) {
            return toBigInteger(objectRef);
        } else {
            return runToString(objectRef);
        }
    }

    public String runToString(ObjectRef objectRef) {
        if (objectRef == null) {
            return null;
        }
        String type = objectRef.getVariableType().getType();
        if (objectRef.getVariableType().isArray() && objectRef.getVariableType().isPrimitiveType()) {
            type = VariableType.OBJECT_TYPE.getType();
        }
        MethodCode methodCode = this.vm.methodArea.loadMethod(type,
Constants.TO_STRING_METHOD_NAME, Constants.TO_STRING_METHOD_DESC);
        this.vm.run(methodCode, new Object[]{objectRef}, false);
        Object result = this.vm.getResultValue();
        String value = (String) getObject((ObjectRef) result);
        return value;
    }

    public String stackTrace(ObjectRef objectRef) {
        if (objectRef == null) {
            return null;
        }
        StringBuilder s = new StringBuilder();
        s.append(runToString(objectRef));
        s.append("\n");
        ObjectRef stackTraceElementsRef = (ObjectRef) getField(objectRef, "stackTraceElements");
        int size = stackTraceElementsRef.getDimensions()[0];
        for (int i = 0; i < size; i++) {
            ObjectRef stackTraceElementRef = (ObjectRef) getArray(stackTraceElementsRef, i);
```

```
            s.append("\tat " + runToString(stackTraceElementRef));
            s.append("\n");
        }
        return s.toString();
    }

    public BigInteger toBigInteger(ObjectRef objectRef) {
        String value = runToString(objectRef);
        if (value == null) {
            return null;
        }
        return new BigInteger(value);
    }

    public ObjectRef newContract(byte[] address, ClassCode contractCode, Repository repository) {
        ObjectRef objectRef = newObject(NativeAddress.toString(address), contractCode);
        this.contract = objectRef;
        this.address = address;
        this.repository = repository;
        return this.contract;
    }

    public ObjectRef loadContract(byte[] address, ClassCode contractCode, Repository repository)
{
        if (this.contract != null) {
            return this.contract;
        }
        ObjectRef objectRef = new ObjectRef(NativeAddress.toString(address),
contractCode.variableType.getDesc());
        this.contract = objectRef;
        this.address = address;
        this.repository = repository;
        this.objectRefCount = this.repository.getStorageValue(this.address,
OBJECT_REF_COUNT).toBigInteger();
        String className = this.contract.getVariableType().getType();
        ObjectRef staticObjectRef = getStaticObjectRef(className);
        Map<String, Object> fields = getFieldsFromState(staticObjectRef);
        if (fields != null) {
            objects.put(staticObjectRef, fields);
        }
        return this.contract;
    }
```

```
public Map<DataWord, DataWord> contractState() {
    Map<DataWord, DataWord> contractState = new HashMap<>(1024);
    contractState.put(OBJECT_REF_COUNT, new DataWord(this.objectRefCount));
    Set<ObjectRef> stateObjectRefs = new HashSet<>(1024);
    String className = this.contract.getVariableType().getType();
    ObjectRef staticObjectRef = getStaticObjectRef(className);
    stateObjectRefs(stateObjectRefs, staticObjectRef);
    stateObjectRefs(stateObjectRefs, this.contract);
    for (ObjectRef objectRef : stateObjectRefs) {
        if (!this.changes.contains(objectRef)) {
            continue;
        }
        Map<String, Object> fields = getFieldsInit(objectRef);
        if (fields == null) {
            continue;
        }
        String key = JsonUtils.encode(objectRef, classNames);
        String value = JsonUtils.encode(fields, classNames);
        contractState.put(new DataWord(key), new DataWord(value));
        if (objectRef.isArray()) {
            for (String k : fields.keySet()) {
                Integer i = Integer.valueOf(k);
                if (i == 0) {
                    continue;
                }
                String arrayKey = objectRef.getRef() + "_" + k;
                Object object = getArrayInit(objectRef, i);
                if (object != null) {
                    Class clazz = objectRef.getVariableType().getPrimitiveTypeClass();
                    if (!objectRef.getVariableType().getComponentType().isPrimitive()) {
                        clazz = ObjectRef.class;
                    }
                    String arrayValue = JsonUtils.encodeArray(object, clazz, classNames);
                    contractState.put(new DataWord(arrayKey), new DataWord(arrayValue));
                }
            }
        }
    }
    return contractState;
}
```

```
    public void stateObjectRefs(Set<ObjectRef> stateObjectRefs, ObjectRef objectRef) {
        if (!stateObjectRefs.contains(objectRef)) {
            stateObjectRefs.add(objectRef);
            Map<String, Object> fields = getFieldsInit(objectRef);
            if (fields != null) {
                for (Map.Entry<String, Object> entry : fields.entrySet()) {
                    String key = entry.getKey();
                    Object object = entry.getValue();
                    if (object != null) {
                        if (object instanceof ObjectRef) {
                            stateObjectRefs(stateObjectRefs, (ObjectRef) object);
                        }
                        if (objectRef.isArray()) {
                            Object array = getArrayInit(objectRef, Integer.valueOf(key));
                            if (array != null &&
!objectRef.getVariableType().getComponentType().isPrimitive()) {
                                int length = Array.getLength(array);
                                for (int i = 0; i < length; i++) {
                                    Object a = Array.get(array, i);
                                    if (a != null) {
                                        stateObjectRefs(stateObjectRefs, (ObjectRef) a);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    private void change(ObjectRef objectRef) {
        if (objectRef != null && !this.changes.contains(objectRef)) {
            this.changes.add(objectRef);
        }
    }

    private void initFields(ClassCode classCode, ObjectRef objectRef) {
        if (StringUtils.isNotBlank(classCode.superName)) {
            ClassCode superClassCode = this.vm.methodArea.loadClass(classCode.superName);
            initFields(superClassCode, objectRef);
        }
```

```
    for (FieldCode fieldCode : classCode.fields.values()) {
        if (!fieldCode.isStatic) {
            putField(objectRef, fieldCode.name, fieldCode.variableType.getDefaultValue());
        }
    }
}

private void checkArray(Object array, int index) {
    if (array instanceof ObjectRef) {
        checkArray((ObjectRef) array, index);
    } else {
        if (array == null) {
            throw new NullPointerException();
        }
        int length = Array.getLength(array);
        if (index < 0 || index >= length) {
            throw new ArrayIndexOutOfBoundsException(index);
        }
    }
}

private void checkArray(ObjectRef arrayRef, int index) {
    if (arrayRef == null) {
        throw new NullPointerException();
    }
    int length = getArrayLength(arrayRef);
    if (index < 0 || index >= length) {
        throw new ArrayIndexOutOfBoundsException(index);
    }
}

private int getArrayLength(ObjectRef arrayRef) {
    int length = arrayRef.getDimensions()[0];
    return length;
}

public boolean existContract(byte[] address) {
    if (this.repository != null && this.repository.isExist(address)) {
        return true;
    } else {
        return false;
    }
```

```
    }

    public BigInteger getObjectRefCount() {
        return objectRefCount;
    }

}

106:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\comparisons\Dcmp.java
 */
package io.nuls.contract.vm.instructions.comparisons;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Dcmp {

    public static void dcmpl(Frame frame) {
        double value2 = frame.operandStack.popDouble();
        double value1 = frame.operandStack.popDouble();
        int result;
        if (Double.isNaN(value1) || Double.isNaN(value2)) {
            result = -1;
        } else {
            result = Double.compare(value1, value2);
        }
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "compare", value2);
    }

    public static void dcmpg(Frame frame) {
        double value2 = frame.operandStack.popDouble();
        double value1 = frame.operandStack.popDouble();
        int result;
        if (Double.isNaN(value1) || Double.isNaN(value2)) {
            result = 1;
        } else {
            result = Double.compare(value1, value2);
        }
        frame.operandStack.pushInt(result);
```

```
        //Log.result(frame.getCurrentOpCode(), result, value1, "compare", value2);
    }

}

107:F:\git\coin\nuls\nuls-1.1.3\nuls\contract-module\base\contract-
vm\src\main\java\io\nuls\contract\vm\instructions\comparisons\Fcmp.java
 */
package io.nuls.contract.vm.instructions.comparisons;

import io.nuls.contract.vm.Frame;
import io.nuls.contract.vm.util.Log;

public class Fcmp {

    public static void fcmpl(Frame frame) {
        float value2 = frame.operandStack.popFloat();
        float value1 = frame.operandStack.popFloat();
        int result;
        if (Float.isNaN(value1) || Float.isNaN(value2)) {
            result = -1;
        } else {
            result = Float.compare(value1, value2);
        }
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "compare", value2);
    }

    public static void fcmpg(Frame frame) {
        float value2 = frame.operandStack.popFloat();
        float value1 = frame.operandStack.popFloat();
        int result;
        if (Float.isNaN(value1) || Float.isNaN(value2)) {
            result = 1;
        } else {
            result = Float.compare(value1, value2);
        }
        frame.operandStack.pushInt(result);

        //Log.result(frame.getCurrentOpCode(), result, value1, "compare", value2);
```

```
    }

}
```