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

0:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\consensus\src\main\java\io\nuls\consensus\constant\ConsensusConstant.java
 *
 */

```
package io.nuls.consensus.constant;

/**
 * Created by ln on 2018/5/7.
 */
public interface ConsensusConstant {

    /**
     * consensus module id
     */
    short MODULE_ID_CONSENSUS = 7;

    /**
     * consensus transaction types
     */
    int TX_TYPE_REGISTER_AGENT = 4;
    int TX_TYPE_JOIN_CONSENSUS = 5;
    int TX_TYPE_CANCEL_DEPOSIT = 6;
    int TX_TYPE_YELLOW_PUNISH = 7;
    int TX_TYPE_RED_PUNISH = 8;
    int TX_TYPE_STOP_AGENT = 9;


}
```

1:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\consensus\src\main\java\io\nuls\consensus\module\AbstractConsensusModule.java
 *
 */

```
package io.nuls.consensus.module;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.kernel.module.BaseModuleBootstrap;

/**
```

```
 * @author Niels
 */
public abstract class AbstractConsensusModule extends BaseModuleBootstrap {
    public AbstractConsensusModule() {
        super(ConsensusConstant.MODULE_ID_CONSENSUS);
    }
}

2:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-
module\consensus\src\main\java\io\nuls\consensus\service\ConsensusService.java
 *
 */

package io.nuls.consensus.service;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.*;
import io.nuls.network.model.Node;

import java.util.List;

/**
 *
 * @author ln
 */
public interface ConsensusService {

    /**
     * receive a new transaction, add in memory pool after verify success
     * @param tx
     * @return Result
     */
    Result newTx(Transaction<? extends BaseNulsData> tx);

    /**
     * receive block from other peers
     * @param block
     * @return Result
     */
    Result newBlock(Block block);

    /**
```

```
     * receive block from other peers
     * @param block
     * @param node
     * @return Result
     */
    Result newBlock(Block block, Node node);

    /**
     * synchronous block from other peers
     * @param block
     * @return Result
     */
    Result addBlock(Block block);

    /**
     * Roll back the latest block and roll back the status of the chain in the consensus service memory
     *
     *
     * @return Result
     */
    Result rollbackBlock(Block block) throws NulsException;

    /**
     * Get all the transactions in the memory pool
     *
     *
     * @return List<Transaction>
     */
    List<Transaction> getMemoryTxs();

    Transaction getTx(NulsDigestData hash);

}

3:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\block\validator\BifurcationUtil.java
 */

package io.nuls.consensus.poc.block.validator;

import io.nuls.consensus.poc.cache.TxMemoryPool;
```

```
import io.nuls.consensus.poc.config.ConsensusConfig;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.model.Evidence;
import io.nuls.consensus.poc.protocol.constant.PunishReasonEnum;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.storage.service.BifurcationEvidenceStorageService;
import io.nuls.consensus.poc.util.ConsensusTool;
import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.CoinData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.service.BlockService;

import java.io.IOException;
import java.util.*;

/**
 * @author: Niels Wang
 */
public class BifurcationUtil {

    private static final BifurcationUtil INSTANCE = new BifurcationUtil();

    private static String CLASS_NAME = BifurcationUtil.class.getName();

    /**
     * PackingAddress
     *  3
     */
    private Map<String, List<Evidence>> bifurcationEvidenceMap = new HashMap<>();

    private BifurcationEvidenceStorageService bifurcationEvidenceStorageService =
NulsContext.getServiceBean(BifurcationEvidenceStorageService.class);
```

```
    private BifurcationUtil() {
    }

    public void setBifurcationEvidenceMap(Map<String, List<Evidence>> bifurcationEvidenceMap) {
        this.bifurcationEvidenceMap = bifurcationEvidenceMap;
    }

    public static BifurcationUtil getInstance() {
        return INSTANCE;
    }

    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);

    private ConsensusService consensusService =
NulsContext.getServiceBean(ConsensusService.class);

    public ValidateResult validate(BlockHeader header) {
        ValidateResult result = ValidateResult.getSuccessResult();
        if (NulsContext.MAIN_NET_VERSION <= 1) {
            return result;
        }
        if
(ConsensusConfig.getSeedNodeStringList().indexOf(AddressTool.getStringAddressByBytes(head
er.getPackingAddress())) >= 0) {
            return result;
        }

        if (header.getHeight() == 0L) {
            return result;
        }
        if (header.getHeight() > NulsContext.getInstance().getBestHeight()) {
            return result;
        }

        BlockHeader otherBlockHeader =
blockService.getBlockHeader(header.getHeight()).getData();
        if (null != otherBlockHeader && !otherBlockHeader.getHash().equals(header.getHash()) &&
Arrays.equals(otherBlockHeader.getPackingAddress(), header.getPackingAddress())) {
            Log.info("-+-+-+-+-+-+-+-+-+- Received block with the same height and different hashes as
the latest local block -+-+-+-+-+-+-+-+-+- ");
            Log.info("-+-+-+-+-+-+-+-+-+- height" + header.getHeight() + ", hash of received block" +
```

```
header.getHash().getDigestHex()
                + ", hash of local latest block" + otherBlockHeader.getHash().getDigestHex());
        Log.info("-+-+-+-+-+-+-+-+-+- Packing address of received block" +
AddressTool.getStringAddressByBytes(header.getPackingAddress())
                + ", Packing address of local latest block" +
AddressTool.getStringAddressByBytes(otherBlockHeader.getPackingAddress()));

        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        Agent agent = null;
        for (Agent a : agentList) {
            if (a.getDelHeight() > 0) {
                continue;
            }
            if (Arrays.equals(a.getPackingAddress(), header.getPackingAddress())) {
                agent = a;
                break;
            }
        }
        if (null == agent) {
            return result;
        }
        recordEvidence(agent, header, otherBlockHeader);
        if (!isRedPunish(agent)) {
            return result;
        }
        RedPunishTransaction redPunishTransaction = new RedPunishTransaction();
        RedPunishData redPunishData = new RedPunishData();
        redPunishData.setAddress(agent.getAgentAddress());

        long txTime = 0;
        try {
            //3   3 * 2
            byte[][] headers = new byte[NulsContext.REDPUNISH_BIFURCATION * 2][];
            List<Evidence> list =
bifurcationEvidenceMap.get(AddressTool.getStringAddressByBytes(agent.getPackingAddress()));
            for (int i = 0; i < list.size() && i < NulsContext.REDPUNISH_BIFURCATION; i++) {
                Evidence evidence = list.get(i);
                int s = i * 2;
                headers[s] = evidence.getBlockHeader1().serialize();
                headers[++s] = evidence.getBlockHeader2().serialize();
                txTime =
```

```
(evidence.getBlockHeader1().getTime()+evidence.getBlockHeader2().getTime())/2;
                }
                redPunishData.setEvidence(ArraysTool.concatenate(headers));
            } catch (Exception e) {
                Log.error(e);
                return result;
            }
            redPunishData.setReasonCode(PunishReasonEnum.BIFURCATION.getCode());
            redPunishTransaction.setTxData(redPunishData);
            redPunishTransaction.setTime(txTime);
            CoinData coinData = null;
            try {
                coinData = ConsensusTool.getStopAgentCoinData(agent,
redPunishTransaction.getTime() + PocConsensusConstant.RED_PUNISH_LOCK_TIME);
            } catch (IOException e) {
                Log.error(e);
                return result;
            }
            redPunishTransaction.setCoinData(coinData);
            try {
redPunishTransaction.setHash(NulsDigestData.calcDigestData(redPunishTransaction.serializeFor
Hash()));
            } catch (IOException e) {
                Log.error(e);
                return result;
            }
            TxMemoryPool.getInstance().add(redPunishTransaction, false);
            return result;
        }

        return result;
    }

    /**
     *
     *
     * @param agent
     * @param header
     * @param otherBlockHeader
     */
    private void recordEvidence(Agent agent, BlockHeader header, BlockHeader
otherBlockHeader) {
```

```
        //PackingAddress3
        String packingAddress =
AddressTool.getStringAddressByBytes(agent.getPackingAddress());
        BlockExtendsData extendsData = new BlockExtendsData(header.getExtend());
        Evidence evidence = new Evidence(extendsData.getRoundIndex(), header,
otherBlockHeader);
        if (!bifurcationEvidenceMap.containsKey(packingAddress)) {
            List<Evidence> list = new ArrayList<>();
            list.add(evidence);
            bifurcationEvidenceMap.put(packingAddress, list);
        } else {
            List<Evidence> evidenceList = bifurcationEvidenceMap.get(packingAddress);
            if (evidenceList.size() >= NulsContext.REDPUNISH_BIFURCATION) {
                return;
            }
            ListIterator<Evidence> iterator = evidenceList.listIterator();
            boolean isSerialRoundIndex = false;
            while (iterator.hasNext()) {
                Evidence e = iterator.next();
                //
                if (e.getRoundIndex() + 1 == extendsData.getRoundIndex()) {
                    iterator.add(evidence);
                    isSerialRoundIndex = true;
                }
            }
            if (!isSerialRoundIndex) {
                //().
                bifurcationEvidenceMap.remove(packingAddress);
            }
        }
bifurcationEvidenceStorageService.save(Evidence.bifurcationEvidenceMapToPoMap(bifurcationE
videnceMap));
    }

    /**
     *
     */
    private boolean isRedPunish(Agent agent) {
        String packingAddress =
AddressTool.getStringAddressByBytes(agent.getPackingAddress());
        if (bifurcationEvidenceMap.containsKey(packingAddress)) {
            if (bifurcationEvidenceMap.get(packingAddress).size() >=
```

```
NulsContext.REDPUNISH_BIFURCATION) {
                //3
                return true;
            }
        }
        return false;
    }

}
```

4:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\cache\CacheLoader.java

```
 *
 */

package io.nuls.consensus.poc.cache;

import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.protocol.constant.PunishType;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.protocol.util.AgentComparator;
import io.nuls.consensus.poc.protocol.util.DepositComparator;
import io.nuls.consensus.poc.protocol.util.PoConvertUtil;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.consensus.poc.storage.utils.PunishLogComparator;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.protocol.service.BlockService;

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

/**
 *
```

```
 * The cmd that loads the cache when the system starts.
 *
 * @author ln
 */
public class CacheLoader {
    /**
     *
     */
    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);
    private AgentStorageService agentStorageService =
NulsContext.getServiceBean(AgentStorageService.class);
    private DepositStorageService depositStorageService =
NulsContext.getServiceBean(DepositStorageService.class);


    /**
     *
     * Loads the latest block of the specified number from the data store.
     *
     * @param size /load count
     * @return /block list
     */
    public List<Block> loadBlocks(int size) {

        List<Block> blockList = new ArrayList<>();

        Block block = blockService.getBestBlock().getData();

        if (null == block) {
            return blockList;
        }

        for (int i = size; i >= 0; i--) {

            if (block == null) {
                break;
            }

            blockList.add(0, block);

            if (block.getHeader().getHeight() == 0L) {
                break;
```

```
        }

        NulsDigestData preHash = block.getHeader().getPreHash();
        block = blockService.getBlock(preHash).getData();
        if (block == null || block.getHeader().getHeight() == 0L) {
            break;
        }
    }

    return blockList;
}

/**
 *
 * @param size
 * @return
 */
public List<BlockHeader> loadBlockHeaders(int size) {

    List<BlockHeader> blockHeaderList = new ArrayList<>();

    BlockHeader blockHeader = blockService.getBestBlockHeader().getData();

    if (null == blockHeader) {
        return blockHeaderList;
    }
    BlockExtendsData roundData = new BlockExtendsData(blockHeader.getExtend());
    long breakRoundIndex = roundData.getRoundIndex() - size;
    while (true) {
        if (blockHeader == null) {
            break;
        }

        blockHeaderList.add(0, blockHeader);

        if (blockHeader.getHeight() == 0L) {
            break;
        }

        NulsDigestData preHash = blockHeader.getPreHash();
        blockHeader = blockService.getBlockHeader(preHash).getData();
        BlockExtendsData blockRoundData = new BlockExtendsData(blockHeader.getExtend());
```

```
            if (blockRoundData.getRoundIndex() <= breakRoundIndex) {
                break;
            }
        }

        return blockHeaderList;
    }

    public List<Agent> loadAgents() {

        List<Agent> agentList = new ArrayList<>();
        List<AgentPo> poList = this.agentStorageService.getList();
        for (AgentPo po : poList) {
            Agent agent = PoConvertUtil.poToAgent(po);
            agentList.add(agent);
        }
        Collections.sort(agentList, new AgentComparator());
        return agentList;
    }

    public List<Deposit> loadDepositList() {
        List<Deposit> depositList = new ArrayList<>();
        List<DepositPo> poList = depositStorageService.getList();
        for (DepositPo po : poList) {
            depositList.add(PoConvertUtil.poToDeposit(po));
        }
        Collections.sort(depositList, new DepositComparator());
        return depositList;
    }

    public List<PunishLogPo> loadYellowPunishList(List<PunishLogPo> allPunishList, int
roundSize) {
        List<PunishLogPo> list = new ArrayList<>();
        BlockHeader blockHeader = blockService.getBestBlockHeader().getData();

        if (null == blockHeader) {
            return list;
        }
        BlockExtendsData roundData = new BlockExtendsData(blockHeader.getExtend());
        long breakRoundIndex = roundData.getRoundIndex() - roundSize;
        for (PunishLogPo po : allPunishList) {
            if (po.getType() == PunishType.RED.getCode()) {
```

```
                continue;
            }
            if (po.getRoundIndex() <= breakRoundIndex) {
                continue;
            }
            list.add(po);
        }
        Collections.sort(list, new PunishLogComparator());
        return list;
    }

    public List<PunishLogPo> loadRedPunishList(List<PunishLogPo> allPunishList) {
        List<PunishLogPo> list = new ArrayList<>();
        for (PunishLogPo po : allPunishList) {
            if (po.getType() == PunishType.RED.getCode()) {
                list.add(po);
            }
        }
        Collections.sort(list, new PunishLogComparator());
        return list;
    }
}
```

5:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\cache\TxMemoryPool.java

```
 *
 */

package io.nuls.consensus.poc.cache;

import io.nuls.cache.LimitHashMap;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Queue;
import java.util.concurrent.LinkedBlockingDeque;

/**
 *
```

```
 *
 * @author ln
 * @date 2018/4/13
 */
public final class TxMemoryPool {

    private final static TxMemoryPool INSTANCE = new TxMemoryPool();

    private Queue<Transaction> txQueue;

    private LimitHashMap<NulsDigestData, Transaction> orphanContainer;

    private TxMemoryPool() {
        txQueue = new LinkedBlockingDeque<>();

//        orphanContainer = new CacheMap<>("orphan-txs", 256, NulsDigestData.class,
TxContainer.class, 3600, 0, null);
        this.orphanContainer = new LimitHashMap(200000);
    }

    public static TxMemoryPool getInstance() {
        return INSTANCE;
    }

    public boolean addInFirst(Transaction tx, boolean isOrphan) {
        try {
            if (tx == null) {
                return false;
            }
            //check Repeatability
            if (isOrphan) {
                NulsDigestData hash = tx.getHash();
                orphanContainer.put(hash, tx);
            } else {
                ((LinkedBlockingDeque) txQueue).addFirst(tx);
            }
            return true;
        } finally {
        }
    }

    public boolean add(Transaction tx, boolean isOrphan) {
```

```
        try {
            if (tx == null) {
                return false;
            }
            //check Repeatability
            if (isOrphan) {
                NulsDigestData hash = tx.getHash();
                orphanContainer.put(hash, tx);
            } else {
                txQueue.offer(tx);
            }
            return true;
        } finally {
        }
    }

    /**
     * Get a TxContainer, the first TxContainer received, removed from the memory pool after acquisition
     * <p>
     *
     *
     * @return TxContainer
     */
    public Transaction get() {
        return txQueue.poll();
    }

    public List<Transaction> getAll() {
        List<Transaction> txs = new ArrayList<>();
        Iterator<Transaction> it = txQueue.iterator();
        while (it.hasNext()) {
            txs.add(it.next());
        }
        return txs;
    }

    public List<Transaction> getAllOrphan() {
        return new ArrayList<>(orphanContainer.values());
    }

    public boolean remove(NulsDigestData hash) {
```

```
//        TxContainer obj = container.remove(hash);
//        if (obj != null) {
//            txHashQueue.remove(hash);
//        } else {
        orphanContainer.remove(hash);
//        }
        return true;
    }

    public boolean exist(NulsDigestData hash) {
        return /*container.containsKey(hash) || */orphanContainer.containsKey(hash);
    }

    public void clear() {
        try {
            txQueue.clear();

            orphanContainer.clear();
        } finally {
        }
    }

    public int size() {
        return txQueue.size();
    }

    public int getPoolSize() {
        return txQueue.size();
    }

    public int getOrphanPoolSize() {
        return orphanContainer.size();
    }

    public void removeOrphan(NulsDigestData hash) {
        this.orphanContainer.remove(hash);
    }
}

6:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\config\ConsensusConfig.java
 */
```

```
package io.nuls.consensus.poc.config;

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.utils.AddressTool;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * @author ln
 */
public class ConsensusConfig {

    private final static String CFG_CONSENSUS_SECTION = "consensus";
    private final static String PROPERTY_PARTAKE_PACKING = "partake.packing";
    private final static String PROPERTY_SEED_NODES = "seed.nodes";
    private final static String SEED_NODES_DELIMITER = ",";

    private static boolean partakePacking = false;
    private static List<byte[]> seedNodeBytesList = new ArrayList<>();
    private static List<String> seedNodeStringList = new ArrayList<>();

    public static void initConfiguration() throws Exception {

        Block genesisBlock = GenesisBlock.getInstance();
        NulsContext.getInstance().setGenesisBlock(genesisBlock);

        partakePacking =
NulsConfig.MODULES_CONFIG.getCfgValue(CFG_CONSENSUS_SECTION,
PROPERTY_PARTAKE_PACKING, false);
        Set<String> seedAddressSet = new HashSet<>();
        String addresses =
NulsConfig.MODULES_CONFIG.getCfgValue(CFG_CONSENSUS_SECTION,
PROPERTY_SEED_NODES, "");
        if (StringUtils.isBlank(addresses)) {
            return;
        }
```

```
        String[] array = addresses.split(SEED_NODES_DELIMITER);
        if (null == array) {
            return;
        }
        for (String address : array) {
            seedAddressSet.add(address);
        }
        for (String address : seedAddressSet) {
            seedNodeBytesList.add(AddressTool.getAddress(address));
            seedNodeStringList.add(address);
        }
    }

    public static boolean isPartakePacking() {
        return partakePacking;
    }

    public static List<byte[]> getSeedNodeList() {
        return seedNodeBytesList;
    }

    public static List<String> getSeedNodeStringList() {
        return seedNodeStringList;
    }
}
```

7:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\config\GenesisBlock.java

```
 */
package io.nuls.consensus.poc.config;

import io.nuls.account.service.AccountService;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.io.StringFileLoader;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.context.NulsContext;
```

```
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.*;

import io.nuls.kernel.script.BlockSignature;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.tx.CoinBaseTransaction;

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

/**
 * @author Niels
 */
public final class GenesisBlock extends Block {

    private final static String GENESIS_BLOCK_FILE = "block/genesis-block.json";

    private static final String CONFIG_FILED_TIME = "time";
    private static final String CONFIG_FILED_HEIGHT = "height";
    private static final String CONFIG_FILED_TXS = "txs";
    private static final String CONFIG_FILED_ADDRESS = "address";
    private static final String CONFIG_FILED_AMOUNT = "amount";
    private static final String CONFIG_FILED_LOCK_TIME = "lockTime";
    private static final String CONFIG_FILED_REMARK = "remark";
    private static final String priKey =
"009cf05b6b3fe8c09b84c13783140c0f1958e8841f8b6f894ef69431522bc65712";

    private static GenesisBlock INSTANCE = new GenesisBlock();

    private transient long blockTime;

    private transient int status = 0;

    public static GenesisBlock getInstance() throws Exception {
        if (INSTANCE.status == 0) {
            String json = null;
            try {
```

```
                json = StringFileLoader.read(GENESIS_BLOCK_FILE);
            } catch (NulsException e) {
                Log.error(e);
            }
            INSTANCE.init(json);
        }
        return INSTANCE;
    }

    private GenesisBlock() {

    }

    private synchronized void init(String json) throws Exception {
        if (status > 0) {
            return;
        }
        Map<String, Object> jsonMap = null;
        try {
            jsonMap = JSONUtils.json2map(json);
        } catch (Exception e) {
            Log.error(e);
        }
        String time = (String) jsonMap.get(CONFIG_FILED_TIME);
        AssertUtil.canNotEmpty(time, KernelErrorCode.CONFIG_ERROR.getMsg());
        blockTime = Long.parseLong(time);
        this.initGengsisTxs(jsonMap);
        this.fillHeader(jsonMap);
        ValidateResult validateResult = this.verify();
        if (validateResult.isFailed()) {
            throw new NulsRuntimeException(validateResult.getErrorCode());
        }
        this.status = 1;
    }

    private void initGengsisTxs(Map<String, Object> jsonMap) throws Exception {
        List<Map<String, Object>> list = (List<Map<String, Object>>)
jsonMap.get(CONFIG_FILED_TXS);
        if (null == list || list.isEmpty()) {
            throw new NulsRuntimeException(KernelErrorCode.CONFIG_ERROR);
        }
```

```
        CoinData coinData = new CoinData();

        for (Map<String, Object> map : list) {
            String address = (String) map.get(CONFIG_FILED_ADDRESS);
            AssertUtil.canNotEmpty(address, KernelErrorCode.NULL_PARAMETER.getMsg());

            Double amount = Double.valueOf("" + map.get(CONFIG_FILED_AMOUNT));
            AssertUtil.canNotEmpty(amount, KernelErrorCode.NULL_PARAMETER.getMsg());
            Long lockTime = Long.valueOf("" + map.get(CONFIG_FILED_LOCK_TIME));

            Address ads = Address.fromHashs(address);

            Coin coin = new Coin(ads.getAddressBytes(), Na.parseNuls(amount), lockTime == null ? 0
: lockTime.longValue());
            coinData.addTo(coin);
        }

        CoinBaseTransaction tx = new CoinBaseTransaction();
        tx.setTime(this.blockTime);
        tx.setCoinData(coinData);
        String remark = (String) jsonMap.get(CONFIG_FILED_REMARK);
        if(StringUtils.isNotBlank(remark)) {
            tx.setRemark(Hex.decode(remark));
        }
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        List<Transaction> txlist = new ArrayList<>();
        txlist.add(tx);
        setTxs(txlist);
    }


    private void fillHeader(Map<String, Object> jsonMap) throws NulsException {
        Integer height = (Integer) jsonMap.get(CONFIG_FILED_HEIGHT);
        AssertUtil.canNotEmpty(height, KernelErrorCode.CONFIG_ERROR.getMsg());

        BlockHeader header = new BlockHeader();
        this.setHeader(header);
        header.setHeight(height);
        header.setTime(blockTime);
        header.setPreHash(NulsDigestData.calcDigestData(new byte[35]));
        header.setTxCount(this.getTxs().size());
        List<NulsDigestData> txHashList = new ArrayList<>();
```

```
        for (Transaction tx : this.getTxs()) {
            txHashList.add(tx.getHash());
        }
        header.setMerkleHash(NulsDigestData.calcMerkleDigestData(txHashList));

        BlockExtendsData data = new BlockExtendsData();
        data.setRoundIndex(1);
        data.setRoundStartTime(header.getTime() -
ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND * 1000);
        data.setConsensusMemberCount(1);
        data.setPackingIndexOfRound(1);
        try {
            header.setExtend(data.serialize());
        } catch (IOException e) {
            throw new NulsRuntimeException(e);

        }
        header.setHash(NulsDigestData.calcDigestData(header));

        BlockSignature p2PKHScriptSig = new BlockSignature();
        NulsSignData signData = this.signature(header.getHash().getDigestBytes());
        p2PKHScriptSig.setSignData(signData);
        p2PKHScriptSig.setPublicKey(getGenesisPubkey());
        header.setBlockSignature(p2PKHScriptSig);
    }

    private NulsSignData signature(byte[] bytes) throws NulsException {
        AccountService service = NulsContext.getServiceBean(AccountService.class);
        return service.signDigest(bytes, ECKey.fromPrivate(new BigInteger(1, Hex.decode(priKey))));
    }

    private byte[] getGenesisPubkey() {
        return ECKey.fromPrivate(new BigInteger(1, Hex.decode(priKey))).getPubKey();
    }
}



8:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\constant\BlockContainerStatus.java
 *
 */
```

```
package io.nuls.consensus.poc.constant;

/**
 *
 * @author ln
 */
public final class BlockContainerStatus {

    public final static int DOWNLOADING = 1;
    public final static int RECEIVED = 2;

}
```

9:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\constant\ConsensusStatus.java

```
 *
 */

package io.nuls.consensus.poc.constant;

/**
 *
 * @author ln
 */
public enum ConsensusStatus {

    // Warning, the following order cannot be adjusted, otherwise the judgment in some places may go wrong
    //

    INITING,
    LOADING_CACHE,
    WAIT_RUNNING,
    RUNNING,
}
```

10:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\constant\PocConsensusConstant.java

```
 *
 */

package io.nuls.consensus.poc.constant;
```

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

/**
 * @author ln
 */
public interface PocConsensusConstant {

    /**
     * Coinbase rewards the number of locked blocks
     * coinbase
     */
    // pierre test comment out
    int COINBASE_UNLOCK_HEIGHT = 1000;

    /**
     * value = 5000000/3154600
     */
    Na BLOCK_REWARD = Na.valueOf(158548960);

    /**
     * Maximum height difference handled by furcation blocks
     *
     */
    int CHANGE_CHAIN_BLOCK_DIFF_COUNT = 3;

    /**
     * Maximum height difference handled by furcation blocks , Blocks that exceed this difference will be discarded directly
     *
     */
    int MAX_ISOLATED_BLOCK_COUNT = 1000;

    /**
     * How long does the current network time exceed the number of blocks that are discarded directly, in milliseconds
     *
     */
    long DISCARD_FUTURE_BLOCKS_TIME = 60 * 1000L;

    /**
     * Load the block header of the last specified number of rounds during initialization
```

```
     *
     */
    int INIT_HEADERS_OF_ROUND_COUNT = 200;

    /**
     * When the system starts up, load the newly specified number of blocks into memory
     *
     */
    int INIT_BLOCKS_COUNT = 10;

    /**
     * Consensus memory expiration data cleaning interval, in milliseconds
     *
     */
    long CLEAR_INTERVAL_TIME = 60000L;

    /**
     * Regularly clear the round before the specified number of rounds of the main chain
     *
     */
    int CLEAR_MASTER_CHAIN_ROUND_COUNT = 5;

    /**
     * The maximum continuous number of yellow punish log.
     */
    int MAXINUM_CONTINUOUS_YELLOW_NUMBER = 100;

    /**
     * reset system time interval , unit minutes
     */
    int RESET_SYSTEM_TIME_INTERVAL = 5;
    long CONSENSUS_LOCK_TIME = -1;
    long STOP_AGENT_LOCK_TIME = 3 * 24 * 3600000L;
    long RED_PUNISH_LOCK_TIME = 60 * 24 * 3600000L;

    double RED_PUNISH_CREDIT_VAL = -1D;
}

11:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\container\BlockContainer.java
 *
 */
```

```
package io.nuls.consensus.poc.container;

import io.nuls.kernel.model.Block;
import io.nuls.network.model.Node;

/**
 * @author ln
 */
public class BlockContainer {

    private Block block;
    private Node node;
    private int status;

    public BlockContainer() {
    }

    public BlockContainer(Block block, int status) {
        this.block = block;
        this.status = status;
    }

    public BlockContainer(Block block, Node node, int status) {
        this.block = block;
        this.node = node;
        this.status = status;
    }

    public Block getBlock() {
        return block;
    }

    public void setBlock(Block block) {
        this.block = block;
    }

    public int getStatus() {
        return status;
    }

    public void setStatus(int status) {
```

```
        this.status = status;
    }

    public Node getNode() {
        return node;
    }

    public void setNode(Node node) {
        this.node = node;
    }
}

12:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\container\ChainContainer.java
 *
 */

package io.nuls.consensus.poc.container;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.config.ConsensusConfig;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.manager.RoundManager;
import io.nuls.consensus.poc.model.*;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.constant.PunishReasonEnum;
import io.nuls.consensus.poc.protocol.constant.PunishType;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.consensus.poc.protocol.tx.*;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.util.ConsensusTool;
import io.nuls.core.tools.log.BlockLog;
import io.nuls.core.tools.log.Log;
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.model.*;
```

```
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.SmallBlock;
import io.nuls.protocol.model.tx.CoinBaseTransaction;
import io.nuls.protocol.model.validator.HeaderSignValidator;
import io.nuls.protocol.service.BlockService;

import java.io.IOException;
import java.util.*;

/**
 * @author ln
 */
public class ChainContainer implements Cloneable {

    private Chain chain;
    private RoundManager roundManager;

    public ChainContainer(Chain chain) {
        this.chain = chain;
        roundManager = new RoundManager(chain);
    }

    public boolean addBlock(Block block) {

        if (!chain.getEndBlockHeader().getHash().equals(block.getHeader().getPreHash()) ||
                chain.getEndBlockHeader().getHeight() + 1 != block.getHeader().getHeight()) {
            return false;
        }

        List<Block> blockList = chain.getBlockList();
        List<BlockHeader> blockHeaderList = chain.getBlockHeaderList();

        List<Agent> agentList = chain.getAgentList();
        List<Deposit> depositList = chain.getDepositList();
        List<PunishLogPo> yellowList = chain.getYellowPunishList();
        List<PunishLogPo> redList = chain.getRedPunishList();
```

```
long height = block.getHeader().getHeight();
BlockExtendsData extendsData = new BlockExtendsData(block.getHeader().getExtend());
List<Transaction> txs = block.getTxs();
for (Transaction tx : txs) {
    int txType = tx.getType();
    if (txType == ConsensusConstant.TX_TYPE_REGISTER_AGENT) {
        // Registered agent transaction
        //
        CreateAgentTransaction registerAgentTx = (CreateAgentTransaction) tx;

        CreateAgentTransaction agentTx = registerAgentTx.clone();
        Agent agent = agentTx.getTxData();
        agent.setDelHeight(-1L);
        agent.setBlockHeight(height);
        agent.setTxHash(agentTx.getHash());
        agent.setTime(agentTx.getTime());

        agentList.add(agent);
    } else if (txType == ConsensusConstant.TX_TYPE_JOIN_CONSENSUS) {

        //
        DepositTransaction joinConsensusTx = (DepositTransaction) tx;

        DepositTransaction depositTx = joinConsensusTx.clone();

        Deposit deposit = depositTx.getTxData();
        deposit.setDelHeight(-1L);
        deposit.setBlockHeight(height);
        deposit.setTxHash(depositTx.getHash());
        deposit.setTime(depositTx.getTime());
        depositList.add(deposit);

    } else if (txType == ConsensusConstant.TX_TYPE_CANCEL_DEPOSIT) {

        CancelDepositTransaction cancelDepositTx = (CancelDepositTransaction) tx;

        NulsDigestData joinHash = cancelDepositTx.getTxData().getJoinTxHash();

        Iterator<Deposit> it = depositList.iterator();
        while (it.hasNext()) {
            Deposit deposit = it.next();
            cancelDepositTx.getTxData().setAddress(deposit.getAddress());
```

```
            if (deposit.getTxHash().equals(joinHash)) {
                if (deposit.getDelHeight() == -1L) {
                    deposit.setDelHeight(height);
                }
                break;
            }
        }
    } else if (txType == ConsensusConstant.TX_TYPE_STOP_AGENT) {

        StopAgentTransaction stopAgentTx = (StopAgentTransaction) tx;

        NulsDigestData agentHash = stopAgentTx.getTxData().getCreateTxHash();

        Iterator<Deposit> it = depositList.iterator();
        while (it.hasNext()) {
            Deposit deposit = it.next();
            if (deposit.getAgentHash().equals(agentHash) && deposit.getDelHeight() == -1L) {
                deposit.setDelHeight(height);
            }
        }

        Iterator<Agent> ita = agentList.iterator();
        while (ita.hasNext()) {
            Agent agent = ita.next();
            stopAgentTx.getTxData().setAddress(agent.getAgentAddress());
            if (agent.getTxHash().equals(agentHash)) {
                if (agent.getDelHeight() == -1L) {
                    agent.setDelHeight(height);
                }
                break;
            }
        }
    } else if (txType == ConsensusConstant.TX_TYPE_RED_PUNISH) {
        RedPunishTransaction transaction = (RedPunishTransaction) tx;
        RedPunishData redPunishData = transaction.getTxData();
        PunishLogPo po = new PunishLogPo();
        po.setAddress(redPunishData.getAddress());
        po.setHeight(height);
        po.setRoundIndex(extendsData.getRoundIndex());
        po.setTime(tx.getTime());
        po.setType(PunishType.RED.getCode());
        redList.add(po);
```

```
            for (Agent agent : agentList) {
                if (!Arrays.equals(agent.getAgentAddress(), po.getAddress())) {
                    continue;
                }
                if (agent.getDelHeight() > 0) {
                    continue;
                }
                agent.setDelHeight(height);
                for (Deposit deposit : depositList) {
                    if (!deposit.getAgentHash().equals(agent.getTxHash())) {
                        continue;
                    }
                    if (deposit.getDelHeight() > 0) {
                        continue;
                    }
                    deposit.setDelHeight(height);
                }
            }
        } else if (txType == ConsensusConstant.TX_TYPE_YELLOW_PUNISH) {
            YellowPunishTransaction transaction = (YellowPunishTransaction) tx;
            for (byte[] bytes : transaction.getTxData().getAddressList()) {
                PunishLogPo po = new PunishLogPo();
                po.setAddress(bytes);
                po.setHeight(height);
                po.setRoundIndex(extendsData.getRoundIndex());
                po.setTime(tx.getTime());
                po.setType(PunishType.YELLOW.getCode());
                yellowList.add(po);
            }
        }
    }

    chain.setEndBlockHeader(block.getHeader());
    blockList.add(block);
    blockHeaderList.add(block.getHeader());

    return true;
}

public Result verifyBlock(Block block) {
    return verifyBlock(block, false, true);
}
```

```
    public Result verifyBlock(Block block, boolean isDownload, boolean isNeedCheckCoinBaseTx)
{

        if (block == null || chain.getEndBlockHeader() == null) {
            return Result.getFailed();
        }

        BlockHeader blockHeader = block.getHeader();
        if (blockHeader == null) {
            return Result.getFailed();
        }
        //todo
        block.verifyWithException();

        // Verify that the block is properly connected
        //
        NulsDigestData preHash = blockHeader.getPreHash();

        BlockHeader bestBlockHeader = chain.getEndBlockHeader();

        if (!preHash.equals(bestBlockHeader.getHash())) {
            Log.error("block height " + blockHeader.getHeight() + " prehash is error! hash :" +
blockHeader.getHash() + ", prehash : " + preHash);
            Log.error("preblock height " + chain.getEndBlockHeader().getHeight() + " prehash is error!
EndBlockHeader hash:" + chain.getEndBlockHeader().getHash() + ", prehash : " +
blockHeader.getPreHash());
            return Result.getFailed();
        }

        BlockExtendsData extendsData = new BlockExtendsData(blockHeader.getExtend());
        BlockExtendsData bestExtendsData = new
BlockExtendsData(bestBlockHeader.getExtend());

        //
        if (extendsData.getRoundIndex() < bestExtendsData.getRoundIndex() ||
                (extendsData.getRoundIndex() == bestExtendsData.getRoundIndex() &&
extendsData.getPackingIndexOfRound() <= bestExtendsData.getPackingIndexOfRound())) {
            Log.error("new block rounddata error, block height : " + blockHeader.getHeight() + " , hash
:" + blockHeader.getHash());
            return Result.getFailed();
```

```
        }

        if (NulsContext.MAIN_NET_VERSION > 1) {
            //1.0
            if (extendsData.getCurrentVersion() == null && NulsVersionManager.getMainVersion() > 1)
{
                Log.info("------block currentVersion low, hash :" +
block.getHeader().getHash().getDigestHex() + ", packAddress:" +
AddressTool.getStringAddressByBytes(block.getHeader().getPackingAddress()));
                return Result.getFailed();
            } else if (null != extendsData.getCurrentVersion() && extendsData.getCurrentVersion() <
NulsVersionManager.getMainVersion()) {
                Log.info("------block currentVersion low, hash :" +
block.getHeader().getHash().getDigestHex() + ", packAddress:" +
AddressTool.getStringAddressByBytes(block.getHeader().getPackingAddress()));
                return Result.getFailed();
            } else if (extendsData.getCurrentVersion() != null && extendsData.getPercent() != null &&
extendsData.getPercent() < ProtocolConstant.MIN_PROTOCOL_UPGRADE_RATE) {
                //60%
                Log.info("------block currentVersion percent error, hash :" +
block.getHeader().getHash().getDigestHex() + ", packAddress:" +
AddressTool.getStringAddressByBytes(block.getHeader().getPackingAddress()));
                return Result.getFailed();
            } else if (extendsData.getCurrentVersion() != null && extendsData.getDelay() != null &&
extendsData.getDelay() < ProtocolConstant.MIN_PROTOCOL_UPGRADE_DELAY) {
                //1000
                Log.info("------block currentVersion delay error, hash :" +
block.getHeader().getHash().getDigestHex() + ", packAddress:" +
AddressTool.getStringAddressByBytes(block.getHeader().getPackingAddress()));
                return Result.getFailed();
            }
        }
        MeetingRound currentRound = roundManager.getCurrentRound();

        if (isDownload && currentRound.getIndex() > extendsData.getRoundIndex()) {
            MeetingRound round = roundManager.getRoundByIndex(extendsData.getRoundIndex());
            if (round != null) {
                currentRound = round;
            }
        }

        boolean hasChangeRound = false;
```

```
        // Verify that the block round and time are correct
        //
//        if(roundData.getRoundIndex() > currentRound.getIndex()) {
//            Log.error("block height " + blockHeader.getHeight() + " round index is error!");
//            Result.getFailed();
//        }
        if (extendsData.getRoundIndex() > currentRound.getIndex()) {
            if (extendsData.getRoundStartTime() > TimeService.currentTimeMillis() +
ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS) {
                Log.error("block height " + blockHeader.getHeight() + " round startTime is error, greater
than current time! hash :" + blockHeader.getHash());
                return Result.getFailed();
            }
            if (!isDownload && (extendsData.getRoundStartTime() +
(extendsData.getPackingIndexOfRound() - 1) *
ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS) > TimeService.currentTimeMillis() +
ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS) {
                Log.error("block height " + blockHeader.getHeight() + " is the block of the future and
received in advance! hash :" + blockHeader.getHash());
                return Result.getFailed();
            }
            if (extendsData.getRoundStartTime() < currentRound.getEndTime()) {
                Log.error("block height " + blockHeader.getHeight() + " round index and start time not
match! hash :" + blockHeader.getHash());
                return Result.getFailed();
            }
            MeetingRound tempRound = roundManager.getNextRound(extendsData, !isDownload);
            if (tempRound.getIndex() > currentRound.getIndex()) {
                tempRound.setPreRound(currentRound);
                hasChangeRound = true;
            }
            currentRound = tempRound;
        } else if (extendsData.getRoundIndex() < currentRound.getIndex()) {
            MeetingRound preRound = currentRound.getPreRound();
            while (preRound != null) {
                if (extendsData.getRoundIndex() == preRound.getIndex()) {
                    currentRound = preRound;
                    break;
                }
                preRound = preRound.getPreRound();
            }
```

```
        }

        if (extendsData.getRoundIndex() != currentRound.getIndex() ||
extendsData.getRoundStartTime() != currentRound.getStartTime()) {
            Log.error("block height " + blockHeader.getHeight() + " round startTime is error! hash :" +
blockHeader.getHash());
            return Result.getFailed();
        }

        Log.debug(currentRound.toString());

        if (extendsData.getConsensusMemberCount() != currentRound.getMemberCount()) {
            Log.error("block height " + blockHeader.getHeight() + " packager count is error! hash :" +
blockHeader.getHash());
            return Result.getFailed();
        }
        // Verify that the packager is correct
        //
        MeetingMember member =
currentRound.getMember(extendsData.getPackingIndexOfRound());
        if (!Arrays.equals(member.getPackingAddress(), blockHeader.getPackingAddress())) {
            Log.error("block height " + blockHeader.getHeight() + " packager error! hash :" +
blockHeader.getHash());
            return Result.getFailed();
        }

        if (member.getPackEndTime() != block.getHeader().getTime()) {
            Log.error("block height " + blockHeader.getHeight() + " time error! hash :" +
blockHeader.getHash());
            return Result.getFailed();
        }

        boolean success = verifyPunishTx(block, currentRound, member);
        if (!success) {
            Log.error("block height " + blockHeader.getHeight() + " verify tx error! hash :" +
blockHeader.getHash());
            return Result.getFailed();
        }
        //
        if (isNeedCheckCoinBaseTx) {
            boolean isCorrect = verifyCoinBaseTx(block, currentRound, member);
            if (!isCorrect) {
```

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

        if (hasChangeRound) {
            roundManager.addRound(currentRound);
        }
        Object[] objects = new Object[]{currentRound, member};
        return Result.getSuccess().setData(objects);
    }

    public boolean verifyCoinBaseTx(Block block, MeetingRound currentRound, MeetingMember
member) {
        Transaction tx = block.getTxs().get(0);
        CoinBaseTransaction coinBaseTransaction = ConsensusTool.createCoinBaseTx(member,
block.getTxs(), currentRound, block.getHeader().getHeight() +
PocConsensusConstant.COINBASE_UNLOCK_HEIGHT);
        if (null == coinBaseTransaction || !tx.getHash().equals(coinBaseTransaction.getHash())) {
            BlockLog.debug("the coin base tx is wrong! height: " + block.getHeader().getHeight() + " ,
hash : " + block.getHeader().getHash());
            Log.error("the coin base tx is wrong! height: " + block.getHeader().getHeight() + " , hash : "
+ block.getHeader().getHash());
            return false;
        }
        return true;
    }

    // Verify penalties
    //
    public boolean verifyPunishTx(Block block, MeetingRound currentRound, MeetingMember
member) {
        List<Transaction> txs = block.getTxs();
        Transaction tx = txs.get(0);
        if (tx.getType() != ProtocolConstant.TX_TYPE_COINBASE) {
            BlockLog.debug("Coinbase transaction order wrong! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
//            Log.error("Coinbase transaction order wrong! height: " + block.getHeader().getHeight() +
" , hash : " + block.getHeader().getHash());
            return false;
        }
        List<RedPunishTransaction> redPunishTxList = new ArrayList<>();
        YellowPunishTransaction yellowPunishTx = null;
```

```
        for (int i = 1; i < txs.size(); i++) {
            Transaction transaction = txs.get(i);
            if (transaction.getType() == ProtocolConstant.TX_TYPE_COINBASE) {
                BlockLog.debug("Coinbase transaction more than one! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
//                Log.error("Coinbase transaction more than one! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
                return false;
            }
            if (null == yellowPunishTx && transaction.getType() ==
ConsensusConstant.TX_TYPE_YELLOW_PUNISH) {
                yellowPunishTx = (YellowPunishTransaction) transaction;
            } else if (null != yellowPunishTx && transaction.getType() ==
ConsensusConstant.TX_TYPE_YELLOW_PUNISH) {
                BlockLog.debug("Yellow punish transaction more than one! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
//                Log.error("Yellow punish transaction more than one! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
                return false;
            } else if (transaction.getType() == ConsensusConstant.TX_TYPE_RED_PUNISH) {
                redPunishTxList.add((RedPunishTransaction) transaction);
            }
        }

        YellowPunishTransaction yellowPunishTransaction = null;
        try {
            yellowPunishTransaction = ConsensusTool.createYellowPunishTx(chain.getBestBlock(),
member, currentRound);
            if (yellowPunishTransaction == null && yellowPunishTx == null) {
                //
            } else if (yellowPunishTransaction == null || yellowPunishTx == null) {
                BlockLog.debug("The yellow punish tx is wrong! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
//                Log.error("The yellow punish tx is wrong! height: " + block.getHeader().getHeight() + " ,
hash : " + block.getHeader().getHash());
                return false;
            } else if (!yellowPunishTransaction.getHash().equals(yellowPunishTx.getHash())) {
                BlockLog.debug("The yellow punish tx's hash is wrong! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
//                Log.error("The yellow punish tx's hash is wrong! height: " +
block.getHeader().getHeight() + " , hash : " + block.getHeader().getHash());
                return false;
```

```
            }

        } catch (Exception e) {
            BlockLog.debug("The tx's wrong! height: " + block.getHeader().getHeight() + " , hash : " +
block.getHeader().getHash(), e);
//            Log.error("The tx's wrong! height: " + block.getHeader().getHeight() + " , hash : " +
block.getHeader().getHash(), e);
            return false;
        }
        if (!redPunishTxList.isEmpty()) {
            Set<String> punishAddress = new HashSet<>();
            if (null != yellowPunishTx) {
                List<byte[]> addressList = yellowPunishTx.getTxData().getAddressList();
                for (byte[] address : addressList) {
                    MeetingMember item = currentRound.getMemberByAgentAddress(address);
                    if (null == item) {
                        item = currentRound.getPreRound().getMemberByAgentAddress(address);
                    }
                    if (item.getCreditVal() <= PocConsensusConstant.RED_PUNISH_CREDIT_VAL) {
punishAddress.add(AddressTool.getStringAddressByBytes(item.getAgent().getAgentAddress()));
                 }
                }
            }
            int countOfTooMuchYP = 0;
            for (RedPunishTransaction redTx : redPunishTxList) {
                RedPunishData data = redTx.getTxData();
                if (data.getReasonCode() ==
PunishReasonEnum.TOO_MUCH_YELLOW_PUNISH.getCode()) {
                    countOfTooMuchYP++;
                    if
(!punishAddress.contains(AddressTool.getStringAddressByBytes(redTx.getTxData().getAddress())
)) {
                        BlockLog.debug("There is a wrong red punish tx!" + block.getHeader().getHash());
                        return false;
                    }
                    if (NulsContext.MAIN_NET_VERSION > 1 && redTx.getTime() !=
block.getHeader().getTime()) {
                        BlockLog.debug("red punish CoinData & TX time is wrong! " +
block.getHeader().getHash());
                        return false;
                    }
                }
```

```
                boolean result = verifyRedPunish(redTx);
                if (!result) {
                    return result;
                }
            }
            if (countOfTooMuchYP != punishAddress.size()) {
                BlockLog.debug("There is a wrong red punish tx!" + block.getHeader().getHash());
                return false;
            }

        }
        return true;
    }

    private boolean verifyYellowPunish(YellowPunishTransaction data) {
        if (null == data || data.getTxData() == null || data.getTxData().getAddressList() == null ||
data.getTxData().getAddressList().isEmpty()) {
            Log.warn(PocConsensusErrorCode.YELLOW_PUNISH_TX_WRONG.getMsg());
            return false;
        }
        List<byte[]> list = data.getTxData().getAddressList();
        for (byte[] address : list) {
            if
(ConsensusConfig.getSeedNodeStringList().contains(AddressTool.getStringAddressByBytes(addr
ess))) {
                return false;
            }
        }
        if (data.getCoinData() != null) {
            Log.warn(PocConsensusErrorCode.YELLOW_PUNISH_TX_WRONG.getMsg());
            return false;
        }
        return true;
    }

    private boolean verifyRedPunish(RedPunishTransaction data) {
        RedPunishData punishData = data.getTxData();
        if
(ConsensusConfig.getSeedNodeStringList().contains(AddressTool.getStringAddressByBytes(puni
shData.getAddress()))) {
            Log.warn("The seed node can not be punished!");
            return false;
```

```
        }
        HeaderSignValidator validator = NulsContext.getServiceBean(HeaderSignValidator.class);
        LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);
        if (punishData.getReasonCode() == PunishReasonEnum.DOUBLE_SPEND.getCode()) {
            if (NulsContext.MAIN_NET_VERSION <= 1) {
                Log.warn("The red punish tx need higher version!");
                return false;
            }
            SmallBlock smallBlock = new SmallBlock();
            try {
                smallBlock.parse(punishData.getEvidence(), 0);
            } catch (NulsException e) {
                Log.error(e);
                return false;
            }
            BlockHeader header = smallBlock.getHeader();
            if (header.getTime() != data.getTime()) {
                return false;
            }
            ValidateResult result = validator.validate(header);
            if (result.isFailed()) {
                Log.warn(result.getMsg());
                return false;
            }
            List<NulsDigestData> txHashList = smallBlock.getTxHashList();
            if (!header.getMerkleHash().equals(NulsDigestData.calcMerkleDigestData(txHashList))) {
                Log.warn(TransactionErrorCode.TX_DATA_VALIDATION_ERROR.getMsg());
                return false;
            }
            List<Transaction> txList = smallBlock.getSubTxList();
            if (null == txList || txList.size() < 2) {
                Log.warn(TransactionErrorCode.TX_DATA_VALIDATION_ERROR.getMsg());
                return false;
            }
            result = ledgerService.verifyDoubleSpend(txList);
            if (result.isSuccess()) {
Log.warn(PocConsensusErrorCode.TRANSACTIONS_NEVER_DOUBLE_SPEND.getMsg());
        return false;
            }
        } else if (punishData.getReasonCode() == PunishReasonEnum.BIFURCATION.getCode()) {
            if (NulsContext.MAIN_NET_VERSION <= 1) {
                Log.warn("The red punish tx need higher version!");
```

```
            return false;
        }
        NulsByteBuffer byteBuffer = new NulsByteBuffer(punishData.getEvidence());
        //
        long[] roundIndex = new long[NulsContext.REDPUNISH_BIFURCATION];
        for (int i = 0; i < NulsContext.REDPUNISH_BIFURCATION && !byteBuffer.isFinished();
i++) {
            BlockHeader header1 = null;
            BlockHeader header2 = null;
            try {
                header1 = byteBuffer.readNulsData(new BlockHeader());
                header2 = byteBuffer.readNulsData(new BlockHeader());
            } catch (NulsException e) {
                Log.error(e);
            }
            if (null == header1 || null == header2) {
                Log.warn(KernelErrorCode.DATA_NOT_FOUND.getMsg());
                return false;
            }
            if (header1.getHeight() != header2.getHeight()) {
                Log.warn(TransactionErrorCode.TX_DATA_VALIDATION_ERROR.getMsg());
                return false;
            }
            if (i == NulsContext.REDPUNISH_BIFURCATION - 1 && (header1.getTime() +
header2.getTime()) / 2 != data.getTime()) {
                return false;
            }
            ValidateResult result = validator.validate(header1);
            if (result.isFailed()) {
                Log.warn(TransactionErrorCode.TX_DATA_VALIDATION_ERROR.getMsg());
                return false;
            }
            result = validator.validate(header2);
            if (result.isFailed()) {
                Log.warn(TransactionErrorCode.TX_DATA_VALIDATION_ERROR.getMsg());
                return false;
            }
            if (!Arrays.equals(header1.getBlockSignature().getPublicKey(),
header2.getBlockSignature().getPublicKey())) {
                Log.warn(TransactionErrorCode.SIGNATURE_ERROR.getMsg());
                return false;
            }
```

```
            BlockExtendsData blockExtendsData = new BlockExtendsData(header1.getExtend());
            roundIndex[i] = blockExtendsData.getRoundIndex();
        }
        //
        boolean rs = true;
        for (int i = 0; i < roundIndex.length; i++) {
            if (i < roundIndex.length - 2 && roundIndex[i + 1] - roundIndex[i] != 1) {
                rs = false;
                break;
            }
        }
        if (!rs) {
            Log.warn(PocConsensusErrorCode.RED_CARD_VERIFICATION_FAILED.getMsg());
            return false;
        }
    } else if (punishData.getReasonCode() !=
PunishReasonEnum.TOO_MUCH_YELLOW_PUNISH.getCode()) {
        Log.warn(PocConsensusErrorCode.RED_CARD_VERIFICATION_FAILED.getMsg());
        return false;
    }

    try {
        return verifyCoinData(data);
    } catch (IOException e) {
        Log.error(e);
        return false;
    }
}

private boolean verifyCoinData(RedPunishTransaction tx) throws IOException {
    List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
    Agent theAgent = null;
    for (Agent agent : agentList) {
        if (agent.getDelHeight() > 0 && (tx.getBlockHeight() <= 0 || agent.getDelHeight() <
tx.getBlockHeight())) {
            continue;
        }
        if (Arrays.equals(tx.getTxData().getAddress(), agent.getAgentAddress())) {
            theAgent = agent;
        }
```

```
        }
        if (null == theAgent) {
            Log.warn(PocConsensusErrorCode.AGENT_NOT_EXIST.getMsg());
            return false;
        }
        CoinData coinData = ConsensusTool.getStopAgentCoinData(theAgent, tx.getTime() +
PocConsensusConstant.RED_PUNISH_LOCK_TIME, tx.getBlockHeight());
        if (NulsContext.MAIN_NET_VERSION <= 1) {
            if (coinData.getTo().size() != tx.getCoinData().getTo().size()) {
                Log.warn(PocConsensusErrorCode.RED_CARD_VERIFICATION_FAILED.getMsg());
                return false;
            }
            for (int i = 0; i < coinData.getTo().size(); i++) {
                Coin coin1 = coinData.getTo().get(i);
                Coin coin2 = tx.getCoinData().getTo().get(i);
                if (!Arrays.equals(coin1.getOwner(), coin2.getOwner()) ||
!coin1.getNa().equals(coin2.getNa())) {
                    Log.warn(PocConsensusErrorCode.RED_CARD_VERIFICATION_FAILED.getMsg());
                    return false;
                }
            }
        } else if (!Arrays.equals(coinData.serialize(), tx.getCoinData().serialize())) {
            Log.error("++++++++++ RedPunish verification does not pass, redPunish type:{}, - hight:{},
- redPunish tx timestamp:{}", tx.getTxData().getReasonCode(), tx.getBlockHeight(), tx.getTime());
            return false;
        }
        Log.info("++++++++++ RedPunish verification passed, redPunish type:{}, - hight:{}, -
redPunish tx timestamp:{}", tx.getTxData().getReasonCode(), tx.getBlockHeight(), tx.getTime());
        return true;
    }

    public Result verifyAndAddBlock(Block block, boolean isDownload, boolean
isNeedCheckCoinBaseTx) {
        Result result = verifyBlock(block, isDownload, isNeedCheckCoinBaseTx);
        if (result.isSuccess()) {
            if (!addBlock(block)) {
                return Result.getFailed();
            }
        }
        return result;
    }
```

```
    public boolean rollback(Block block) {

        Block bestBlock = chain.getBestBlock();

        if (block == null || !block.getHeader().getHash().equals(bestBlock.getHeader().getHash())) {
            Log.warn("rollbackTransaction block is not best block");
            return false;
        }

        List<Block> blockList = chain.getBlockList();

        if (blockList == null || blockList.size() == 0) {
            return false;
        }
        if (blockList.size() <= 2) {
            addBlockInBlockList(blockList);
            if (blockList.size() > 0) {
                Block startBlock = blockList.get(0);
                if (startBlock != null && chain.getStartBlockHeader().getHeight() >
startBlock.getHeader().getHeight()) {
                    chain.setStartBlockHeader(startBlock.getHeader());
                }
            }
        }

        blockList.remove(blockList.size() - 1);

        List<BlockHeader> blockHeaderList = chain.getBlockHeaderList();

        chain.setEndBlockHeader(blockHeaderList.get(blockHeaderList.size() - 2));
        BlockHeader rollbackBlockHeader = blockHeaderList.remove(blockHeaderList.size() - 1);

        // update txs
        List<Agent> agentList = chain.getAgentList();
        List<Deposit> depositList = chain.getDepositList();
        List<PunishLogPo> yellowList = chain.getYellowPunishList();
        List<PunishLogPo> redPunishList = chain.getRedPunishList();

        long height = rollbackBlockHeader.getHeight();

        for (int i = agentList.size() - 1; i >= 0; i--) {
            Agent agent = agentList.get(i);
```

```
    if (agent.getDelHeight() == height) {
        agent.setDelHeight(-1L);
    }

    if (agent.getBlockHeight() == height) {
        agentList.remove(i);
    }
}

for (int i = depositList.size() - 1; i >= 0; i--) {
    Deposit deposit = depositList.get(i);

    if (deposit.getDelHeight() == height) {
        deposit.setDelHeight(-1L);
    }

    if (deposit.getBlockHeight() == height) {
        depositList.remove(i);
    }
}

for (int i = yellowList.size() - 1; i >= 0; i--) {
    PunishLogPo tempYellow = yellowList.get(i);
    if (tempYellow.getHeight() < height) {
        break;
    }
    if (tempYellow.getHeight() == height) {
        yellowList.remove(i);
    }
}

for (int i = redPunishList.size() - 1; i >= 0; i--) {
    PunishLogPo redPunish = redPunishList.get(i);
    if (redPunish.getHeight() < height) {
        break;
    }
    if (redPunish.getHeight() == height) {
        redPunishList.remove(i);
    }
}
```

```
        //
        roundManager.checkIsNeedReset();

        return true;
    }

    private void addBlockInBlockList(List<Block> blockList) {
        BlockService blockService = NulsContext.getServiceBean(BlockService.class);
        if (blockList.isEmpty()) {
            blockList.add(blockService.getBestBlock().getData());
        }
        while (blockList.size() < PocConsensusConstant.INIT_BLOCKS_COUNT) {
            Block preBlock = blockList.get(0);
            if (preBlock.getHeader().getHeight() == 0) {
                break;
            }
            blockList.add(0, blockService.getBlock(preBlock.getHeader().getPreHash()).getData());
        }
    }

    /**
     * Get the state of the complete chain after the combination of a chain and the current chain bifurcation point, that is, first obtain the bifurcation point between the bifurcation chain and the current chain.
     * Then create a brand new chain, copy all the states before the bifurcation point of the main chain to the brand new chain
     * <p>
     *
     *
     *
     * @return ChainContainer
     */
    public ChainContainer getBeforeTheForkChain(ChainContainer chainContainer) {

        Chain newChain = new Chain();
        newChain.setId(chainContainer.getChain().getId());
        newChain.setStartBlockHeader(chain.getStartBlockHeader());
        newChain.setEndBlockHeader(chain.getEndBlockHeader());
        newChain.setBlockHeaderList(new ArrayList<>(chain.getBlockHeaderList()));
        newChain.setBlockList(new ArrayList<>(chain.getBlockList()));

        if (chain.getAgentList() != null) {
```

```
    List<Agent> agentList = new ArrayList<>();

    for (Agent agent : chain.getAgentList()) {
        try {
            agentList.add(agent.clone());
        } catch (CloneNotSupportedException e) {
            Log.error(e);
        }
    }

    newChain.setAgentList(agentList);
}
if (chain.getDepositList() != null) {
    List<Deposit> depositList = new ArrayList<>();

    for (Deposit deposit : chain.getDepositList()) {
        try {
            depositList.add(deposit.clone());
        } catch (CloneNotSupportedException e) {
            Log.error(e);
        }
    }

    newChain.setDepositList(depositList);
}
if (chain.getYellowPunishList() != null) {
    newChain.setYellowPunishList(new ArrayList<>(chain.getYellowPunishList()));
}
if (chain.getRedPunishList() != null) {
    newChain.setRedPunishList(new ArrayList<>(chain.getRedPunishList()));
}
ChainContainer newChainContainer = new ChainContainer(newChain);

// Bifurcation
//
BlockHeader pointBlockHeader = chainContainer.getChain().getStartBlockHeader();

List<Block> blockList = newChain.getBlockList();
for (int i = blockList.size() - 1; i >= 0; i--) {
    Block block = blockList.get(i);
    if (pointBlockHeader.getPreHash().equals(block.getHeader().getHash())) {
        break;
```

```
            }
            newChainContainer.rollback(block);
        }

        newChainContainer.initRound();

        return newChainContainer;
    }

    /**
     * Get the block information of the current chain and branch chain after the cross point and combine them into a new branch chain
     * <p>
     *
     *
     * @return ChainContainer
     */
    public ChainContainer getAfterTheForkChain(ChainContainer chainContainer) {

        // Bifurcation
        //
        BlockHeader pointBlockHeader = chainContainer.getChain().getStartBlockHeader();

        Chain chain = new Chain();

        List<Block> blockList = getChain().getBlockList();

        boolean canAdd = false;
        for (int i = 0; i < blockList.size(); i++) {

            Block block = blockList.get(i);

            if (canAdd) {
                chain.getBlockList().add(block);
                chain.getBlockHeaderList().add(block.getHeader());
            }

            if (pointBlockHeader.getPreHash().equals(block.getHeader().getHash())) {
                canAdd = true;
                if (i + 1 < blockList.size()) {
                    chain.setStartBlockHeader(blockList.get(i + 1).getHeader());
                    chain.setEndBlockHeader(getChain().getEndBlockHeader());
```

```
                chain.setPreChainId(chainContainer.getChain().getId());
            }
            continue;
        }
    }
    return new ChainContainer(chain);
}

public MeetingRound getCurrentRound() {
    return roundManager.getCurrentRound();
}

public MeetingRound getOrResetCurrentRound() {
    return roundManager.resetRound(true);
}

public MeetingRound getOrResetCurrentRound(boolean isRealTime) {
    return roundManager.resetRound(isRealTime);
}

public MeetingRound initRound() {
    return roundManager.initRound();
}

public void clearRound(int count) {
    roundManager.clearRound(count);
}

public Block getBestBlock() {
    return chain.getBestBlock();
}

public Chain getChain() {
    return chain;
}

public void setChain(Chain chain) {
    this.chain = chain;
}

@Override
public boolean equals(Object obj) {
```

```
        if (obj == null || !(obj instanceof ChainContainer)) {
            return false;
        }
        ChainContainer other = (ChainContainer) obj;
        if (other.getChain() == null || this.chain == null) {
            return false;
        }
        return other.getChain().getId().equals(this.chain.getId());
    }

    public RoundManager getRoundManager() {
        return roundManager;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

13:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\container\TxContainer.java

```
 *
 */
package io.nuls.consensus.poc.container;

import io.nuls.kernel.model.Transaction;

import java.io.Serializable;

/**
 * @author ln
 */
public class TxContainer implements Serializable{

    private Transaction tx;
    private int packageCount;

    public TxContainer(Transaction tx) {
        this.tx = tx;
    }
```

```
    public TxContainer(Transaction tx, int packageCount) {
        this.tx = tx;
        this.packageCount = packageCount;
    }

    public Transaction getTx() {
        return tx;
    }

    public void setTx(Transaction tx) {
        this.tx = tx;
    }

    public int getPackageCount() {
        return packageCount;
    }

    public void setPackageCount(int packageCount) {
        this.packageCount = packageCount;
    }
}

14:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\context\ConsensusStatusContext.java
 */

package io.nuls.consensus.poc.context;

import io.nuls.consensus.poc.constant.ConsensusStatus;

/**
 * @author ln
 */
public class ConsensusStatusContext {

    private static ConsensusStatus consensusStatus;

    public static ConsensusStatus getConsensusStatus() {
        return consensusStatus;
    }

    public static void setConsensusStatus(ConsensusStatus consensusStatus) {
```

```
        ConsensusStatusContext.consensusStatus = consensusStatus;
    }

    public static boolean isRunning() {
        return consensusStatus == ConsensusStatus.RUNNING;
    }
}
```

15:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\context\PocConsensusContext.java

```
 */

package io.nuls.consensus.poc.context;

import io.nuls.consensus.poc.manager.ChainManager;

/**
 *
 * @author ln
 */
public class PocConsensusContext {

    private static ChainManager chainManager;

    public static ChainManager getChainManager() {
        return chainManager;
    }

    public static void setChainManager(ChainManager chainManager) {
        PocConsensusContext.chainManager = chainManager;
    }
}
```

16:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\locker\Lockers.java

```
 *
 */

package io.nuls.consensus.poc.locker;

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

```
/**
 *
 * @author ln
 */
public final class Lockers {

    public final static Lock ROUND_LOCK = new ReentrantLock();

    public final static Lock CHAIN_LOCK = new ReentrantLock();
}

17:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\manager\CacheManager.java
 *
 */

package io.nuls.consensus.poc.manager;

import io.nuls.consensus.poc.cache.CacheLoader;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.container.ChainContainer;
import io.nuls.consensus.poc.model.Chain;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.storage.service.PunishLogStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.Transaction;

import java.util.HashSet;
import java.util.List;

/**
 * @author ln
 */
```

```
public class CacheManager {

    private ChainManager chainManager;

    private CacheLoader cacheLoader = new CacheLoader();

    public CacheManager(ChainManager chainManager) {
        this.chainManager = chainManager;
    }

    public void load() throws NulsException {

        //load storage data to memory

        List<BlockHeader> blockHeaderList =
cacheLoader.loadBlockHeaders(PocConsensusConstant.INIT_HEADERS_OF_ROUND_COUNT)
;
        List<Block> blockList =
cacheLoader.loadBlocks(PocConsensusConstant.INIT_BLOCKS_COUNT);

        if (blockHeaderList == null || blockHeaderList.size() == 0 || blockList == null || blockList.size()
== 0) {
            Log.error("load cache error ,not find the block info!");
            throw new NulsRuntimeException(KernelErrorCode.DATA_ERROR);
        }
        List<Agent> agentList = cacheLoader.loadAgents();
        List<Deposit> depositList = cacheLoader.loadDepositList();
        List<PunishLogPo> allPunishList =
NulsContext.getServiceBean(PunishLogStorageService.class).getPunishList();
        List<PunishLogPo> yellowPunishList = cacheLoader.loadYellowPunishList(allPunishList,
PocConsensusConstant.INIT_HEADERS_OF_ROUND_COUNT);
        List<PunishLogPo> redPunishList = cacheLoader.loadRedPunishList(allPunishList);

        Chain masterChain = new Chain();

        masterChain.setBlockHeaderList(blockHeaderList);
        masterChain.setBlockList(blockList);

        masterChain.setStartBlockHeader(blockList.get(0).getHeader());
        masterChain.setEndBlockHeader(blockList.get(blockList.size() - 1).getHeader());
        masterChain.setAgentList(agentList);
        masterChain.setDepositList(depositList);
```

```
        masterChain.setYellowPunishList(yellowPunishList);
        masterChain.setRedPunishList(redPunishList);

        ChainContainer masterChainContainer = new ChainContainer(masterChain);

        chainManager.setMasterChain(masterChainContainer);

        chainManager.getMasterChain().initRound();
    }

    public void reload() throws NulsException {
        clear();
        load();
    }

    public void clear() {
        chainManager.clear();
    }

    public CacheLoader getCacheLoader() {
        return cacheLoader;
    }

    public void setCacheLoader(CacheLoader cacheLoader) {
        this.cacheLoader = cacheLoader;
    }
}

18:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\manager\ChainManager.java
 *
 */

package io.nuls.consensus.poc.manager;

import io.nuls.consensus.poc.container.ChainContainer;
import io.nuls.consensus.poc.model.Chain;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;

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

```
/**
 *
 * @author ln
 */
public class ChainManager {

    private ChainContainer masterChain;
    private List<ChainContainer> chains;
    private List<ChainContainer> orphanChains;

    public ChainManager() {
        chains = new ArrayList<>();
        orphanChains = new ArrayList<>();
    }

    public void newOrphanChain(Block block) {
        BlockHeader header = block.getHeader();

        Chain orphanChain = new Chain();
        orphanChain.setStartBlockHeader(header);
        orphanChain.setEndBlockHeader(header);
        orphanChain.getBlockHeaderList().add(header);
        orphanChain.getBlockList().add(block);

        ChainContainer orphanChainContainer = new ChainContainer(orphanChain);
        orphanChains.add(orphanChainContainer);
    }

    public boolean checkIsBeforeOrphanChainAndAdd(Block block) {
        BlockHeader header = block.getHeader();

        boolean success = false;
        for(ChainContainer chainContainer : orphanChains) {
            Chain chain = chainContainer.getChain();
            if(header.getHash().equals(chain.getStartBlockHeader().getPreHash())) {
                success = true;
                chain.setStartBlockHeader(header);
                chain.getBlockHeaderList().add(0, header);
                chain.getBlockList().add(0, block);
            }
        }
```

```
        return success;
    }

    public boolean checkIsAfterOrphanChainAndAdd(Block block) {
        BlockHeader header = block.getHeader();

        for(ChainContainer chainContainer : orphanChains) {
            Chain chain = chainContainer.getChain();
            if(header.getPreHash().equals(chain.getEndBlockHeader().getHash())) {
                chain.setEndBlockHeader(header);
                chain.getBlockHeaderList().add(header);
                chain.getBlockList().add(block);
                return true;
            }
        }
        return false;
    }

    public long getBestBlockHeight() {
        if(masterChain == null || masterChain.getChain() == null ||
masterChain.getChain().getEndBlockHeader() == null) {
            return 0L;
        }
        return masterChain.getChain().getEndBlockHeader().getHeight();
    }

    public void clear() {
        masterChain = null;
        chains.clear();
        orphanChains.clear();
    }

    public ChainContainer getMasterChain() {
        return masterChain;
    }

    public void setMasterChain(ChainContainer masterChain) {
        this.masterChain = masterChain;
    }

    public List<ChainContainer> getChains() {
        return chains;
```

```
    }

    public Block getBestBlock() {
        return masterChain.getBestBlock();
    }

    public List<ChainContainer> getOrphanChains() {
        return orphanChains;
    }
}

19:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\manager\RoundManager.java
 *
 */

package io.nuls.consensus.poc.manager;

import io.nuls.account.service.AccountService;
import io.nuls.consensus.poc.config.ConsensusConfig;
import io.nuls.consensus.poc.locker.Lockers;
import io.nuls.consensus.poc.model.*;
import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.protocol.constant.PunishType;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.core.tools.calc.DoubleUtils;
import io.nuls.core.tools.log.ConsensusLog;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.protocol.constant.ProtocolConstant;

import java.util.*;

/**
 * @author ln
 */
```

```
public class RoundManager {

    private AccountService accountService;

    private List<MeetingRound> roundList = new ArrayList<>();

    private Chain chain;

    public RoundManager(Chain chain) {
        this.chain = chain;
    }

    public MeetingRound getRoundByIndex(long roundIndex) {
        MeetingRound round = null;
        for (int i = roundList.size() - 1; i >= 0; i--) {
            round = roundList.get(i);
            if (round.getIndex() == roundIndex) {
                break;
            }
        }
        return round;
    }

    public void addRound(MeetingRound meetingRound) {
        roundList.add(meetingRound);
    }

    public void checkIsNeedReset() {
        if (roundList == null || roundList.size() == 0) {
            initRound();
        } else {
            MeetingRound lastRound = roundList.get(roundList.size() - 1);
            Block bestBlcok = chain.getBestBlock();
            BlockExtendsData blockRoundData = new
BlockExtendsData(bestBlcok.getHeader().getExtend());
            if (blockRoundData.getRoundIndex() < lastRound.getIndex()) {
                roundList.clear();
                initRound();
            }
        }
    }
```

```
public boolean clearRound(int count) {
    if (roundList.size() > count) {
        roundList = roundList.subList(roundList.size() - count, roundList.size());
        MeetingRound round = roundList.get(0);
        round.setPreRound(null);
    }
    return true;
}

public MeetingRound getCurrentRound() {
    Lockers.ROUND_LOCK.lock();
    try {
        if (roundList == null || roundList.size() == 0) {
            return null;
        }
        MeetingRound round = roundList.get(roundList.size() - 1);
        if (round.getPreRound() == null && roundList.size() >= 2) {
            round.setPreRound(roundList.get(roundList.size() - 2));
        }
        return round;
    } finally {
        Lockers.ROUND_LOCK.unlock();
    }
}

public MeetingRound initRound() {
    MeetingRound currentRound = resetRound(false);

    if (currentRound.getPreRound() == null) {

        BlockExtendsData extendsData = null;
        List<BlockHeader> blockHeaderList = chain.getBlockHeaderList();
        for (int i = blockHeaderList.size() - 1; i >= 0; i--) {
            BlockHeader blockHeader = blockHeaderList.get(i);
            extendsData = new BlockExtendsData(blockHeader.getExtend());
            if (extendsData.getRoundIndex() < currentRound.getIndex()) {
                break;
            }
        }
        MeetingRound preRound = getNextRound(extendsData, false);
        currentRound.setPreRound(preRound);
    }
```

```
        return currentRound;
    }

    public MeetingRound resetRound(boolean isRealTime) {
        Lockers.ROUND_LOCK.lock();
        try {

            MeetingRound round = getCurrentRound();

            if (isRealTime) {
                if (round == null || round.getEndTime() < TimeService.currentTimeMillis()) {
                    MeetingRound nextRound = getNextRound(null, true);
                    nextRound.setPreRound(round);
                    roundList.add(nextRound);
                    round = nextRound;
                }
                return round;
            }

            BlockExtendsData extendsData = new
BlockExtendsData(chain.getEndBlockHeader().getExtend());
            if (round != null && extendsData.getRoundIndex() == round.getIndex() &&
extendsData.getPackingIndexOfRound() != extendsData.getConsensusMemberCount()) {
                return round;
            }

            MeetingRound nextRound = getNextRound(extendsData, false);
            if (round != null && nextRound.getIndex() <= round.getIndex()) {
                return nextRound;
            }
            nextRound.setPreRound(round);
            roundList.add(nextRound);
            return nextRound;
        } finally {
            Lockers.ROUND_LOCK.unlock();
        }
    }

    public MeetingRound getNextRound(BlockExtendsData roundData, boolean isRealTime) {
        Lockers.ROUND_LOCK.lock();
        try {
```

```
            if (isRealTime && roundData == null) {
                return getNextRoudByRealTime();
            } else if (!isRealTime && roundData == null) {
                return getNextRoundByNotRealTime();
            } else {
                return getNextRoundByExpectedRound(roundData);
            }
        } finally {
            Lockers.ROUND_LOCK.unlock();
        }
    }

    private MeetingRound getNextRoudByRealTime() {

        BlockHeader bestBlockHeader = chain.getEndBlockHeader();

        BlockHeader startBlockHeader = bestBlockHeader;

        BlockExtendsData bestRoundData = new BlockExtendsData(bestBlockHeader.getExtend());

        if (startBlockHeader.getHeight() != 0L) {
            long roundIndex = bestRoundData.getRoundIndex();
            if (bestRoundData.getConsensusMemberCount() ==
bestRoundData.getPackingIndexOfRound() || TimeService.currentTimeMillis() >=
bestRoundData.getRoundEndTime()) {
                roundIndex += 1;
            }
            startBlockHeader = getFirstBlockHeightOfPreRoundByRoundIndex(roundIndex);
        }

        long nowTime = TimeService.currentTimeMillis();
        long index = 0L;
        long startTime = 0L;

        if (nowTime < bestRoundData.getRoundEndTime()) {
            index = bestRoundData.getRoundIndex();
            startTime = bestRoundData.getRoundStartTime();
        } else {
            long diffTime = nowTime - bestRoundData.getRoundEndTime();
            int diffRoundCount = (int) (diffTime / (bestRoundData.getConsensusMemberCount() *
ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND * 1000L));
            index = bestRoundData.getRoundIndex() + diffRoundCount + 1;
```

```
            startTime = bestRoundData.getRoundEndTime() + diffRoundCount *
bestRoundData.getConsensusMemberCount() *
ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND * 1000L;
        }
        return calculationRound(startBlockHeader, index, startTime);
    }

    private MeetingRound getNextRoundByNotRealTime() {
        BlockHeader bestBlockHeader = chain.getEndBlockHeader();
        BlockExtendsData extendsData = new BlockExtendsData(bestBlockHeader.getExtend());
        extendsData.setRoundStartTime(extendsData.getRoundEndTime());
        extendsData.setRoundIndex(extendsData.getRoundIndex() + 1);
        return getNextRoundByExpectedRound(extendsData);
    }

    private MeetingRound getNextRoundByExpectedRound(BlockExtendsData roundData) {
        BlockHeader startBlockHeader = chain.getEndBlockHeader();

        long roundIndex = roundData.getRoundIndex();
        long roundStartTime = roundData.getRoundStartTime();
        if (startBlockHeader.getHeight() != 0L) {
//            if(roundData.getConsensusMemberCount() == roundData.getPackingIndexOfRound()) {
//                roundIndex += 1;
//                roundStartTime = roundData.getRoundEndTime();
//            }
            startBlockHeader = getFirstBlockHeightOfPreRoundByRoundIndex(roundIndex);
        }

        return calculationRound(startBlockHeader, roundIndex, roundStartTime);
    }

    private MeetingRound calculationRound(BlockHeader startBlockHeader, long index, long
startTime) {

        MeetingRound round = new MeetingRound();

        round.setIndex(index);
        round.setStartTime(startTime);

        setMemberList(round, startBlockHeader);

        round.calcLocalPacker(getAccountService().getAccountList().getData());
```

```
        ConsensusLog.debug("\ncalculation||index:{},startTime:{},startHeight:{},hash:{}\n" +
round.toString() + "\n\n", index, startTime, startBlockHeader.getHeight(),
startBlockHeader.getHash());
        return round;
    }


    private void setMemberList(MeetingRound round, BlockHeader startBlockHeader) {

        List<MeetingMember> memberList = new ArrayList<>();
        double totalWeight = 0;

        for (byte[] address : ConsensusConfig.getSeedNodeList()) {
            MeetingMember member = new MeetingMember();
            member.setAgentAddress(address);
            member.setPackingAddress(address);
            member.setRewardAddress(address);
            member.setCreditVal(0);
            member.setRoundStartTime(round.getStartTime());
            memberList.add(member);
        }

        List<Deposit> depositTempList = new ArrayList<>();

        List<Agent> agentList = getAliveAgentList(startBlockHeader.getHeight());
        for (Agent agent : agentList) {

            MeetingMember member = new MeetingMember();
            member.setAgent(agent);
            member.setAgentHash(agent.getTxHash());
            member.setAgentAddress(agent.getAgentAddress());
            member.setRewardAddress(agent.getRewardAddress());
            member.setPackingAddress(agent.getPackingAddress());
            member.setOwnDeposit(agent.getDeposit());
            member.setCommissionRate(agent.getCommissionRate());
            member.setRoundStartTime(round.getStartTime());

            List<Deposit> cdlist = getDepositListByAgentId(agent.getTxHash(),
startBlockHeader.getHeight());
            for (Deposit dtx : cdlist) {
                member.setTotalDeposit(member.getTotalDeposit().add(dtx.getDeposit()));
```

```
                depositTempList.add(dtx);
            }
            member.setDepositList(cdlist);
            agent.setTotalDeposit(member.getTotalDeposit().getValue());
            boolean isItIn =
member.getTotalDeposit().isGreaterOrEquals(PocConsensusProtocolConstant.SUM_OF_DEPOS
IT_OF_AGENT_LOWER_LIMIT);
            if (isItIn) {
                member.setCreditVal(calcCreditVal(member, startBlockHeader));
                agent.setCreditVal(member.getRealCreditVal());
                totalWeight = DoubleUtils.sum(totalWeight,
DoubleUtils.mul(agent.getDeposit().getValue(), member.getCalcCreditVal()));
                totalWeight = DoubleUtils.sum(totalWeight,
DoubleUtils.mul(member.getTotalDeposit().getValue(), member.getCalcCreditVal()));
                memberList.add(member);
            }
        }

        Collections.sort(memberList);

        for (int i = 0; i < memberList.size(); i++) {
            MeetingMember member = memberList.get(i);
            member.setRoundIndex(round.getIndex());
            member.setPackingIndexOfRound(i + 1);
        }

        round.init(memberList);

        Collections.sort(depositTempList, new Comparator<Deposit>() {
            @Override
            public int compare(Deposit o1, Deposit o2) {
                return o1.getTxHash().getDigestHex().compareTo(o2.getTxHash().getDigestHex());
            }
        });
    }

    private List<Deposit> getDepositListByAgentId(NulsDigestData agentHash, long
startBlockHeight) {

        List<Deposit> depositList = chain.getDepositList();
        List<Deposit> resultList = new ArrayList<>();
```

```
        for (int i = depositList.size() - 1; i >= 0; i--) {
            Deposit deposit = depositList.get(i);
            if (deposit.getDelHeight() != -1L && deposit.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (deposit.getBlockHeight() > startBlockHeight || deposit.getBlockHeight() < 0L) {
                continue;
            }
            if (!deposit.getAgentHash().equals(agentHash)) {
                continue;
            }
            resultList.add(deposit);
        }

        return resultList;
    }

    private List<Agent> getAliveAgentList(long startBlockHeight) {
        List<Agent> resultList = new ArrayList<>();
        for (int i = chain.getAgentList().size() - 1; i >= 0; i--) {
            Agent agent = chain.getAgentList().get(i);
            if (agent.getDelHeight() != -1L && agent.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (agent.getBlockHeight() > startBlockHeight || agent.getBlockHeight() < 0L) {
                continue;
            }
            resultList.add(agent);
        }
        return resultList;
    }

    private double calcCreditVal(MeetingMember member, BlockHeader blockHeader) {

        BlockExtendsData roundData = new BlockExtendsData(blockHeader.getExtend());

        long roundStart = roundData.getRoundIndex() -
PocConsensusProtocolConstant.RANGE_OF_CAPACITY_COEFFICIENT;
        if (roundStart < 0) {
            roundStart = 0;
        }
        long blockCount = getBlockCountByAddress(member.getPackingAddress(), roundStart,
```

```
roundData.getRoundIndex() - 1);
        long sumRoundVal = getPunishCountByAddress(member.getAgentAddress(), roundStart,
roundData.getRoundIndex() - 1, PunishType.YELLOW.getCode());
        double ability = DoubleUtils.div(blockCount,
PocConsensusProtocolConstant.RANGE_OF_CAPACITY_COEFFICIENT);

        double penalty =
DoubleUtils.div(DoubleUtils.mul(PocConsensusProtocolConstant.CREDIT_MAGIC_NUM,
sumRoundVal),
DoubleUtils.mul(PocConsensusProtocolConstant.RANGE_OF_CAPACITY_COEFFICIENT, PocC
onsensusProtocolConstant.RANGE_OF_CAPACITY_COEFFICIENT));

        return DoubleUtils.round(DoubleUtils.sub(ability, penalty), 4);
    }

    private long getPunishCountByAddress(byte[] address, long roundStart, long roundEnd, int
code) {
        long count = 0;
        List<PunishLogPo> punishList = new ArrayList<>(chain.getYellowPunishList());

        if (code == PunishType.RED.getCode()) {
            punishList = chain.getRedPunishList();
        }

        for (int i = punishList.size() - 1; i >= 0; i--) {
            PunishLogPo punish = punishList.get(i);

            if (punish.getRoundIndex() > roundEnd) {
                continue;
            }
            if (punish.getRoundIndex() < roundStart) {
                break;
            }
            if (Arrays.equals(punish.getAddress(), address)) {
                count++;
            }
        }
        //aa+99aa-1100101
        //Each round of punishment is likely to contain a rounds punishment record, calculated from a
to a + 99 rounds of punishment record,
        // a round of punishment is likely to be punished in an address in a - 1 round not out of the
blocks,
```

```
        // lead to round up to 100 May be 101 punishment record, treatment here
        if (count > 100) {
            return 100;
        }
        return count;
    }

    private long getBlockCountByAddress(byte[] packingAddress, long roundStart, long roundEnd) {
        long count = 0;
        List<BlockHeader> blockHeaderList = chain.getBlockHeaderList();

        for (int i = blockHeaderList.size() - 1; i >= 0; i--) {
            BlockHeader blockHeader = blockHeaderList.get(i);
            BlockExtendsData roundData = new BlockExtendsData(blockHeader.getExtend());

            if (roundData.getRoundIndex() > roundEnd) {
                continue;
            }
            if (roundData.getRoundIndex() < roundStart) {
                break;
            }
            if (Arrays.equals(blockHeader.getPackingAddress(), packingAddress)) {
                count++;
            }
        }
        return count;
    }

    private BlockHeader getFirstBlockHeightOfPreRoundByRoundIndex(long roundIndex) {
        BlockHeader firstBlockHeader = null;
        long startRoundIndex = 0L;
        List<BlockHeader> blockHeaderList = chain.getBlockHeaderList();
        for (int i = blockHeaderList.size() - 1; i >= 0; i--) {
            BlockHeader blockHeader = blockHeaderList.get(i);
            long currentRoundIndex = new
BlockExtendsData(blockHeader.getExtend()).getRoundIndex();
            if (roundIndex > currentRoundIndex) {
                if (startRoundIndex == 0L) {
                    startRoundIndex = currentRoundIndex;
                }
                if (currentRoundIndex < startRoundIndex) {
                    firstBlockHeader = blockHeaderList.get(i + 1);
```

```
                BlockExtendsData roundData = new
BlockExtendsData(firstBlockHeader.getExtend());
                if (roundData.getPackingIndexOfRound() > 1) {
                    firstBlockHeader = blockHeader;
                }
                break;
            }
        }
    }
    if (firstBlockHeader == null) {
        firstBlockHeader = chain.getStartBlockHeader();
        Log.warn("the first block of pre round not found");
    }
    return firstBlockHeader;
}

public Chain getChain() {
    return chain;
}

public List<MeetingRound> getRoundList() {
    return roundList;
}

public AccountService getAccountService() {
    if (accountService == null) {
        accountService = NulsContext.getServiceBean(AccountService.class);
    }
    return accountService;
}
}

20:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\model\BlockData.java
 *
 */
package io.nuls.consensus.poc.model;

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;

import java.util.List;
```

```
/**
 * @author Niels
 */
public class BlockData {
    private long height;
    private NulsDigestData preHash;
    private List<Transaction> txList;
    private BlockExtendsData extendsData;
    private long time;
    /**
     * pierre add
     */
    private byte[] stateRoot;

    public long getHeight() {
        return height;
    }

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

    public NulsDigestData getPreHash() {
        return preHash;
    }

    public void setPreHash(NulsDigestData preHash) {
        this.preHash = preHash;
    }

    public List<Transaction> getTxList() {
        return txList;
    }

    public void setTxList(List<Transaction> txList) {
        this.txList = txList;
    }

    public long getTime() {
        return time;
    }
```

```
    public void setTime(long time) {
        this.time = time;
    }

    public byte[] getStateRoot() {
        return stateRoot;
    }

    public void setStateRoot(byte[] stateRoot) {
        this.stateRoot = stateRoot;
    }

    public BlockExtendsData getExtendsData() {
        return extendsData;
    }

    public void setExtendsData(BlockExtendsData extendsData) {
        this.extendsData = extendsData;
    }
}
```

21:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\model\BlockExtendsData.java

```
 *
 */
package io.nuls.consensus.poc.model;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.constant.ProtocolConstant;

import java.io.IOException;

/**
 * @author Niels
 */
public class BlockExtendsData extends BaseNulsData {
```

```
    protected long roundIndex;

    protected int consensusMemberCount;

    protected long roundStartTime;

    protected int packingIndexOfRound;

    private Integer mainVersion;

    private Integer currentVersion;

    private Integer percent;

    private Long delay;

    private byte[] stateRoot;



    public long getRoundEndTime() {
        return roundStartTime + consensusMemberCount *
ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND * 1000L;
    }

    public BlockExtendsData() {
    }

    public BlockExtendsData(byte[] extend) {
        try {
            this.parse(extend, 0);
        } catch (NulsException e) {
            Log.error(e);
        }
    }

    public int getConsensusMemberCount() {
        return consensusMemberCount;
    }

    public void setConsensusMemberCount(int consensusMemberCount) {
```

```
    this.consensusMemberCount = consensusMemberCount;
}

public long getRoundStartTime() {
    return roundStartTime;
}

public void setRoundStartTime(long roundStartTime) {
    this.roundStartTime = roundStartTime;
}

public int getPackingIndexOfRound() {
    return packingIndexOfRound;
}

public void setPackingIndexOfRound(int packingIndexOfRound) {
    this.packingIndexOfRound = packingIndexOfRound;
}

public long getRoundIndex() {
    return roundIndex;
}

public void setRoundIndex(long roundIndex) {
    this.roundIndex = roundIndex;
}


@Override
public int size() {
    int size = 0;
    size += SerializeUtils.sizeOfUint32();  // roundIndex
    size += SerializeUtils.sizeOfUint16();  // consensusMemberCount
    size += SerializeUtils.sizeOfUint48();  // roundStartTime
    size += SerializeUtils.sizeOfUint16();  // packingIndexOfRound
    if (currentVersion != null) {
        size += SerializeUtils.sizeOfUint32();  // mainVersion
        size += SerializeUtils.sizeOfUint32();  // currentVersion
        size += SerializeUtils.sizeOfUint16();  // percent;
        size += SerializeUtils.sizeOfUint32();  // delay;
        size += SerializeUtils.sizeOfBytes(stateRoot);
    }
```

```
    return size;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeUint32(roundIndex);
    stream.writeUint16(consensusMemberCount);
    stream.writeUint48(roundStartTime);
    stream.writeUint16(packingIndexOfRound);
    if (currentVersion != null) {
        stream.writeUint32(mainVersion);
        stream.writeUint32(currentVersion);
        stream.writeUint16(percent);
        stream.writeUint32(delay);
        stream.writeBytesWithLength(stateRoot);
    }
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.roundIndex = byteBuffer.readUint32();
    this.consensusMemberCount = byteBuffer.readUint16();
    this.roundStartTime = byteBuffer.readUint48();
    this.packingIndexOfRound = byteBuffer.readUint16();
    if (!byteBuffer.isFinished()) {
        this.mainVersion = byteBuffer.readInt32();
        this.currentVersion = byteBuffer.readInt32();
        this.percent = byteBuffer.readUint16();
        this.delay = byteBuffer.readUint32();
        this.stateRoot = byteBuffer.readByLengthByte();
    }
}

public Integer getMainVersion() {
    return mainVersion;
}

public void setMainVersion(Integer mainVersion) {
    this.mainVersion = mainVersion;
}

public Integer getCurrentVersion() {
```

```
        return currentVersion;
    }

    public void setCurrentVersion(Integer currentVersion) {
        this.currentVersion = currentVersion;
    }

    public Integer getPercent() {
        return percent;
    }

    public void setPercent(Integer percent) {
        this.percent = percent;
    }

    public Long getDelay() {
        return delay;
    }

    public void setDelay(Long delay) {
        this.delay = delay;
    }

    public String getProtocolKey() {
        if (currentVersion != null && currentVersion > 1) {
            return this.currentVersion + "-" + this.percent + "-" + this.delay;
        }
        return null;
    }

    public byte[] getStateRoot() {
        return stateRoot;
    }

    public void setStateRoot(byte[] stateRoot) {
        this.stateRoot = stateRoot;
    }
}

22:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\model\BlockRoundData.java
 *
```

```
 */
package io.nuls.consensus.poc.model;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.kernel.utils.VarInt;
import io.nuls.protocol.constant.ProtocolConstant;
import io.protostuff.Tag;

import java.io.IOException;

/**
 * @author Niels
 */
public class BlockRoundData extends BaseNulsData {


    protected long roundIndex;

    protected int consensusMemberCount;

    protected long roundStartTime;

    protected int packingIndexOfRound;

    public long getRoundEndTime() {
        return roundStartTime + consensusMemberCount *
ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND * 1000L;
    }

    public BlockRoundData() {
    }

    public BlockRoundData(byte[] extend) {
        try {
            this.parse(extend,0);
        } catch (NulsException e) {
            Log.error(e);
```

```
    }
}

public int getConsensusMemberCount() {
    return consensusMemberCount;
}

public void setConsensusMemberCount(int consensusMemberCount) {
    this.consensusMemberCount = consensusMemberCount;
}

public long getRoundStartTime() {
    return roundStartTime;
}

public void setRoundStartTime(long roundStartTime) {
    this.roundStartTime = roundStartTime;
}

public int getPackingIndexOfRound() {
    return packingIndexOfRound;
}

public void setPackingIndexOfRound(int packingIndexOfRound) {
    this.packingIndexOfRound = packingIndexOfRound;
}

public long getRoundIndex() {
    return roundIndex;
}

public void setRoundIndex(long roundIndex) {
    this.roundIndex = roundIndex;
}


@Override
public int size() {
    int size = 0;
    size += SerializeUtils.sizeOfUint32(); // roundIndex
    size += SerializeUtils.sizeOfUint16(); // consensusMemberCount
    size += SerializeUtils.sizeOfUint48();  //roundStartTime
```

```
        size += SerializeUtils.sizeOfUint16();  // packingIndexOfRound
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeUint32(roundIndex);
        stream.writeUint16(consensusMemberCount);
        stream.writeUint48(roundStartTime);
        stream.writeUint16(packingIndexOfRound);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.roundIndex = byteBuffer.readUint32();
        this.consensusMemberCount = byteBuffer.readUint16();
        this.roundStartTime = byteBuffer.readUint48();
        this.packingIndexOfRound = byteBuffer.readUint16();
    }
}
```

23:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\model\Chain.java

```
 *
 */

package io.nuls.consensus.poc.model;

import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.Transaction;

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

/**
```

```
 * @author ln
 */
public class Chain implements Cloneable {

    private String id;
    private String preChainId;
    private BlockHeader startBlockHeader;
    private BlockHeader endBlockHeader;
    private List<BlockHeader> blockHeaderList;
    private List<Block> blockList;
    private List<Agent> agentList;
    private List<Deposit> depositList;
    private List<PunishLogPo> yellowPunishList;
    private List<PunishLogPo> redPunishList;

    public Chain() {
        blockHeaderList = new ArrayList<>();
        blockList = new ArrayList<>();
        id = StringUtils.getNewUUID();
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setPreChainId(String preChainId) {
        this.preChainId = preChainId;
    }

    public void setStartBlockHeader(BlockHeader startBlockHeader) {
        this.startBlockHeader = startBlockHeader;
    }

    public void setEndBlockHeader(BlockHeader endBlockHeader) {
        this.endBlockHeader = endBlockHeader;
    }

    public void setBlockHeaderList(List<BlockHeader> blockHeaderList) {
        this.blockHeaderList = blockHeaderList;
    }

    public String getId() {
```

```
    return id;
}

public String getPreChainId() {
    return preChainId;
}

public BlockHeader getStartBlockHeader() {
    return startBlockHeader;
}

public BlockHeader getEndBlockHeader() {
    return endBlockHeader;
}

public List<BlockHeader> getBlockHeaderList() {
    return blockHeaderList;
}

public List<Block> getBlockList() {
    return blockList;
}

public List<Agent> getAgentList() {
    return agentList;
}

public List<Deposit> getDepositList() {
    return depositList;
}

public void setBlockList(List<Block> blockList) {
    this.blockList = blockList;
}

public void setAgentList(List<Agent> agentList) {
    this.agentList = agentList;
}

public void setDepositList(List<Deposit> depositList) {
    this.depositList = depositList;
}
```

```
    public List<PunishLogPo> getYellowPunishList() {
        return yellowPunishList;
    }

    public void setYellowPunishList(List<PunishLogPo> yellowPunishList) {
        this.yellowPunishList = yellowPunishList;
    }

    public List<PunishLogPo> getRedPunishList() {
        return redPunishList;
    }

    public void setRedPunishList(List<PunishLogPo> redPunishList) {
        this.redPunishList = redPunishList;
    }

    public Agent getAgentByAddress(byte[] address) {
        for (Agent agent : agentList) {
            if (agent.getDelHeight() > 0) {
                continue;
            }
            if (ArraysTool.arrayEquals(agent.getAgentAddress(), address)) {
                return agent;
            }
        }
        return null;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    public Block getBestBlock() {
        if (blockList == null || blockList.size() == 0) {
            return null;
        }
        return blockList.get(blockList.size() - 1);
    }
}
```

24:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\model\Evidence.java
 */
```
package io.nuls.consensus.poc.model;

import io.nuls.consensus.poc.storage.po.EvidencePo;
import io.nuls.kernel.model.BlockHeader;

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

/**
 *
 * @author: Charlie
 * @date: 2018/9/4
 */
public class Evidence {
    private long roundIndex;
    private BlockHeader blockHeader1;
    private BlockHeader blockHeader2;

    public Evidence(long roundIndex, BlockHeader blockHeader1, BlockHeader blockHeader2){
        this.roundIndex = roundIndex;
        this.blockHeader1 = blockHeader1;
        this.blockHeader2 = blockHeader2;
    }
    public Evidence(EvidencePo evidencePo){
        this.roundIndex = evidencePo.getRoundIndex();
        this.blockHeader1 = evidencePo.getBlockHeader1();
        this.blockHeader2 = evidencePo.getBlockHeader2();
    }

    public EvidencePo toEvidencePo(){
        return new EvidencePo(this.roundIndex, this.blockHeader1, this.blockHeader2);
    }

    public static Map<String, List<EvidencePo>> bifurcationEvidenceMapToPoMap(Map<String, List<Evidence>> map){

        Map<String, List<EvidencePo>> poMap = new HashMap<>();
```

```
        for(Map.Entry<String, List<Evidence>> entry : map.entrySet()){
            List<EvidencePo> list = new ArrayList<>();
            for(Evidence evidence : entry.getValue()){
                list.add(evidence.toEvidencePo());
            }
            poMap.put(entry.getKey(), list);
        }

        return poMap;
    }

    public static Map<String, List<Evidence>> bifurcationEvidencePoMapToMap(Map<String,
List<EvidencePo>> poMap){

        Map<String, List<Evidence>> map = new HashMap<>(poMap.size());
        for(Map.Entry<String, List<EvidencePo>> entry : poMap.entrySet()){
            List<Evidence> list = new ArrayList<>();
            for(EvidencePo evidencePo : entry.getValue()){
                list.add(new Evidence(evidencePo));
            }
            map.put(entry.getKey(), list);
        }
        return map;
    }



    public long getRoundIndex() {
        return roundIndex;
    }

    public void setRoundIndex(long roundIndex) {
        this.roundIndex = roundIndex;
    }

    public BlockHeader getBlockHeader1() {
        return blockHeader1;
    }

    public void setBlockHeader1(BlockHeader blockHeader1) {
        this.blockHeader1 = blockHeader1;
    }
```

```
    public BlockHeader getBlockHeader2() {
        return blockHeader2;
    }

    public void setBlockHeader2(BlockHeader blockHeader2) {
        this.blockHeader2 = blockHeader2;
    }
}

25:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\model\MeetingMember.java
 *
 */
package io.nuls.consensus.poc.model;

import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.crypto.Sha256Hash;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.SerializeUtils;

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

/**
 * @author Niels
 */
public class MeetingMember implements Comparable<MeetingMember> {
    private long roundIndex;
    private long roundStartTime;
    private byte[] agentAddress;
    private byte[] packingAddress;
    private byte[] rewardAddress;
    private NulsDigestData agentHash;
    /**
     * Starting from 1
     */
    private int packingIndexOfRound;
    private double creditVal;
    private Agent agent;
```

```
    private List<Deposit> depositList = new ArrayList<>();
    private Na totalDeposit = Na.ZERO;
    private Na ownDeposit = Na.ZERO;
    private double commissionRate;
    private String sortValue;
    private long packStartTime;
    private long packEndTime;

    public Na getTotalDeposit() {
        return totalDeposit;
    }

    public void setTotalDeposit(Na totalDeposit) {
        this.totalDeposit = totalDeposit;
    }

    public String getSortValue() {
        if (this.sortValue == null) {
            byte[] hash = ArraysTool.concatenate(packingAddress,
SerializeUtils.uint64ToByteArray(roundStartTime));
            sortValue = Sha256Hash.twiceOf(hash).toString();
        }
        return sortValue;
    }

    public long getRoundStartTime() {
        return roundStartTime;
    }

    public void setRoundStartTime(long roundStartTime) {
        this.roundStartTime = roundStartTime;
        this.sortValue = null;
    }

    public int getPackingIndexOfRound() {
        return packingIndexOfRound;
    }

    public void setPackingIndexOfRound(int packingIndexOfRound) {
        this.packingIndexOfRound = packingIndexOfRound;
    }
```

```
public long getPackStartTime() {
    return packStartTime;
}

public void setPackStartTime(long packStartTime) {
    this.packStartTime = packStartTime;
}

public long getPackEndTime() {
    return packEndTime;
}

public void setPackEndTime(long packEndTime) {
    this.packEndTime = packEndTime;
}

public long getRoundIndex() {
    return roundIndex;
}

public void setRoundIndex(long roundIndex) {
    this.roundIndex = roundIndex;
}

public double getRealCreditVal() {
    return creditVal;
}

public double getCalcCreditVal() {
    return creditVal < 0d ? 0D : this.creditVal;
}

public void setCreditVal(double creditVal) {
    this.creditVal = creditVal;
}

public Na getOwnDeposit() {
    return ownDeposit;
}

public void setOwnDeposit(Na ownDeposit) {
    this.ownDeposit = ownDeposit;
```

```
}

@Override
public int compareTo(MeetingMember o2) {
    return this.getSortValue().compareTo(o2.getSortValue());
}

public double getCommissionRate() {
    return commissionRate;
}

public void setCommissionRate(double commissionRate) {
    this.commissionRate = commissionRate;
}

public byte[] getAgentAddress() {
    return agentAddress;
}

public void setAgentAddress(byte[] agentAddress) {
    this.agentAddress = agentAddress;
}

public byte[] getPackingAddress() {
    return packingAddress;
}

public void setPackingAddress(byte[] packingAddress) {
    this.packingAddress = packingAddress;
}

public NulsDigestData getAgentHash() {
    return agentHash;
}

public void setAgentHash(NulsDigestData agentHash) {
    this.agentHash = agentHash;
}

public double getCreditVal() {
    return creditVal;
}
```

```
    public Agent getAgent() {
        return agent;
    }

    public void setAgent(Agent agent) {
        this.agent = agent;
    }

    public List<Deposit> getDepositList() {
        return depositList;
    }

    public void setDepositList(List<Deposit> depositList) {
        this.depositList = depositList;
    }

    public void setSortValue(String sortValue) {
        this.sortValue = sortValue;
    }

    public byte[] getRewardAddress() {
        return rewardAddress;
    }

    public void setRewardAddress(byte[] rewardAddress) {
        this.rewardAddress = rewardAddress;
    }
}
```

26:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\model\MeetingRound.java

```
 *
 */
package io.nuls.consensus.poc.model;

import io.nuls.account.model.Account;
import io.nuls.kernel.model.Address;
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.core.tools.calc.DoubleUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsRuntimeException;
```

```
import io.nuls.kernel.func.TimeService;
import io.nuls.protocol.constant.ProtocolConstant;

import java.util.*;

/**
 * @author Niels
 */
public class MeetingRound {

    private Account localPacker;
    private double totalWeight;
    private long index;
    private long startTime;
    private long endTime;
    private int memberCount;
    private List<MeetingMember> memberList;
    private MeetingRound preRound;
    private MeetingMember myMember;

    public MeetingRound getPreRound() {
        return preRound;
    }

    public void setPreRound(MeetingRound preRound) {
        this.preRound = preRound;
    }

    public long getStartTime() {
        return startTime;
    }

    public void setStartTime(long startTime) {
        this.startTime = startTime;
    }

    public long getEndTime() {
        return endTime;
    }

    public int getMemberCount() {
        return memberCount;
```

```
    }

    public void init(List<MeetingMember> memberList) {

        assert (startTime > 0L);

        this.memberList = memberList;
        if (null == memberList || memberList.isEmpty()) {
            throw new NulsRuntimeException(KernelErrorCode.DATA_ERROR);
        }

        Collections.sort(memberList);

        this.memberCount = memberList.size();
        totalWeight = 0d;
        for (int i = 0; i < memberList.size(); i++) {
            MeetingMember member = memberList.get(i);
            member.setRoundIndex(this.getIndex());
            member.setRoundStartTime(this.getStartTime());
            member.setPackingIndexOfRound(i + 1);
            member.setPackStartTime(startTime + i *
ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS);
            member.setPackEndTime(member.getPackStartTime() +
ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS);
            totalWeight +=DoubleUtils.mul( DoubleUtils.sum(member.getTotalDeposit().getValue() ,
member.getOwnDeposit().getValue()),member.getCalcCreditVal()) ;
        }
        endTime = startTime + memberCount * ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS;
    }

    public MeetingMember getMember(int order) {
        if (order == 0) {
            throw new NulsRuntimeException(KernelErrorCode.DATA_ERROR);
        }
        if (null == memberList || memberList.isEmpty()) {
            throw new NulsRuntimeException(KernelErrorCode.DATA_ERROR);
        }
        return this.memberList.get(order - 1);
    }

    public MeetingMember getMember(byte[] address) {
        for (MeetingMember member : memberList) {
```

```
        if (Arrays.equals(address, member.getPackingAddress())) {
            return member;
        }
    }
    return null;
}
public MeetingMember getMemberByAgentAddress(byte[] address) {
    for (MeetingMember member : memberList) {
        if (Arrays.equals(address, member.getAgentAddress())) {
            return member;
        }
    }
    return null;
}

public Account getLocalPacker() {
    return localPacker;
}

public long getIndex() {
    return index;
}

public void setIndex(long index) {
    this.index = index;
}


public double getTotalWeight() {
    return totalWeight;
}

public List<MeetingMember> getMemberList() {
    return memberList;
}

public MeetingMember getMyMember() {
    return myMember;
}

public void calcLocalPacker(Collection<Account> accountList) {
    for (Account account : accountList) {
```

```
            if(account.isEncrypted()) {
                continue;
            }
            MeetingMember member = getMember(account.getAddress().getAddressBytes());
            if (null != member) {
                this.localPacker = account;
                myMember = member;
                return;
            }
        }
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();
        for (MeetingMember member : this.getMemberList()) {
            str.append(Address.fromHashs(member.getPackingAddress()).getBase58());
            str.append(" ,order:" + member.getPackingIndexOfRound());
            str.append(",packTime:" + new Date(member.getPackEndTime()));
            str.append(",creditVal:" + member.getRealCreditVal());
            str.append("\n");
        }
        if (null == this.getPreRound()) {
            return ("round:index:" + this.getIndex() + " , start:" + new Date(this.getStartTime())
                    + ", netTime:(" + new Date(TimeService.currentTimeMillis()).toString() + ") ,
totalWeight : " + totalWeight + " ,members:\n :" + str);
        } else {
            return ("round:index:" + this.getIndex() + " ,preIndex:" + this.getPreRound().getIndex() + " ,
start:" + new Date(this.getStartTime())
                    + ", netTime:(" + new Date(TimeService.currentTimeMillis()).toString() + ") ,
totalWeight : " + totalWeight + "  , members:\n :" + str);
        }
    }
}

27:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\model\RewardItem.java
 */

package io.nuls.consensus.poc.model;

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

```
/**
 * @author: Niels Wang
 */
public class RewardItem {
    private long time;
    private Na na;

    public RewardItem(long time, Na na) {
        this.time = time;
        this.na = na;

    }

    public long getTime() {
        return time;
    }

    public void setTime(long time) {
        this.time = time;
    }

    public Na getNa() {
        return na;
    }

    public void setNa(Na na) {
        this.na = na;
    }

}

28:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\model\RewardStatisticsParam.java
 */

package io.nuls.consensus.poc.model;

import io.nuls.kernel.model.Block;

/**
 * @author: Niels Wang
```

```
 */
public class RewardStatisticsParam {

    private int type;

    private Block block;

    public RewardStatisticsParam(int type, Block block) {
        this.type = type;
        this.block = block;

    }

    public int getType() {
        return type;
    }

    public void setType(int type) {
        this.type = type;
    }

    public Block getBlock() {
        return block;
    }

    public void setBlock(Block block) {
        this.block = block;
    }
}
```

29:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\module\impl\PocConsensusModuleBootstrap.java

```
 *
 */
package io.nuls.consensus.poc.module.impl;

import io.nuls.consensus.module.AbstractConsensusModule;
import io.nuls.consensus.poc.block.validator.BifurcationUtil;
import io.nuls.consensus.poc.config.ConsensusConfig;
import io.nuls.consensus.poc.constant.ConsensusStatus;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.consensus.poc.model.Evidence;
```

```
import io.nuls.consensus.poc.process.NulsProtocolProcess;
import io.nuls.consensus.poc.scheduler.ConsensusScheduler;
import io.nuls.consensus.poc.storage.po.EvidencePo;
import io.nuls.consensus.poc.storage.service.BifurcationEvidenceStorageService;
import io.nuls.consensus.poc.util.ProtocolTransferTool;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.ModuleStatusEnum;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.thread.BaseThread;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.base.version.ProtocolContainer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.storage.po.BlockProtocolInfoPo;
import io.nuls.protocol.storage.service.VersionManagerStorageService;


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

/**
 * @author Niels
 */
public class PocConsensusModuleBootstrap extends AbstractConsensusModule {

    private boolean protocolInited;

    private VersionManagerStorageService versionManagerStorageService;

    @Override
    public void init() throws Exception {
        ConsensusStatusContext.setConsensusStatus(ConsensusStatus.INITING);
        ConsensusConfig.initConfiguration();
        BifurcationEvidenceStorageService bes =
NulsContext.getServiceBean(BifurcationEvidenceStorageService.class);
```

```
        Map<String, List<EvidencePo>> map = bes.getBifurcationEvidence();
        if (null != map) {
BifurcationUtil.getInstance().setBifurcationEvidenceMap(Evidence.bifurcationEvidencePoMapToMap(map));
        }
    }

    @Override
    public void start() {
        this.waitForDependencyRunning(ProtocolConstant.MODULE_ID_PROTOCOL);
        if (!protocolInited) {
            protocolInited = true;
            initNulsProtocol();
        }
        ConsensusScheduler.getInstance().start();
        this.registerHandlers();
        Log.info("the POC consensus module is started!");
    }

    private void initNulsProtocol() {
        try {
            //
            NulsVersionManager.init();
            BlockService blockService = NulsContext.getServiceBean(BlockService.class);
            if (NulsContext.MAIN_NET_VERSION == 1 &&
NulsContext.CURRENT_PROTOCOL_VERSION == 2) {

                long bestHeight = blockService.getBestBlockHeader().getData().getHeight();
                Long consensusVersionHeight =
getVersionManagerStorageService().getConsensusVersionHeight();
                if (consensusVersionHeight == null) {
//                    consensusVersionHeight = 680000L;
                    consensusVersionHeight = 1L;
                } else {
                    long height = consensusVersionHeight + 1;
                    BlockProtocolInfoPo infoPo = null;
                    while (true) {
                        height--;
//                        if(height == 680000L) {
//                            break;
//                        }
                        if(height <= 0) {
```

```
                    break;
                }
                infoPo = getVersionManagerStorageService().getBlockProtocolInfoPo(height);
                if (infoPo != null) {
                    break;
                }
            }

            if (infoPo != null) {
                ProtocolContainer container =
NulsVersionManager.getProtocolContainer(infoPo.getVersion());
                ProtocolTransferTool.copyFromBlockProtocolInfoPo(infoPo, container);
            }
        }
        for (long i = consensusVersionHeight; i <= bestHeight; i++) {
            Result<BlockHeader> result = blockService.getBlockHeader(i);
            if (result.isSuccess()) {
                NulsProtocolProcess.getInstance().processProtocolUpGrade(result.getData());
            }
        }
    } else {
        NulsVersionManager.loadVersion();
    }

} catch (Exception e) {
    Log.error(e);
    System.exit(-1);
}
}

private void registerHandlers() {

}

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

@Override
public void destroy() {
```

```
    }

    @Override
    public String getInfo() {
        if (this.getStatus() == ModuleStatusEnum.UNINITIALIZED || this.getStatus() ==
ModuleStatusEnum.INITIALIZING) {
            return "";
        }
        StringBuilder str = new StringBuilder();
        str.append("module:[consensus]:\n");
        str.append("thread count:");
        List<BaseThread> threadList = TaskManager.getThreadList(this.getModuleId());
        if (null == threadList) {
            str.append(0);
        } else {
            str.append(threadList.size());
            for (BaseThread thread : threadList) {
                str.append("\n");
                str.append(thread.getName());
                str.append("{");
                str.append(thread.getPoolName());
                str.append("}");
            }
        }
        return str.toString();
    }

    private VersionManagerStorageService getVersionManagerStorageService() {
        if (versionManagerStorageService == null) {
            versionManagerStorageService =
NulsContext.getServiceBean(VersionManagerStorageService.class);
        }
        return versionManagerStorageService;
    }

}

30:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\process\BlockMonitorProcess.java
 */
```

```
package io.nuls.consensus.poc.process;

import io.nuls.consensus.poc.config.ConsensusConfig;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.manager.ChainManager;
import io.nuls.consensus.poc.service.impl.ConsensusPocServiceImpl;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.protocol.service.DownloadService;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * @author Niels
 */
public class BlockMonitorProcess {

    private final static long RESET_TIME_INTERVAL =
PocConsensusConstant.RESET_SYSTEM_TIME_INTERVAL * 60 * 1000L;

    private final ChainManager chainManager;

    public BlockMonitorProcess(ChainManager chainManager) {
        this.chainManager = chainManager;
    }

    private NulsDigestData lastBestHash;

    public void doProcess() {
        Block bestBlock = NulsContext.getInstance().getBestBlock();
        if (bestBlock.getHeader().getHeight() == 0) {
            return;
        }
        if (bestBlock.getHeader().getHash().equals(lastBestHash) &&
bestBlock.getHeader().getTime() < (TimeService.currentTimeMillis() - RESET_TIME_INTERVAL))
{
```

```
            lastBestHash = bestBlock.getHeader().getHash();
            NulsContext.getServiceBean(ConsensusPocServiceImpl.class).reset();
            return;
        }
        lastBestHash = bestBlock.getHeader().getHash();
        List<Block> blockList = chainManager.getMasterChain().getChain().getBlockList();
        int minCount = 10;
        if (blockList.size() < minCount) {
            return;
        }
        int count = 0;
        Set<String> addressSet = new HashSet<>();
        for (Block block : blockList) {
addressSet.add(AddressTool.getStringAddressByBytes(block.getHeader().getPackingAddress()));
           count++;
            if (count > minCount) {
                break;
            }
        }
        DownloadService downloadService = NulsContext.getServiceBean(DownloadService.class);
        if (count > minCount && addressSet.size() == 1 &&
ConsensusConfig.getSeedNodeList().size() > 1) {
            NulsContext.getServiceBean(ConsensusPocServiceImpl.class).reset();
            return;
        }
        if (downloadService.isDownloadSuccess().isSuccess() &&
                bestBlock.getHeader().getTime() < (TimeService.currentTimeMillis() -
RESET_TIME_INTERVAL)) {
            NulsContext.getServiceBean(ConsensusPocServiceImpl.class).reset();
        }
    }
}

31:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\process\BlockProcess.java
 *
 */

package io.nuls.consensus.poc.process;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.block.validator.BifurcationUtil;
```

```
import io.nuls.consensus.poc.cache.TxMemoryPool;
import io.nuls.consensus.poc.constant.BlockContainerStatus;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.container.BlockContainer;
import io.nuls.consensus.poc.container.ChainContainer;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.manager.ChainManager;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.model.Chain;
import io.nuls.consensus.poc.model.MeetingMember;
import io.nuls.consensus.poc.model.MeetingRound;
import io.nuls.consensus.poc.protocol.constant.PunishReasonEnum;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.provider.OrphanBlockProvider;
import io.nuls.consensus.poc.storage.service.TransactionCacheStorageService;
import io.nuls.consensus.poc.util.ConsensusTool;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.BlockLog;
import io.nuls.core.tools.log.ChainLog;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.*;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.cache.TemporaryCacheManager;
import io.nuls.protocol.model.SmallBlock;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.TransactionService;
```

```
import java.io.IOException;
import java.util.*;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Future;

/**
 * @author ln
 */
public class BlockProcess {

    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);

    private ChainManager chainManager;
    private OrphanBlockProvider orphanBlockProvider;
    private BifurcationUtil bifurcationUtil = BifurcationUtil.getInstance();

    private LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);
    private TransactionService tansactionService =
NulsContext.getServiceBean(TransactionService.class);
    private ContractService contractService = NulsContext.getServiceBean(ContractService.class);
    private TransactionCacheStorageService transactionCacheStorageService =
NulsContext.getServiceBean(TransactionCacheStorageService.class);

    private ExecutorService signExecutor =
TaskManager.createThreadPool(Runtime.getRuntime().availableProcessors(),
Integer.MAX_VALUE, new NulsThreadFactory(ConsensusConstant.MODULE_ID_CONSENSUS,
""));

    private NulsProtocolProcess nulsProtocolProcess = NulsProtocolProcess.getInstance();
    private TemporaryCacheManager cacheManager = TemporaryCacheManager.getInstance();

    public BlockProcess(ChainManager chainManager, OrphanBlockProvider orphanBlockProvider)
{
        this.chainManager = chainManager;
        this.orphanBlockProvider = orphanBlockProvider;
    }

    /**
     * Dealing with new blocks, the new block has two cases, the block when downloading and the
latest block received, there are two different authentication logic
```

```
    *      * The download block is added. The verification round is not the current round. You need to restore the block to generate the current round status.
    *      * The new block received during operation must be verified with the latest round status
    *      * New block processing flow:
    *      * 1. Preprocessing, basic verification, including verification of block header field information, block size verification, signature verification
    *      * 2, try to add blocks to the main chain, first verify the block of the round and packaged people, if the verification fails, then put into the isolated block pool, if the verification is successful, then add into the main chain, add the memory state into
    *      * 3, verify the transaction of the block is legitimate, whether there are double flowers or other illegal transactions, if there is, then put in the isolated block pool, if not, save the block
    *      * 4, save the block header information, save the block transaction information
    *      * 5. Forwarding block
    * <p>
    *
    *
    *
    *
    * 1
    * 2
    * 3
    * 4
    * 5
    *
    * @return boolean
    */
   public boolean addBlock(BlockContainer blockContainer) throws IOException {


        boolean isDownload = blockContainer.getStatus() ==
BlockContainerStatus.DOWNLOADING;
        Block block = blockContainer.getBlock();
        Log.info("******** BlockProcess - addBlock ******* height:{} , hash:{}, preHash:{}",
block.getHeader().getHeight(), block.getHeader().getHash(), block.getHeader().getPreHash());
        Log.info(" packingAddress:{}",
AddressTool.getStringAddressByBytes(block.getHeader().getPackingAddress()));
        // Discard future blocks
        //
        if (TimeService.currentTimeMillis() +
PocConsensusConstant.DISCARD_FUTURE_BLOCKS_TIME < block.getHeader().getTime()) {
            return false;
        }
```

```
        // Verify the the block, the content to be verified includes: whether the block size exceeds the limit,
        // whether the attribute of the block header is legal, the Merkel tree root is correct, the signature is correct,
        // and whether the expanded round of information is valid
        //
        block.verifyWithException();
        bifurcationUtil.validate(block.getHeader());

        ValidateResult<List<Transaction>> validateResult = ledgerService.verifyDoubleSpend(block);
        if (validateResult.isFailed() && validateResult.getErrorCode().equals(TransactionErrorCode.TRANSACTION_REPEATED)) {
            RedPunishTransaction redPunishTransaction = new RedPunishTransaction();
            RedPunishData redPunishData = new RedPunishData();
            byte[] packingAddress = AddressTool.getAddress(block.getHeader().getBlockSignature().getPublicKey());
            List<Agent> agentList = PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
            Agent agent = null;
            for (Agent a : agentList) {
                if (a.getDelHeight() > 0) {
                    continue;
                }
                if (Arrays.equals(a.getPackingAddress(), packingAddress)) {
                    agent = a;
                    break;
                }
            }
            if (null == agent) {
                return false;
            }
            redPunishData.setAddress(agent.getAgentAddress());
            SmallBlock smallBlock = new SmallBlock();
            smallBlock.setHeader(block.getHeader());
            smallBlock.setTxHashList(block.getTxHashList());
            for (Transaction tx : validateResult.getData()) {
                smallBlock.addBaseTx(tx);
            }
            redPunishData.setEvidence(smallBlock.serialize());
            redPunishData.setReasonCode(PunishReasonEnum.DOUBLE_SPEND.getCode());
            redPunishTransaction.setTxData(redPunishData);
```

```
            redPunishTransaction.setTime(smallBlock.getHeader().getTime());
            CoinData coinData = ConsensusTool.getStopAgentCoinData(agent,
redPunishTransaction.getTime() + PocConsensusConstant.RED_PUNISH_LOCK_TIME);
            redPunishTransaction.setCoinData(coinData);
redPunishTransaction.setHash(NulsDigestData.calcDigestData(redPunishTransaction.serializeFor
Hash()));
            TxMemoryPool.getInstance().add(redPunishTransaction, false);
            return false;
        }

        // Verify that the block round information is correct, if correct, join the main chain
        //
        Result verifyAndAddBlockResult =
chainManager.getMasterChain().verifyAndAddBlock(block, isDownload, false);
        if (verifyAndAddBlockResult.isSuccess()) {
            boolean success = true;
            try {
                do {
                    // Verify that the block transaction is valid, save the block if the verification passes,
and discard the block if it fails
                    //
                    long time = System.currentTimeMillis();
                    List<Future<Boolean>> futures = new ArrayList<>();

                    List<Transaction> txs = block.getTxs();

                    //
                    Set<Integer> txTypeSet =
NulsVersionManager.getMainProtocolContainer().getTxMap().keySet();
                    for (Transaction tx : txs) {
                        if (!txTypeSet.contains(tx.getType())) {
                            Log.info("---------------------- block tx discard, current protocol version:" +
NulsVersionManager.getMainProtocolContainer().getVersion() + ",  tx.type:" + tx.getType());
                            return false;
                        }
                    }

                    for (Transaction tx : txs) {
                        Future<Boolean> res = signExecutor.submit(new Callable<Boolean>() {
                            @Override
                            public Boolean call() throws Exception {
                                ValidateResult verify = tx.verify();
```

```
                /** ********************************************************/
                if (verify.isFailed()) {
                    Log.error(JSONUtils.obj2json(verify.getErrorCode()));
                }
                /** ********************************************************/
                boolean result = verify.isSuccess();
                return result;
            }
        });
        futures.add(res);
    }

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

    /**
     * pierre add
     */
    BlockHeader bestBlockHeader =
NulsContext.getInstance().getBestBlock().getHeader();
    BlockHeader verifyHeader = block.getHeader();
    long bestHeight = bestBlockHeader.getHeight();
    byte[] receiveStateRoot = ConsensusTool.getStateRoot(verifyHeader);
    byte[] stateRoot = ConsensusTool.getStateRoot(bestBlockHeader);
    ContractResult contractResult;
    Map<String, Coin> contractUsedCoinMap = new HashMap<>();
    int totalGasUsed = 0;

    //
    contractService.createContractTempBalance();
    //
    contractService.createBatchExecute(stateRoot);
    // ()
    BlockHeader tempHeader = new BlockHeader();
    tempHeader.setTime(verifyHeader.getTime());
    tempHeader.setHeight(verifyHeader.getHeight());
    tempHeader.setPackingAddress(verifyHeader.getPackingAddress());
    contractService.createCurrentBlockHeader(tempHeader);

    List<ContractResult> contractResultList = new ArrayList<>();
    // stateRoot,
    byte[] tempStateRoot = null;
```

```
            for (Transaction tx : txs) {

                if (tx.isSystemTx()) {
                    continue;
                }

                // GasGAS
                if (totalGasUsed > ContractConstant.MAX_PACKAGE_GAS) {
                    if (ContractUtil.isGasCostContractTransaction(tx)) {
                        Log.info("verify block failed: Excess contract transaction detected.");
                        success = false;
                        break;
                    }
                }

                ValidateResult result = ledgerService.verifyCoinData(tx, toMaps, fromSet);
                if (result.isFailed()) {
                    Log.info("failed message:" + result.getMsg());
                    success = false;
                    break;
                }

                //
                if (ContractUtil.isContractTransaction(tx)) {
                    contractResult = contractService.batchProcessTx(tx, bestHeight, block,
stateRoot, toMaps, contractUsedCoinMap, false).getData();
                    if (contractResult != null) {
                        tempStateRoot = contractResult.getStateRoot();
                        totalGasUsed += contractResult.getGasUsed();
                        contractResultList.add(contractResult);
                    }
                }

            }

            if (!success) {
                break;
            }

            //
            contractService.removeContractTempBalance();
```

```
                stateRoot = contractService.commitBatchExecute().getData();
                //
                contractService.removeBatchExecute();
                //
                contractService.removeCurrentBlockHeader();
                //
                if (tempStateRoot != null) {
                    stateRoot = tempStateRoot;
                }
                block.getHeader().setStateRoot(stateRoot);
                for (ContractResult result : contractResultList) {
                    result.setStateRoot(stateRoot);
                }

                //
                if ((receiveStateRoot != null || stateRoot != null) && !Arrays.equals(receiveStateRoot,
stateRoot)) {
                    Log.info("contract stateRoot incorrect. receiveStateRoot is {}, stateRoot is {}.",
receiveStateRoot != null ? Hex.encode(receiveStateRoot) : receiveStateRoot, stateRoot != null ?
Hex.encode(stateRoot) : stateRoot);
                    success = false;
                    break;
                }

                // CoinBase
                Object[] objects = (Object[]) verifyAndAddBlockResult.getData();
                MeetingRound currentRound = (MeetingRound) objects[0];
                MeetingMember member = (MeetingMember) objects[1];
                if (!chainManager.getMasterChain().verifyCoinBaseTx(block, currentRound, member))
{
                    success = false;
                    break;
                }

                if (!success) {
                    break;
                }

                ValidateResult validateResult1 = tansactionService.conflictDetect(block.getTxs());
                if (validateResult1.isFailed()) {
                    success = false;
                    Log.info("failed message:" + validateResult1.getMsg());
```

```
                    break;
                }

                for (Future<Boolean> future : futures) {
                    if (!future.get()) {
                        success = false;
                        Log.info("verify failed!");
                        break;
                    }
                }
//                  Log.info("" + (System.currentTimeMillis() - time));
                if (!success) {
                    break;
                }
                time = System.currentTimeMillis();

                // save block
                Result result = blockService.saveBlock(block);
                success = result.isSuccess();
                if (!success) {
                    Log.warn("save block fail : reason : " + result.getMsg() + ", block height : " +
block.getHeader().getHeight() + ", hash : " + block.getHeader().getHash());
                } else {
                    RewardStatisticsProcess.addBlock(block);
                    //
                    nulsProtocolProcess.processProtocolUpGrade(block.getHeader());
                    BlockLog.debug("save block height : " + block.getHeader().getHeight() + " , hash : "
+ block.getHeader().getHash());
                }
//                  Log.info("" + (System.currentTimeMillis() - time));
            } while (false);
        } catch (Exception e) {
            Log.error("save block error : " + e.getMessage(), e);
        }
        if (success) {
            long t = System.currentTimeMillis();
            NulsContext.getInstance().setBestBlock(block);
            // remove tx from memory pool
            removeTxFromMemoryPool(block);
//              Log.info("" + (System.currentTimeMillis() - t));
            //
            forwardingBlock(blockContainer);
```

```
//                Log.info("" + (System.currentTimeMillis() - t));

                return true;
            } else {
                contractService.removeContractTempBalance();
                contractService.removeBatchExecute();
                contractService.removeCurrentBlockHeader();

                chainManager.getMasterChain().rollback(block);
                NulsContext.getInstance().setBestBlock(chainManager.getBestBlock());

                Log.error("save block fail : " + block.getHeader().getHeight() + " , isDownload : " +
isDownload);
            }
        } else {
            // Failed to block directly in the download
            //
            if (isDownload && !ConsensusStatusContext.isRunning()) {
                return false;
            }
            boolean hasFoundForkChain = checkAndAddForkChain(block);
            if (!hasFoundForkChain) {

                ChainLog.debug("add block {} - {} in queue", block.getHeader().getHeight(),
block.getHeader().getHash().getDigestHex());

                orphanBlockProvider.addBlock(blockContainer);
            }
        }
        return false;
    }

    /**
     * forwarding block
     * <p>
     *
     */
    private void forwardingBlock(BlockContainer blockContainer) {
        if (blockContainer.getStatus() == BlockContainerStatus.DOWNLOADING) {
            return;
        }
        if (blockContainer.getNode() == null) {
```

```
            return;
        }
        SmallBlock smallBlock = ConsensusTool.getSmallBlock(blockContainer.getBlock());
        cacheManager.cacheSmallBlock(smallBlock);
        Result result = blockService.forwardBlock(blockContainer.getBlock().getHeader().getHash(),
blockContainer.getNode());
        if (!result.isSuccess()) {
            Log.warn("forward the block failed, block height: " +
blockContainer.getBlock().getHeader().getHeight() + " , hash : " +
blockContainer.getBlock().getHeader().getHash());
        }
    }

    /**
     * The transaction is confirmed and the transaction in the memory pool is removed
     * <p>
     *
     *
     * @return boolean
     */
    public boolean removeTxFromMemoryPool(Block block) {
        boolean success = true;
        for (Transaction tx : block.getTxs()) {
            transactionCacheStorageService.removeTx(tx.getHash());
        }
        return success;
    }

    /**
     * When a new block cannot be added to the main chain, it may exist on a forked chain, or it
may be that the local main chain is not the latest one.
     * When this happens, it is necessary to check whether the block is forked with the main chain
or connected with an already existing forked chain.
     * if you can combine into a new branch chain, add a new branch chain
     * <p>
     *
     *
     *
     * @return boolean
     */
    protected boolean checkAndAddForkChain(Block block) {
        // check the preHash is in the other chain
```

```
        boolean hasFoundForkChain = checkForkChainFromForkChains(block);
        if (hasFoundForkChain) {
            return hasFoundForkChain;
        }
        return checkForkChainFromMasterChain(block);
    }

    /**
     * When a block cannot be connected to the main chain, it is checked whether it is a branch of the main chain.
     * If it is a branch of the main chain, a bifurcation chain is generated, and then the bifurcation chain is added into the forked chain pool to be verified.
     * <p>
     *
     *
     * @return boolean
     */
    protected boolean checkForkChainFromMasterChain(Block block) {

        BlockHeader blockHeader = block.getHeader();

        Chain masterChain = chainManager.getMasterChain().getChain();
        List<BlockHeader> headerList = masterChain.getBlockHeaderList();

        for (int i = headerList.size() - 1; i >= 0; i--) {
            BlockHeader header = headerList.get(i);

            if (header.getHash().equals(blockHeader.getHash())) {
                // found a same block , return true
                return true;
            } else if (header.getHash().equals(blockHeader.getPreHash())) {

                if (header.getHeight() + 1L != blockHeader.getHeight()) {
                    // Discard data blocks that are incorrect
                    //
                    return true;
                }
                Chain newForkChain = new Chain();

                newForkChain.getBlockList().add(block);
                newForkChain.getBlockHeaderList().add(block.getHeader());
```

```
                newForkChain.setStartBlockHeader(block.getHeader());
                newForkChain.setEndBlockHeader(block.getHeader());

                chainManager.getChains().add(new ChainContainer(newForkChain));
                return true;
            }

            if (header.getHeight() < blockHeader.getHeight()) {
                break;
            }
        }
        return false;
    }

    /**
     * When a block cannot be connected to the main chain, it checks whether it is a branch of a forked chain,
     * or is connected to a forked chain, and if so, it produces a forked chain, and then adds the forked chain into the fork to be verified. Chain pool;
     * or add the block directly to the corresponding branch chain
     * <p>
     *
     *
     * @return boolean
     */
    protected boolean checkForkChainFromForkChains(Block block) {

        BlockHeader blockHeader = block.getHeader();
        NulsDigestData preHash = blockHeader.getPreHash();

        // check the preHash is in the waitVerifyChainList
        for (ChainContainer chainContainer : chainManager.getChains()) {

            Chain forkChain = chainContainer.getChain();
            List<BlockHeader> headerList = forkChain.getBlockHeaderList();

            for (int i = headerList.size() - 1; i >= 0; i--) {
                BlockHeader header = headerList.get(i);

                if (header.getHash().equals(blockHeader.getHash())) {
                    // found a same block , return true
                    return true;
```

```
            } else if (header.getHash().equals(preHash)) {

                if (header.getHeight() + 1L != blockHeader.getHeight()) {
                    // Discard data blocks that are incorrect
                    //
                    return true;
                }

                // Check whether it is forked or connected. If it is a connection, add it.
                //
                if (i == headerList.size() - 1) {
                    chainContainer.getChain().setEndBlockHeader(block.getHeader());
                    chainContainer.getChain().getBlockHeaderList().add(block.getHeader());
                    chainContainer.getChain().getBlockList().add(block);
                    return true;
                }

                // The block is again forked in the forked chain
                //
                List<Block> blockList = forkChain.getBlockList();

                Chain newForkChain = new Chain();

                newForkChain.getBlockList().addAll(blockList.subList(0, i + 1));
                newForkChain.getBlockHeaderList().addAll(headerList.subList(0, i + 1));

                newForkChain.getBlockList().add(block);
                newForkChain.getBlockHeaderList().add(block.getHeader());

                newForkChain.setStartBlockHeader(forkChain.getStartBlockHeader());
                newForkChain.setEndBlockHeader(block.getHeader());

                return chainManager.getChains().add(new ChainContainer(newForkChain));
            } else if (header.getHeight() < blockHeader.getHeight()) {
                break;
            }
        }
    }
    return false;
  }

}
```

32:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\process\ConsensusProcess.java
 *
 */
```
package io.nuls.consensus.poc.process;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.cache.TxMemoryPool;
import io.nuls.consensus.poc.config.ConsensusConfig;
import io.nuls.consensus.poc.constant.BlockContainerStatus;
import io.nuls.consensus.poc.constant.ConsensusStatus;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.container.BlockContainer;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.manager.ChainManager;
import io.nuls.consensus.poc.model.BlockData;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.model.MeetingMember;
import io.nuls.consensus.poc.model.MeetingRound;
import io.nuls.consensus.poc.protocol.constant.PunishReasonEnum;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.protocol.tx.YellowPunishTransaction;
import io.nuls.consensus.poc.provider.BlockQueueProvider;
import io.nuls.consensus.poc.util.ConsensusTool;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.date.DateUtil;
import io.nuls.core.tools.log.Log;
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.model.*;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;
import io.nuls.network.service.NetworkService;
```

```
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.cache.TemporaryCacheManager;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.SmallBlock;
import io.nuls.protocol.model.tx.CoinBaseTransaction;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.TransactionService;
import io.nuls.protocol.utils.SmallBlockDuplicateRemoval;

import java.io.IOException;
import java.util.*;

/**
 * @author ln
 */
public class ConsensusProcess {

    private ChainManager chainManager;

    private TxMemoryPool txMemoryPool = TxMemoryPool.getInstance();
    private BlockQueueProvider blockQueueProvider = BlockQueueProvider.getInstance();

    private NetworkService networkService = NulsContext.getServiceBean(NetworkService.class);
    private LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);
    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);

    private TransactionService transactionService =
NulsContext.getServiceBean(TransactionService.class);
    private ContractService contractService = NulsContext.getServiceBean(ContractService.class);

    private TemporaryCacheManager temporaryCacheManager =
TemporaryCacheManager.getInstance();

    private boolean hasPacking;
    private long memoryPoolLastClearTime;

    public ConsensusProcess(ChainManager chainManager) {
        this.chainManager = chainManager;
    }

    public void process() {
        boolean canPackage = checkCanPackage();
```

```
        if (!canPackage) {
            return;
        }
        doWork();
    }

    private void doWork() {
        // pierre test comment out
        if (ConsensusStatusContext.getConsensusStatus().ordinal() <
ConsensusStatus.RUNNING.ordinal()) {
            return;
        }
        MeetingRound round = chainManager.getMasterChain().getOrResetCurrentRound();
        if (round == null) {
            return;
        }
        //check i am is a consensus node
        MeetingMember member = round.getMyMember();
        if (member == null) {
            clearTxMemoryPool();
            return;
        }
        if (!hasPacking && member.getPackStartTime() < TimeService.currentTimeMillis() &&
member.getPackEndTime() > TimeService.currentTimeMillis()) {
            hasPacking = true;
            try {
                if (Log.isDebugEnabled()) {
                    Log.debug(" " + DateUtil.convertDate(new Date(TimeService.currentTimeMillis())) + "
, : " +
                            DateUtil.convertDate(new Date(member.getPackStartTime())) + " , : " +
                            DateUtil.convertDate(new Date(member.getPackEndTime())) + " , : " +
                            DateUtil.convertDate(new Date(round.getStartTime())) + " , : " +
                            DateUtil.convertDate(new Date(round.getEndTime())));
                }
                packing(member, round);
            } catch (Exception e) {
                Log.error(e);
            }

            while (member.getPackEndTime() > TimeService.currentTimeMillis()) {
                try {
                    Thread.sleep(500L);
```

```
                } catch (InterruptedException e) {
                    Log.error(e);
                }
            }
            hasPacking = false;
        }
    }

    private void packing(MeetingMember self, MeetingRound round) throws IOException,
NulsException {

        waitReceiveNewestBlock(self, round);
        long start = System.currentTimeMillis();
        Block block = doPacking(self, round);
        Log.info("doPacking use:" + (System.currentTimeMillis() - start) + "ms");
        boolean need =
!block.getHeader().getPreHash().equals(chainManager.getBestBlock().getHeader().getHash());
        if (need) {
            for (Transaction transaction : block.getTxs()) {
                if (transaction.isSystemTx()) {
                    continue;
                }
                txMemoryPool.add(transaction, false);
            }
            start = System.currentTimeMillis();
            block = doPacking(self, round);
            Log.info("doPacking2 use:" + (System.currentTimeMillis() - start) + "ms");
        }
        if (null == block) {
            Log.error("make a null block");
            return;
        }

        boolean success = saveBlock(block);
        if (success) {
            broadcastSmallBlock(block);
        } else {
            Log.error("make a block, but save block error");
        }
    }

    private void clearTxMemoryPool() {
```

```
        if (TimeService.currentTimeMillis() - memoryPoolLastClearTime > 60000L) {
            txMemoryPool.clear();
            memoryPoolLastClearTime = TimeService.currentTimeMillis();
        }
    }

    private boolean checkCanPackage() {
        if (!ConsensusConfig.isPartakePacking()) {
            this.clearTxMemoryPool();
            return false;
        }
        // pierre test comment out
        // wait consensus ready running
        if (ConsensusStatusContext.getConsensusStatus().ordinal() <=
ConsensusStatus.WAIT_RUNNING.ordinal()) {
            return false;
        }
        // check network status
        if (networkService.getAvailableNodes().size() <
ProtocolConstant.ALIVE_MIN_NODE_COUNT) {
            return false;
        }
        return true;
    }

    private boolean waitReceiveNewestBlock(MeetingMember self, MeetingRound round) {

        long timeout = ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS / 2;
        long endTime = self.getPackStartTime() + timeout;

        boolean hasReceiveNewestBlock = false;

        try {
            while (!hasReceiveNewestBlock) {
                hasReceiveNewestBlock = hasReceiveNewestBlock(self, round);
                if (hasReceiveNewestBlock) {
                    break;
                }
                Thread.sleep(100L);
                if (TimeService.currentTimeMillis() >= endTime) {
                    break;
                }
```

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

        return !hasReceiveNewestBlock;
    }

    private boolean hasReceiveNewestBlock(MeetingMember self, MeetingRound round) {
        BlockHeader bestBlockHeader = blockService.getBestBlockHeader().getData();
        byte[] packingAddress = bestBlockHeader.getPackingAddress();

        int thisIndex = self.getPackingIndexOfRound();

        MeetingMember preMember;
        if (thisIndex == 1) {
            MeetingRound preRound = round.getPreRound();
            if (preRound == null) {
                Log.error("PreRound is null!");
                return true;
            }
            preMember = preRound.getMember(preRound.getMemberCount());
        } else {
            preMember = round.getMember(self.getPackingIndexOfRound() - 1);
        }
        if (preMember == null) {
            return true;
        }
        byte[] preBlockPackingAddress = preMember.getPackingAddress();
        long thisRoundIndex = preMember.getRoundIndex();
        int thisPackageIndex = preMember.getPackingIndexOfRound();

        BlockExtendsData blockRoundData = new BlockExtendsData(bestBlockHeader.getExtend());
        long roundIndex = blockRoundData.getRoundIndex();
        int packageIndex = blockRoundData.getPackingIndexOfRound();

        if (Arrays.equals(packingAddress, preBlockPackingAddress) && thisRoundIndex ==
roundIndex && thisPackageIndex == packageIndex) {
            return true;
        } else {
            return false;
        }
```

```
    }

    private boolean saveBlock(Block block) throws IOException {
        return blockQueueProvider.put(new BlockContainer(block,
BlockContainerStatus.RECEIVED));
    }

    private void broadcastSmallBlock(Block block) {
        SmallBlock smallBlock = ConsensusTool.getSmallBlock(block);
        temporaryCacheManager.cacheSmallBlock(smallBlock);
        SmallBlockDuplicateRemoval.needDownloadSmallBlock(smallBlock.getHeader().getHash());
        blockService.broadcastBlock(smallBlock);
    }

    private Block doPacking(MeetingMember self, MeetingRound round) throws NulsException,
IOException {
        Block bestBlock = chainManager.getBestBlock();
        /**
********************************************************************************************************** */

        // pierre test comment out
        //try {
        //    Log.info("");
        //    Log.info("**************************************************");
        //    Log.info("bestblock, height:{}- {}", bestBlock.getHeader().getHeight(),
bestBlock.getHeader().getHash());
        //    Log.info("EndBlockHeader, height:{}- {}",
chainManager.getMasterChain().getChain().getEndBlockHeader().getHeight(),
        //            chainManager.getMasterChain().getChain().getEndBlockHeader().getHash());
        //    Log.info("**************************************************");
        //    Log.info("");
        //
        //} catch (Exception e) {
        //    e.printStackTrace();
        //}
        /**
********************************************************************************************************** */

        BlockData bd = new BlockData();
        bd.setHeight(bestBlock.getHeader().getHeight() + 1);
        bd.setPreHash(bestBlock.getHeader().getHash());
        bd.setTime(self.getPackEndTime());
```

```
        BlockExtendsData extendsData = new BlockExtendsData();
        extendsData.setRoundIndex(round.getIndex());
        extendsData.setConsensusMemberCount(round.getMemberCount());
        extendsData.setPackingIndexOfRound(self.getPackingIndexOfRound());
        extendsData.setRoundStartTime(round.getStartTime());
        //
        if (NulsVersionManager.getCurrentVersion() > 1) {
            extendsData.setMainVersion(NulsVersionManager.getMainVersion());
            extendsData.setCurrentVersion(NulsVersionManager.getCurrentVersion());
extendsData.setPercent(NulsVersionManager.getCurrentProtocolContainer().getPercent());
extendsData.setDelay(NulsVersionManager.getCurrentProtocolContainer().getDelay());
        }

        StringBuilder str = new StringBuilder();
        str.append(self.getPackingAddress());
        str.append(" ,order:" + self.getPackingIndexOfRound());
        str.append(",packTime:" + new Date(self.getPackEndTime()));
        str.append("\n");
        Log.debug("pack round:" + str);

        bd.setExtendsData(extendsData);

        List<Transaction> packingTxList = new ArrayList<>();
        Set<NulsDigestData> outHashSet = new HashSet<>();

        long totalSize = 0L;

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

        /**
         * pierre add
         */
        byte[] stateRoot = ConsensusTool.getStateRoot(bestBlock.getHeader());
        //
        bd.setStateRoot(stateRoot);
        long height = bestBlock.getHeader().getHeight();
        ContractResult contractResult;
        Map<String, Coin> contractUsedCoinMap = new HashMap<>();
        int totalGasUsed = 0;

        int count = 0;
```

```
long start = 0;
long ledgerUse = 0;
long verifyUse = 0;
long outHashSetUse = 0;
long getTxUse = 0;
long sleepTIme = 0;
long whileTime = 0;
long startWhile = System.currentTimeMillis();
long sizeTime = 0;
long failed1Use = 0;
long addTime = 0;

Block tempBlock = new Block();
BlockHeader tempHeader = new BlockHeader();
tempHeader.setTime(bd.getTime());
tempHeader.setHeight(bd.getHeight());
tempHeader.setPackingAddress(round.getLocalPacker().getAddress().getAddressBytes());
tempBlock.setHeader(tempHeader);

//
contractService.createContractTempBalance();
//
contractService.createBatchExecute(stateRoot);
// ()
contractService.createCurrentBlockHeader(tempHeader);

List<ContractResult> contractResultList = new ArrayList<>();
Set<String> redPunishAddress = new HashSet<>();
while (true) {

    if ((self.getPackEndTime() - TimeService.currentTimeMillis()) <= 500L) {
        break;
    }
    start = System.nanoTime();
    Transaction tx = txMemoryPool.get();
    getTxUse += (System.nanoTime() - start);
    if (tx == null) {
        try {
            sleepTIme += 100;
            Thread.sleep(100L);
        } catch (InterruptedException e) {
            Log.error("packaging error ", e);
```

```
            }
            continue;
        }

        start = System.nanoTime();
        long txSize = tx.size();
        sizeTime += (System.nanoTime() - start);
        if ((totalSize + txSize) > ProtocolConstant.MAX_BLOCK_SIZE) {
            txMemoryPool.addInFirst(tx, false);
            break;
        }
        // GasGAS
        if (totalGasUsed > ContractConstant.MAX_PACKAGE_GAS &&
ContractUtil.isGasCostContractTransaction(tx)) {
            txMemoryPool.addInFirst(tx, false);
            continue;
        }
        count++;
        start = System.nanoTime();
        Transaction repeatTx = ledgerService.getTx(tx.getHash());
        ledgerUse += (System.nanoTime() - start);
        if (repeatTx != null) {
            continue;
        }

        ValidateResult result = ValidateResult.getSuccessResult();
        if (tx.isSystemTx() && tx.getType() == ConsensusConstant.TX_TYPE_RED_PUNISH) {
            RedPunishTransaction rpTx = (RedPunishTransaction) tx;
            boolean con =
redPunishAddress.add(AddressTool.getStringAddressByBytes(rpTx.getTxData().getAddress()))
&&
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentByAddress(rpTx
.getTxData().getAddress()) != null;
            result.setSuccess(con);
        } else if (tx.isSystemTx()) {
            result = ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TX_NOT_EFFECTIVE);
        } else {
            start = System.nanoTime();
            result = ledgerService.verifyCoinData(tx, toMaps, fromSet);
            verifyUse += (System.nanoTime() - start);
        }
```

```
            start = System.nanoTime();
            if (result.isFailed()) {
                if (tx == null) {
                    continue;
                }
                if (result.getErrorCode().equals(TransactionErrorCode.ORPHAN_TX)) {
                    txMemoryPool.add(tx, true);
                }
                failed1Use += (System.nanoTime() - start);
                continue;
            }
            start = System.nanoTime();
            if (!outHashSet.add(tx.getHash())) {
                outHashSetUse += (System.nanoTime() - start);
                Log.warn("");
                continue;
            }
            outHashSetUse += (System.nanoTime() - start);

            //
            if (ContractUtil.isContractTransaction(tx)) {
                contractResult = contractService.batchPackageTx(tx, height, tempBlock, stateRoot,
toMaps, contractUsedCoinMap).getData();
                if (contractResult != null) {
                    totalGasUsed += contractResult.getGasUsed();
                    contractResultList.add(contractResult);
                }
            }

            tx.setBlockHeight(bd.getHeight());
            start = System.nanoTime();
            packingTxList.add(tx);
            addTime += (System.nanoTime() - start);

            totalSize += txSize;
        }
        //
        contractService.removeContractTempBalance();
        stateRoot = contractService.commitBatchExecute().getData();
        //
        contractService.removeBatchExecute();
        //
```

```
contractService.removeCurrentBlockHeader();
tempBlock.getHeader().setStateRoot(stateRoot);
for (ContractResult result : contractResultList) {
    result.setStateRoot(stateRoot);
}
//
bd.setStateRoot(stateRoot);

whileTime = System.currentTimeMillis() - startWhile;
ValidateResult validateResult = null;
int failedCount = 0;
long failedUse = 0;

start = System.nanoTime();
while (null == validateResult || validateResult.isFailed()) {
    failedCount++;
    validateResult = transactionService.conflictDetect(packingTxList);
    if (validateResult.isFailed()) {
        if (validateResult.getData() instanceof Transaction) {
            packingTxList.remove(validateResult.getData());
        } else if (validateResult.getData() instanceof List) {
            List<Transaction> list = (List<Transaction>) validateResult.getData();
            if (list.size() == 2) {
                packingTxList.remove(list.get(1));
            } else {
                packingTxList.removeAll(list);
            }
        } else if (validateResult.getData() == null) {
            Log.error("Cann't find the wrong transaction!");
        }
    }
}
// CoinBaseGas
failedUse = System.nanoTime() - start;

start = System.nanoTime();
addConsensusTx(bestBlock, packingTxList, self, round);
long consensusTxUse = System.nanoTime() - start;
bd.setTxList(packingTxList);


start = System.nanoTime();
```

```
        //
        bd.getExtendsData().setStateRoot(bd.getStateRoot());

        Block newBlock = ConsensusTool.createBlock(bd, round.getLocalPacker());
        long createBlockUser = System.nanoTime() - start;
        Log.info("make block height:" + newBlock.getHeader().getHeight() + ",txCount: " +
newBlock.getTxs().size() + " , block size: " + newBlock.size() + " , time:" +
DateUtil.convertDate(new Date(newBlock.getHeader().getTime())) + ",packEndTime:" +
                DateUtil.convertDate(new Date(self.getPackEndTime())));

        // pierre test comment out
        Log.debug("\ncheck count:" + count + "\ngetTxUse:" + getTxUse / 1000000 + "
,\nledgerExistUse:" + ledgerUse / 1000000 + ", \nverifyUse:" + verifyUse / 1000000 + "
,\noutHashSetUse:" + outHashSetUse / 1000000 + " ,\nfailedTimes:" + failedCount + ",
\nfailedUse:" + failedUse / 1000000
                + " ,\nconsensusTx:" + consensusTxUse / 1000000 + ", \nblockUse:" + createBlockUser
/ 1000000 + ", \nsleepTIme:" + sleepTIme + ",\nwhileTime:" + whileTime
                + ", \naddTime:" + addTime / 1000000 + " ,\nsizeTime:" + sizeTime / 1000000 + "
,\nfailed1Use:" + failed1Use / 1000000);
        return newBlock;
    }

    /**
     * CoinBase transaction & Punish transaction
     *
     * @param bestBlock local highest block
     * @param txList    all tx of block
     * @param self      agent meeting data
     */
    private void addConsensusTx(Block bestBlock, List<Transaction> txList, MeetingMember self,
MeetingRound round) throws NulsException, IOException {
        CoinBaseTransaction coinBaseTransaction = ConsensusTool.createCoinBaseTx(self, txList,
round, bestBlock.getHeader().getHeight() + 1 +
PocConsensusConstant.COINBASE_UNLOCK_HEIGHT);
        txList.add(0, coinBaseTransaction);
        punishTx(bestBlock, txList, self, round);
    }

    private void punishTx(Block bestBlock, List<Transaction> txList, MeetingMember self,
MeetingRound round) throws NulsException, IOException {
        YellowPunishTransaction yellowPunishTransaction =
```

```
ConsensusTool.createYellowPunishTx(bestBlock, self, round);
        if (null == yellowPunishTransaction) {
            return;
        }
        txList.add(yellowPunishTransaction);
        //100
        //When 100 yellow CARDS in a row, give a red card.
        List<byte[]> addressList = yellowPunishTransaction.getTxData().getAddressList();
        Set<Integer> punishedSet = new HashSet<>();
        for (byte[] address : addressList) {
            MeetingMember member = round.getMemberByAgentAddress(address);
            if (null == member) {
                member = round.getPreRound().getMemberByAgentAddress(address);
            }
            if (member.getCreditVal() <= PocConsensusConstant.RED_PUNISH_CREDIT_VAL) {
                if (!punishedSet.add(member.getPackingIndexOfRound())) {
                    continue;
                }
                if (member.getAgent().getDelHeight() > 0L) {
                    continue;
                }
                RedPunishTransaction redPunishTransaction = new RedPunishTransaction();
                RedPunishData redPunishData = new RedPunishData();
                redPunishData.setAddress(address);
redPunishData.setReasonCode(PunishReasonEnum.TOO_MUCH_YELLOW_PUNISH.getCode()
);
                redPunishTransaction.setTxData(redPunishData);
                redPunishTransaction.setTime(self.getPackEndTime());
                CoinData coinData =
ConsensusTool.getStopAgentCoinData(redPunishData.getAddress(),
redPunishTransaction.getTime() + PocConsensusConstant.RED_PUNISH_LOCK_TIME);
                redPunishTransaction.setCoinData(coinData);
redPunishTransaction.setHash(NulsDigestData.calcDigestData(redPunishTransaction.serializeFor
Hash()));
                txList.add(redPunishTransaction);
            }
        }
    }
}
```

33:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\process\ForkChainProcess.java

```
 *
 */

package io.nuls.consensus.poc.process;

import com.google.common.primitives.UnsignedBytes;
import io.nuls.consensus.poc.constant.ConsensusStatus;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.container.ChainContainer;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.consensus.poc.locker.Lockers;
import io.nuls.consensus.poc.manager.ChainManager;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.model.Chain;
import io.nuls.consensus.poc.model.MeetingMember;
import io.nuls.consensus.poc.model.MeetingRound;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.util.ConsensusTool;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.service.ContractService;
import io.nuls.contract.util.ContractUtil;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.ChainLog;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.*;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.TransactionService;

import java.io.IOException;
import java.util.*;

/**
```

```
 * @author ln
 */
public class ForkChainProcess {

    private ChainManager chainManager;

    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);

    private long time = 0L;
    private long lastClearTime = 0L;

    private LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);
    private ContractService contractService = NulsContext.getServiceBean(ContractService.class);
    private TransactionService tansactionService =
NulsContext.getServiceBean(TransactionService.class);

    private NulsProtocolProcess nulsProtocolProcess = NulsProtocolProcess.getInstance();

    public ForkChainProcess(ChainManager chainManager) {
        this.chainManager = chainManager;
    }

    public boolean doProcess() throws IOException, NulsException {

        if (ConsensusStatusContext.getConsensusStatus().ordinal() <
ConsensusStatus.RUNNING.ordinal()) {
            return false;
        }
        Lockers.CHAIN_LOCK.lock();
        try {

            printChainStatusLog();

            // Monitor the status of the orphan chain, if it is available, join the verification chain
            //
            monitorOrphanChains();

            long newestBlockHeight = chainManager.getBestBlockHeight() +
PocConsensusConstant.CHANGE_CHAIN_BLOCK_DIFF_COUNT;

            ChainContainer newChain = chainManager.getMasterChain();
            if (null == newChain) {
```

```
            return false;
        }
        //hashhash
        BlockHeader newChainBlockHeader = newChain.getBestBlock().getHeader();

        Iterator<ChainContainer> iterator = chainManager.getChains().iterator();
        while (iterator.hasNext()) {
            ChainContainer forkChain = iterator.next();
            if (forkChain.getChain() == null || forkChain.getChain().getStartBlockHeader() == null ||
forkChain.getChain().getEndBlockHeader() == null) {
                iterator.remove();
                continue;
            }
            long newChainHeight = forkChain.getChain().getEndBlockHeader().getHeight();
            BlockHeader forkChainBlockHeader = forkChain.getChain().getEndBlockHeader();
            byte[] rightHash = null;
            byte[] rightAddress = null;
            //String forkChainBlockHash = forkChainBlockHeader.getHash().getDigestHex();
            byte[] forkChainBlockHash = forkChainBlockHeader.getHash().getDigestBytes();
            boolean sameAddress = false;
            //hash
            if (newChainBlockHeader.getHeight() == newChainHeight) {
                byte[] newChainBlockHash = newChainBlockHeader.getHash().getDigestBytes();
                rightHash = rightHash(newChainBlockHash, forkChainBlockHash);
                sameAddress = ArraysTool.arrayEquals(forkChainBlockHeader.getPackingAddress(),
newChainBlockHeader.getPackingAddress());
            }
            boolean hashEquals = Arrays.equals(forkChainBlockHash, rightHash);
            if (newChainHeight > newestBlockHeight
                    || (newChainHeight == newestBlockHeight &&
forkChain.getChain().getEndBlockHeader().getTime() <
newChain.getChain().getEndBlockHeader().getTime())
                    || (newChainBlockHeader.getHeight() == newChainHeight && hashEquals &&
sameAddress)) {
                if (newChainBlockHeader.getHeight() == newChainHeight && hashEquals &&
sameAddress) {
                    Log.info("-+-+-+-+-+-+-+-+- Change chain with the same height but different hash
block -+-+-+-+-+-+-+-+-");
                    Log.info("-+-+-+-+-+-+-+-+- height: " + newChainHeight + ", Right hash" +
Hex.encode(rightHash));
                    /**
********************************************************************************************** */
```

```
                try {
                    Log.info("");
                    Log.info("**************************************************");
                    Log.info("bestblock, height:{}- hash{}",
chainManager.getBestBlock().getHeader().getHeight(),
chainManager.getBestBlock().getHeader().getHash());
                    Log.info("**************************************************");
                    Log.info("");

                } catch (Exception e) {
                    e.printStackTrace();
                }
                /**
********************************************************************************************** */
            }
            newChain = forkChain;
            newestBlockHeight = newChainHeight;
        }
    }

    if (!newChain.equals(chainManager.getMasterChain())) {

        ChainLog.debug("discover the fork chain {} : start {} - {} , end {} - {} , exceed the master
{} - {} - {}, start verify the fork chian", newChain.getChain().getId(),
newChain.getChain().getStartBlockHeader().getHeight(),
newChain.getChain().getStartBlockHeader().getHash(),
newChain.getChain().getEndBlockHeader().getHeight(),
newChain.getChain().getEndBlockHeader().getHash(),
chainManager.getMasterChain().getChain().getId(), chainManager.getBestBlockHeight(),
chainManager.getBestBlock().getHeader().getHash());

        //ChainContainer resultChain = verifyNewChain(newChain);
        //Verify the new chain, combined with the current latest chain, to get the status of the
branch node
        //
        ChainContainer resultChain =
chainManager.getMasterChain().getBeforeTheForkChain(newChain);

        //Combined with the new bifurcated block chain, combine and verify one by one
        //
        List<Object[]> verifyResultList = new ArrayList<>();
        for (Block forkBlock : newChain.getChain().getBlockList()) {
```

```
                Result success = resultChain.verifyAndAddBlock(forkBlock, true, false);
                if (success.isFailed()) {
                    resultChain = null;
                    break;
                } else {
                    verifyResultList.add((Object[]) success.getData());
                }
            }

            if (resultChain == null) {
                ChainLog.debug("verify the fork chain fail {} remove it", newChain.getChain().getId());

                chainManager.getChains().remove(newChain);
            } else {
                //Verify pass, try to switch chain
                //
                boolean success = changeChain(resultChain, newChain, verifyResultList);
                if (success) {
                    chainManager.getChains().remove(newChain);
                    /**
********************************************************************************************************** */
                    try {
                        Log.info("");
                        Log.info("**************************************************");
                        Log.info(" bestblock, height:{}- {}",
chainManager.getBestBlock().getHeader().getHeight(),
chainManager.getBestBlock().getHeader().getHash());
                        Log.info("**************************************************");
                        Log.info("");

                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    /**
********************************************************************************************************** */
                }
                ChainLog.debug("verify the fork chain {} success, change master chain result : {} ,
new master chain is {} : {} - {}", newChain.getChain().getId(), success,
chainManager.getBestBlock().getHeader().getHeight(),
chainManager.getBestBlock().getHeader().getHash());
            }
        }
```

```
            clearExpiredChain();
        } finally {
            Lockers.CHAIN_LOCK.unlock();
        }
        return true;
    }

    /**
     * hashhash
     */
    private byte[] rightHash(byte[] hash1, byte[] hash2) {
        Comparator<byte[]> comparator = UnsignedBytes.lexicographicalComparator();
        if (comparator.compare(hash1, hash2) <= 0) {
            return hash1;
        }
        return hash2;
    }

    private void printChainStatusLog() {
        if (chainManager.getMasterChain() == null || chainManager.getMasterChain().getChain() ==
null || chainManager.getMasterChain().getChain().getEndBlockHeader() == null) {
            return;
        }

        if (time == 0L) {
            printLog();
        } else if (System.currentTimeMillis() - time > 5 * 60 * 1000L) {
            printLog();
        }
    }

    private void printLog() {
        time = System.currentTimeMillis();

        StringBuilder sb = new StringBuilder();

        sb.append("========================\n");

        sb.append("Master Chain Status : \n");
        sb.append(getChainStatus(chainManager.getMasterChain()));
```

```
        sb.append("\n");

        List<ChainContainer> chains = chainManager.getChains();

        if (chains != null && chains.size() > 0) {
            sb.append("fork chains : \n");
            for (ChainContainer chain : chains) {
                sb.append(getChainStatus(chain));
            }
            sb.append("\n");
        }

        List<ChainContainer> iss = chainManager.getOrphanChains();

        if (iss != null && iss.size() > 0) {
            sb.append("orphan chains : \n");
            for (ChainContainer chain : iss) {
                sb.append(getChainStatus(chain));
            }
            sb.append("\n");
        }

        ChainLog.debug(sb.toString());
    }

    private String getChainStatus(ChainContainer chain) {
        StringBuilder sb = new StringBuilder();

        if (chain == null || chain.getChain() == null) {
            return sb.toString();
        }

        sb.append("id: " + chain.getChain().getId() + "\n");

        if (chain.getChain().getStartBlockHeader() == null) {
            sb.append("start Block Header is null \n");
        } else {
            sb.append("start height : " + chain.getChain().getStartBlockHeader().getHeight() + " \n");
            sb.append("start hash : " + chain.getChain().getStartBlockHeader().getHash() + " \n");
        }
        if (chain.getChain().getEndBlockHeader() == null) {
            sb.append("end Block Header is null \n");
```

```
        } else {
            sb.append("end height : " + chain.getChain().getEndBlockHeader().getHeight() + " \n");
            sb.append("end hash : " + chain.getChain().getEndBlockHeader().getHash() + " \n");
        }

        List<BlockHeader> blockHeaderList = chain.getChain().getBlockHeaderList();

        if (blockHeaderList != null && blockHeaderList.size() > 0) {
            sb.append("start blockHeaders height : " + blockHeaderList.get(0).getHeight() + " \n");
            sb.append("end blockHeaders height : " + blockHeaderList.get(blockHeaderList.size() -
1).getHeight() + " \n");
            sb.append("start blockHeaders hash : " + blockHeaderList.get(0).getHash() + " \n");
            sb.append("end blockHeaders hash : " + blockHeaderList.get(blockHeaderList.size() -
1).getHash() + " \n");
        }

        List<Block> block = chain.getChain().getBlockList();

        if (block != null && block.size() > 0) {
            sb.append("start blocks height : " + block.get(0).getHeader().getHeight() + " \n");
            sb.append("end blocks height : " + block.get(block.size() - 1).getHeader().getHeight() + "
\n");
            sb.append("start blocks hash : " + block.get(0).getHeader().getHash() + " \n");
            sb.append("end blocks hash : " + block.get(block.size() - 1).getHeader().getHash() + " \n");
        }
        sb.append("\n");

        return sb.toString();
    }

    /**
     * Monitor the orphan chain, if there is a connection with the main chain or the forked chain, the
merged chain
     * <p>
     *
     */
    private void monitorOrphanChains() {
        List<ChainContainer> orphanChains = chainManager.getOrphanChains();

        Iterator<ChainContainer> iterator = orphanChains.iterator();
        while (iterator.hasNext()) {
            ChainContainer orphanChain = iterator.next();
```

```
            if (checkOrphanChainHasConnection(orphanChain)) {
                iterator.remove();
            }
        }
    }

    private boolean checkOrphanChainHasConnection(ChainContainer orphanChain) {
        // Determine whether the orphan chain is connected to the main chain
        //
        BlockHeader startBlockHeader = orphanChain.getChain().getStartBlockHeader();

        List<BlockHeader> blockHeaderList =
chainManager.getMasterChain().getChain().getBlockHeaderList();

        int count = blockHeaderList.size() >
PocConsensusConstant.MAX_ISOLATED_BLOCK_COUNT ?
PocConsensusConstant.MAX_ISOLATED_BLOCK_COUNT : blockHeaderList.size();
        for (int i = blockHeaderList.size() - 1; i >= blockHeaderList.size() - count; i--) {
            BlockHeader header = blockHeaderList.get(i);
            if (startBlockHeader.getPreHash().equals(header.getHash()) &&
startBlockHeader.getHeight() == header.getHeight() + 1) {
                //yes connectioned
orphanChain.getChain().setPreChainId(chainManager.getMasterChain().getChain().getId());
     chainManager.getChains().add(orphanChain);

                ChainLog.debug("discover the OrphanChain {} : start {} - {} , end {} - {} , connection the
master chain of {} - {} - {}, move into the fork chians", orphanChain.getChain().getId(),
startBlockHeader.getHeight(), startBlockHeader.getHash().getDigestHex(),
orphanChain.getChain().getEndBlockHeader().getHeight(),
orphanChain.getChain().getEndBlockHeader().getHash(),
chainManager.getMasterChain().getChain().getId(),
chainManager.getMasterChain().getChain().getBestBlock().getHeader().getHeight(),
chainManager.getMasterChain().getChain().getBestBlock().getHeader().getHash());

                return true;
            } else if (startBlockHeader.getHeight() > header.getHeight()) {
                break;
            }
        }

        // Determine whether the lone chain is connected to the forked chain to be verified
        //
```

```
        for (ChainContainer forkChain : chainManager.getChains()) {

            Chain chain = forkChain.getChain();

            if (startBlockHeader.getHeight() > chain.getEndBlockHeader().getHeight() + 1 ||
startBlockHeader.getHeight() <= chain.getStartBlockHeader().getHeight()) {
                continue;
            }

            blockHeaderList = chain.getBlockHeaderList();

            for (int i = 0; i < blockHeaderList.size(); i++) {
                BlockHeader header = blockHeaderList.get(i);
                if (startBlockHeader.getPreHash().equals(header.getHash()) &&
startBlockHeader.getHeight() == header.getHeight() + 1) {
                    //yes connectioned
                    orphanChain.getChain().setPreChainId(chain.getPreChainId());
                    orphanChain.getChain().setStartBlockHeader(chain.getStartBlockHeader());

                    orphanChain.getChain().getBlockHeaderList().addAll(0, blockHeaderList.subList(0, i +
1));
                    orphanChain.getChain().getBlockList().addAll(0, chain.getBlockList().subList(0, i + 1));

                    chainManager.getChains().add(orphanChain);

                    if (i == blockHeaderList.size() - 1) {
                        chainManager.getChains().remove(forkChain);
                    }

                    ChainLog.debug("discover the OrphanChain {} : start {} - {} , end {} - {} , connection
the fork chain of : start {} - {} , end {} - {}, move into the fork chians",
orphanChain.getChain().getId(), startBlockHeader.getHeight(),
startBlockHeader.getHash().getDigestHex(),
orphanChain.getChain().getEndBlockHeader().getHeight(),
orphanChain.getChain().getEndBlockHeader().getHash(),
chainManager.getMasterChain().getChain().getId(), chain.getStartBlockHeader().getHeight(),
chain.getStartBlockHeader().getHash(), chain.getEndBlockHeader().getHeight(),
chain.getEndBlockHeader().getHash());

                    return true;
                } else if (startBlockHeader.getHeight() == header.getHeight() + 1) {
                    break;
```

```
            }
        }
    }

    // Determine whether the orphan chains are connected
    //
    for (ChainContainer orphan : chainManager.getOrphanChains()) {
        if
(orphan.getChain().getEndBlockHeader().getHash().equals(orphanChain.getChain().getStartBlock
Header().getPreHash()) &&
                orphan.getChain().getEndBlockHeader().getHeight() + 1 ==
orphanChain.getChain().getStartBlockHeader().getHeight()) {
            Chain chain = orphan.getChain();
            chain.setEndBlockHeader(orphanChain.getChain().getEndBlockHeader());
            chain.getBlockHeaderList().addAll(orphanChain.getChain().getBlockHeaderList());
            chain.getBlockList().addAll(orphanChain.getChain().getBlockList());
            return true;
        }
    }

    return false;
}

/*
 * Verify the block header information of the new chain, and if they all pass, start switching
 * However, in the case that both are passed, it may also fail because the transaction is not
verified.
 * The transaction cannot be verified here because the data has not been rolled back
 *
 *
 *
 *
 */
private ChainContainer verifyNewChain(ChainContainer needVerifyChain) {
    //Verify the new chain, combined with the current latest chain, to get the status of the branch
node
    //
    ChainContainer forkChain =
chainManager.getMasterChain().getBeforeTheForkChain(needVerifyChain);

    //Combined with the new bifurcated block chain, combine and verify one by one
    //
```

```
        for (Block forkBlock : needVerifyChain.getChain().getBlockList()) {
            Result success = forkChain.verifyAndAddBlock(forkBlock, true, false);
            if (success.isFailed()) {
                return null;
            }
        }
        return forkChain;
    }

    /*
     * Switching the master chain to a new chain and verifying the block header before the switch is legal, so only the transactions in the block need to be verified here.
     * In order to ensure the correctness of the transaction verification, you first need to roll back all blocks after the fork of the main chain, and then the new chain will start to go into service.
     * If the verification fails during the warehousing process, it means that the transaction in the block is illegal, then the new connection that proves the need to switch is not trusted.
     * Once the new chain is not trusted, you need to add the previously rolled back block back
     * This method needs to be synchronized with the add block method
     *
     * master
     *
     *
     *
     *
     */
    private boolean changeChain(ChainContainer newMasterChain, ChainContainer originalForkChain, List<Object[]> verifyResultList) throws NulsException, IOException {

        if (newMasterChain == null || originalForkChain == null || verifyResultList == null) {
            return false;
        }

        //Now the master chain, the forked chain after the switch, needs to be put into the list of chains to be verified.
        //
        ChainContainer oldChain = chainManager.getMasterChain().getAfterTheForkChain(originalForkChain);

        //rollbackTransaction
        List<Block> rollbackBlockList = oldChain.getChain().getBlockList();

        ChainLog.debug("rollbackTransaction the master chain , need rollbackTransaction block
```

```
count is {}, master chain is {} : {} - {} , service best block : {} - {}", rollbackBlockList.size(),
chainManager.getMasterChain().getChain().getId(),
chainManager.getBestBlock().getHeader().getHeight(),
chainManager.getBestBlock().getHeader().getHash(),
blockService.getBestBlock().getData().getHeader().getHeight(),
blockService.getBestBlock().getData().getHeader().getHash());

        //Need descending order
        //
        Collections.reverse(rollbackBlockList);

        boolean rollbackResult = rollbackBlocks(rollbackBlockList);
        if (!rollbackResult) {
            return false;
        }

        boolean changeSuccess = true;

        List<Block> successList = new ArrayList<>();
        try {
            changeSuccess = doChange(successList, originalForkChain, verifyResultList);
        } catch (Exception e) {
            Log.error(e);
            changeSuccess = false;
        }
        ChainLog.debug("add new blocks complete, result {}, success count is {} , now service best
block : {} - {}", changeSuccess, successList.size(),
blockService.getBestBlock().getData().getHeader().getHeight(),
blockService.getBestBlock().getData().getHeader().getHash());

        if (changeSuccess) {
            chainManager.setMasterChain(newMasterChain);
            newMasterChain.initRound();
            NulsContext.getInstance().setBestBlock(newMasterChain.getBestBlock());

            if (oldChain.getChain().getBlockList().size() > 0) {
                chainManager.getChains().add(oldChain);
            }
        } else {
            //Fallback status
            //
            Collections.reverse(successList);
```

```
            for (Block rollBlock : successList) {
                Result rs = blockService.rollbackBlock(rollBlock);
                if (rs.isSuccess()) {
                    //
                    nulsProtocolProcess.processProtocolRollback(rollBlock.getHeader());
                }
                RewardStatisticsProcess.rollbackBlock(rollBlock);
            }

            Collections.reverse(rollbackBlockList);
            for (Block addBlock : rollbackBlockList) {
                Result rs = blockService.saveBlock(addBlock);
                RewardStatisticsProcess.addBlock(addBlock);
                if (rs.isSuccess()) {
                    //
                    nulsProtocolProcess.processProtocolUpGrade(addBlock.getHeader());
                }
            }
        }
        return changeSuccess;
    }

    private boolean doChange(List<Block> successList, ChainContainer originalForkChain,
List<Object[]> verifyResultList) {
        boolean changeSuccess = true;
        //add new block
        List<Block> addBlockList = originalForkChain.getChain().getBlockList();

        Result<Block> preBlockResult =
blockService.getBlock(addBlockList.get(0).getHeader().getPreHash());
        Block preBlock = preBlockResult.getData();
        Block newBlock;
        //Need to sort in ascending order, the default is
        //
        //for (Block newBlock : addBlockList) {
        for (int i = 0, size = addBlockList.size(); i < size; i++) {
            newBlock = addBlockList.get(i);
            newBlock.verifyWithException();

            Map<String, Coin> toMaps = new HashMap<>();
            Set<String> fromSet = new HashSet<>();
```

```
/**
 * pierre add
 */
long bestHeight = preBlock.getHeader().getHeight();
byte[] stateRoot = ConsensusTool.getStateRoot(preBlock.getHeader());
preBlock = newBlock;
BlockHeader verifyHeader = newBlock.getHeader();
byte[] receiveStateRoot = ConsensusTool.getStateRoot(verifyHeader);
ContractResult contractResult;
Map<String, Coin> contractUsedCoinMap = new HashMap<>();
int totalGasUsed = 0;

//
contractService.createContractTempBalance();
//
contractService.createBatchExecute(stateRoot);
// ()
BlockHeader tempHeader = new BlockHeader();
tempHeader.setTime(verifyHeader.getTime());
tempHeader.setHeight(verifyHeader.getHeight());
tempHeader.setPackingAddress(verifyHeader.getPackingAddress());
contractService.createCurrentBlockHeader(tempHeader);

List<ContractResult> contractResultList = new ArrayList<>();
// stateRoot,
byte[] tempStateRoot = null;

for (Transaction tx : newBlock.getTxs()) {

    if (tx.isSystemTx()) {
        continue;
    }

    // GasGAS
    if (totalGasUsed > ContractConstant.MAX_PACKAGE_GAS) {
        if (ContractUtil.isGasCostContractTransaction(tx)) {
            Log.info("verify block failed: Excess contract transaction detected.");
            changeSuccess = false;
            break;
        }
    }
```

```
            ValidateResult result = tx.verify();
            if (result.isSuccess()) {
                result = ledgerService.verifyCoinData(tx, toMaps, fromSet, bestHeight);
                if (result.isFailed()) {
                    ErrorData errorData = (ErrorData) result.getData();
                    if (null == errorData) {
                        Log.info("failed message:" + result.getMsg());
                    } else {
                        Log.info("failed message:" + errorData.getMsg());
                    }
                    changeSuccess = false;
                    break;
                }
            } else {
                ErrorData errorData = (ErrorData) result.getData();
                if (null == errorData) {
                    Log.info("failed message:" + result.getMsg());
                } else {
                    Log.info("failed message:" + errorData.getMsg());
                }
                changeSuccess = false;
                break;
            }

            //
            if (ContractUtil.isContractTransaction(tx)) {
                contractResult = contractService.batchProcessTx(tx, bestHeight, newBlock,
stateRoot, toMaps, contractUsedCoinMap, true).getData();
                if (contractResult != null) {
                    tempStateRoot = contractResult.getStateRoot();
                    totalGasUsed += contractResult.getGasUsed();
                    contractResultList.add(contractResult);
                }
            }

        }

        if (!changeSuccess) {
            break;
        }
```

```
        //
        contractService.removeContractTempBalance();
        stateRoot = contractService.commitBatchExecute().getData();
        //
        contractService.removeBatchExecute();
        //
        contractService.removeCurrentBlockHeader();

        //
        if (tempStateRoot != null) {
            stateRoot = tempStateRoot;
        }
        newBlock.getHeader().setStateRoot(stateRoot);
        for (ContractResult result : contractResultList) {
            result.setStateRoot(stateRoot);
        }

        //
        if ((receiveStateRoot != null || stateRoot != null) && !Arrays.equals(receiveStateRoot,
stateRoot)) {
            Log.info("contract stateRoot incorrect. receiveStateRoot is {}, stateRoot is {}.",
receiveStateRoot != null ? Hex.encode(receiveStateRoot) : receiveStateRoot, stateRoot != null ?
Hex.encode(stateRoot) : stateRoot);
            changeSuccess = false;
            break;
        }

        // CoinBase
        Object[] objects = verifyResultList.get(i);
        MeetingRound currentRound = (MeetingRound) objects[0];
        MeetingMember member = (MeetingMember) objects[1];
        if (!chainManager.getMasterChain().verifyCoinBaseTx(newBlock, currentRound, member))
{
            changeSuccess = false;
            break;
        }

        if (!changeSuccess) {
            break;
        }
        ValidateResult validateResult1 = tansactionService.conflictDetect(newBlock.getTxs());
        if (validateResult1.isFailed()) {
```

```
                Log.info("failed message:" + validateResult1.getMsg());
                changeSuccess = false;
                break;
            }

            try {
                Result result = blockService.saveBlock(newBlock);
                boolean success = result.isSuccess();
                if (success) {
                    //
                    successList.add(newBlock);
                    nulsProtocolProcess.processProtocolUpGrade(newBlock.getHeader());
                } else {
                    ChainLog.debug("save block error : " + result.getMsg() + " , block height : " +
newBlock.getHeader().getHeight() + " , hash: " + newBlock.getHeader().getHash());
                    changeSuccess = false;
                    break;
                }
            } catch (Exception e) {
                Log.info("change fork chain error at save block, ", e);
                changeSuccess = false;
                break;
            }
        }
        if (!changeSuccess) {
            contractService.removeContractTempBalance();
            contractService.removeBatchExecute();
            contractService.removeCurrentBlockHeader();
        }
        return changeSuccess;
    }

    private boolean rollbackBlocks(List<Block> rollbackBlockList) {

        List<Block> rollbackList = new ArrayList<>();
        for (Block rollbackBlock : rollbackBlockList) {
            try {
                boolean success = blockService.rollbackBlock(rollbackBlock).isSuccess();
                if (success) {
                    //
                    nulsProtocolProcess.processProtocolRollback(rollbackBlock.getHeader());
                    RewardStatisticsProcess.rollbackBlock(rollbackBlock);
```

```
                rollbackList.add(rollbackBlock);
            } else {
                Collections.reverse(rollbackList);
                for (Block block : rollbackList) {
                    try {
                        Result rs = blockService.saveBlock(block);
                        RewardStatisticsProcess.addBlock(block);
                        if (rs.isSuccess()) {
                            //
                            nulsProtocolProcess.processProtocolUpGrade(block.getHeader());
                        }
                    } catch (Exception ex) {
                        Log.error("Rollback failed, failed to save block during recovery", ex);
                        break;
                    }
                }
                Log.error("Rollback block height : " + rollbackBlock.getHeader().getHeight() + " hash :
" + rollbackBlock.getHeader().getHash() + " failed, change chain failed !");
                return false;
            }
        } catch (Exception e) {
            Collections.reverse(rollbackList);
            for (Block block : rollbackList) {
                try {
                    Result rs = blockService.saveBlock(block);
                    RewardStatisticsProcess.addBlock(block);
                    if (rs.isSuccess()) {
                        //
                        nulsProtocolProcess.processProtocolUpGrade(block.getHeader());
                    }
                } catch (Exception ex) {
                    Log.error("Rollback failed, failed to save block during recovery", ex);
                    break;
                }
            }
            Log.error("Rollback failed during switch chain, skip this chain", e);
            e.printStackTrace();
            return false;
        }
    }

    ChainLog.debug("rollbackTransaction complete, success count is {} , now service best block :
```

```
{} - {}", rollbackList.size(), blockService.getBestBlock().getData().getHeader().getHeight(),
blockService.getBestBlock().getData().getHeader().getHash());
        return true;
    }

    protected void clearExpiredChain() {
        if (TimeService.currentTimeMillis() - lastClearTime <
PocConsensusConstant.CLEAR_INTERVAL_TIME) {
            return;
        }
        lastClearTime = TimeService.currentTimeMillis();
        //clear the master data
        clearMasterDatas();

        //clear the expired chain
        long bestHeight = chainManager.getBestBlockHeight();

        Iterator<ChainContainer> it = chainManager.getChains().iterator();
        while (it.hasNext()) {
            ChainContainer chain = it.next();
            if (checkChainIsExpired(chain, bestHeight)) {
                it.remove();
            }
        }

        it = chainManager.getOrphanChains().iterator();
        while (it.hasNext()) {
            ChainContainer orphanChain = it.next();
            if (checkChainIsExpired(orphanChain, bestHeight)) {
                it.remove();
            }
        }
    }

    private boolean checkChainIsExpired(ChainContainer orphanChain, long bestHeight) {
        if (bestHeight - orphanChain.getChain().getEndBlockHeader().getHeight() >
PocConsensusConstant.MAX_ISOLATED_BLOCK_COUNT) {
            return true;
        }
        return false;
    }
```

```
    private void clearMasterDatas() {
        clearMasterChainRound();
        clearMasterChainData();
    }

    private void clearMasterChainData() {
        Chain masterChain = chainManager.getMasterChain().getChain();
        long bestHeight = masterChain.getEndBlockHeader().getHeight();

        List<BlockHeader> blockHeaderList = masterChain.getBlockHeaderList();
        List<Block> blockList = masterChain.getBlockList();

        if (blockHeaderList.size() > 30000) {
            masterChain.setBlockHeaderList(blockHeaderList.subList(blockHeaderList.size() - 30000,
blockHeaderList.size()));
        }
        if (blockList.size() > PocConsensusConstant.MAX_ISOLATED_BLOCK_COUNT) {
            masterChain.setBlockList(blockList.subList(blockList.size() -
PocConsensusConstant.MAX_ISOLATED_BLOCK_COUNT, blockList.size()));
        }

        List<Agent> agentList = masterChain.getAgentList();
        List<Deposit> depositList = masterChain.getDepositList();

        Iterator<Agent> ait = agentList.iterator();
        while (ait.hasNext()) {
            Agent agent = ait.next();
            if (agent.getDelHeight() > 0L && (bestHeight - 1000) > agent.getDelHeight()) {
                ait.remove();
            }
        }

        Iterator<Deposit> dit = depositList.iterator();
        while (dit.hasNext()) {
            Deposit deposit = dit.next();
            if (deposit.getDelHeight() > 0L && (bestHeight - 1000) > deposit.getDelHeight()) {
                dit.remove();
            }
        }

        BlockExtendsData roundData = new
BlockExtendsData(chainManager.getBestBlock().getHeader().getExtend());
```

```
        List<PunishLogPo> yellowList = masterChain.getYellowPunishList();
        Iterator<PunishLogPo> yit = yellowList.iterator();
        while (yit.hasNext()) {
            PunishLogPo punishLog = yit.next();
            if (punishLog.getRoundIndex() < roundData.getPackingIndexOfRound() -
PocConsensusConstant.INIT_HEADERS_OF_ROUND_COUNT) {
                yit.remove();
            }
        }
    }

    private void clearMasterChainRound() {
chainManager.getMasterChain().clearRound(PocConsensusConstant.CLEAR_MASTER_CHAIN_
ROUND_COUNT);
    }


}

34:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\process\NulsProtocolProcess.java
 */
package io.nuls.consensus.poc.process;

import io.nuls.consensus.poc.cache.TxMemoryPool;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.manager.RoundManager;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.model.MeetingMember;
import io.nuls.consensus.poc.model.MeetingRound;
import io.nuls.consensus.poc.storage.service.TransactionCacheStorageService;
import io.nuls.consensus.poc.storage.service.TransactionQueueStorageService;
import io.nuls.consensus.poc.util.ProtocolTransferTool;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Address;
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.protocol.base.version.NulsVersionManager;
```

```
import io.nuls.protocol.base.version.ProtocolContainer;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.storage.po.BlockProtocolInfoPo;
import io.nuls.protocol.storage.po.ProtocolInfoPo;
import io.nuls.protocol.storage.po.ProtocolTempInfoPo;
import io.nuls.protocol.storage.service.VersionManagerStorageService;

import java.math.BigDecimal;
import java.util.*;

public class NulsProtocolProcess {

    private static NulsProtocolProcess protocolProcess = new NulsProtocolProcess();

    private NulsProtocolProcess() {

    }

    public static NulsProtocolProcess getInstance() {
        return protocolProcess;
    }

    private VersionManagerStorageService versionManagerStorageService;

    private BlockService blockService;

    /**
     *
     *
     * @param blockHeader
     */
    public void processProtocolUpGrade(BlockHeader blockHeader) {
        BlockExtendsData extendsData = new BlockExtendsData(blockHeader.getExtend());
        //
        if (extendsData.getCurrentVersion() == null) {
            extendsData.setCurrentVersion(1);
        }
        if (extendsData.getCurrentVersion() == 2) {
            return;
        }
NulsVersionManager.getConsensusVersionMap().put(AddressTool.getStringAddressByBytes(bloc
kHeader.getPackingAddress()), extendsData.getCurrentVersion());
```

```
getVersionManagerStorageService().saveConsensusVersionMap(NulsVersionManager.getConse
nsusVersionMap());
        if (extendsData.getCurrentVersion() < 1) {
getVersionManagerStorageService().saveConsensusVersionHeight(blockHeader.getHeight());
    return;
        }
        refreshProtocolCoverageRate(extendsData, blockHeader);
        refreshTempProtocolCoverageRate(extendsData, blockHeader);
        //
        if (extendsData.getCurrentVersion() > NulsVersionManager.getMainVersion()) {
            ProtocolContainer protocolContainer =
NulsVersionManager.getProtocolContainer(extendsData.getCurrentVersion());
            if (protocolContainer != null) {
                calcNewProtocolCoverageRate(protocolContainer, extendsData, blockHeader);
                //tempInfoblockpackingaddressSet
                for (ProtocolTempInfoPo tempInfoPo :
getVersionManagerStorageService().getProtocolTempMap().values()) {
                    if (tempInfoPo.getStatus() != ProtocolContainer.VALID) {
                        String packingAddress =
AddressTool.getStringAddressByBytes(blockHeader.getPackingAddress());
                        if (tempInfoPo.getAddressSet().contains(packingAddress)) {
                            tempInfoPo.getAddressSet().remove(packingAddress);
                        }
                    }
                }
                //Container
                for (ProtocolContainer container :
NulsVersionManager.getAllProtocolContainers().values()) {
                    if (container.getStatus() != ProtocolContainer.VALID
                            && container.getVersion().intValue() !=
protocolContainer.getVersion().intValue()) {
                        String packingAddress =
AddressTool.getStringAddressByBytes(blockHeader.getPackingAddress());
                        if (container.getAddressSet().contains(packingAddress)) {
                            container.getAddressSet().remove(packingAddress);
                        }
                    }
                }
            } else {
                //,
                ProtocolTempInfoPo protocolTempInfoPo =
getVersionManagerStorageService().getProtocolTempInfoPo(extendsData.getProtocolKey());
```

```
            if (protocolTempInfoPo == null) {
                protocolTempInfoPo =
ProtocolTransferTool.createProtocolTempInfoPo(extendsData);
            }
            calcTempProtocolCoverageRate(protocolTempInfoPo, extendsData, blockHeader);
            //Containerblockpacking
            for (ProtocolContainer container :
NulsVersionManager.getAllProtocolContainers().values()) {
                if (container.getStatus() != ProtocolContainer.VALID) {
                    String packingAddress =
AddressTool.getStringAddressByBytes(blockHeader.getPackingAddress());
                    if (container.getAddressSet().contains(packingAddress)) {
                        container.getAddressSet().remove(packingAddress);
                    }
                }
            }
            //tempInfo
            for (ProtocolTempInfoPo tempInfoPo :
getVersionManagerStorageService().getProtocolTempMap().values()) {
                if (tempInfoPo.getStatus() != ProtocolContainer.VALID
                        && tempInfoPo.getVersion() != protocolTempInfoPo.getVersion()) {
                    String packingAddress =
AddressTool.getStringAddressByBytes(blockHeader.getPackingAddress());
                    if (tempInfoPo.getAddressSet().contains(packingAddress)) {
                        tempInfoPo.getAddressSet().remove(packingAddress);
                    }
                }
            }
        }
    } else {
        calcDelay(blockHeader, extendsData);
        calcTempDelay(blockHeader, extendsData);
    }
getVersionManagerStorageService().saveConsensusVersionHeight(blockHeader.getHeight());
 //
    if (extendsData.getCurrentVersion() == 1) {
        extendsData.setCurrentVersion(null);
    }
  }

  /**
   *
```

```
    *
    * @param extendsData
    */
    private void refreshProtocolCoverageRate(BlockExtendsData extendsData, BlockHeader
header) {
        //
        for (ProtocolContainer container : NulsVersionManager.getAllProtocolContainers().values()) {
            if (container.getStatus() != ProtocolContainer.VALID) {
                //
                if (container.getRoundIndex() < extendsData.getRoundIndex()) {
                    MeetingRound currentRound =
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound();
                    List<MeetingMember> memberList = currentRound.getMemberList();
                    Set<String> memberAddressSet = new HashSet<>();
                    for (MeetingMember member : memberList) {
memberAddressSet.add(AddressTool.getStringAddressByBytes(member.getPackingAddress()));
                    }
                    Iterator<String> iterator = container.getAddressSet().iterator();
                    while (iterator.hasNext()) {
                        String address = iterator.next();
                        if (!memberAddressSet.contains(address)) {
                            iterator.remove();
                        }
                    }
                    if (container.getStatus() == ProtocolContainer.INVALID) {
                        container.setCurrentDelay(0);
                    } else {
                        //
                        //
                        Result<BlockHeader> result =
getBlockService().getBlockHeader(header.getPreHash());
                        BlockHeader preHeader = result.getData();
                        BlockExtendsData preExtendsData = new
BlockExtendsData(preHeader.getExtend());

                        int rate = calcRate(container, preExtendsData);
                        container.setCurrentPercent(rate);
                        if (rate < container.getPercent()) {
                            container.setCurrentDelay(0);
                            container.setStatus(ProtocolContainer.INVALID);
                        }
                    }
```

```
                container.setPrePercent(container.getCurrentPercent());
                //container.getAddressSet().clear();
                container.setRoundIndex(extendsData.getRoundIndex());
                saveProtocolInfo(container);
            }
        }
    }
}

/**
 * ()
 *
 * @param extendsData
 */
private void refreshTempProtocolCoverageRate(BlockExtendsData extendsData, BlockHeader
header) {
    for (ProtocolTempInfoPo tempInfoPo :
getVersionManagerStorageService().getProtocolTempMap().values()) {
        if (tempInfoPo.getStatus() != ProtocolContainer.VALID) {

            //
            if (tempInfoPo.getRoundIndex() < extendsData.getRoundIndex()) {
                if (tempInfoPo.getStatus() == ProtocolContainer.INVALID) {
                    tempInfoPo.setCurrentDelay(0);
                } else {

                    MeetingRound currentRound =
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound();
                    List<MeetingMember> memberList = currentRound.getMemberList();
                    Set<String> memberAddressSet = new HashSet<>();
                    for (MeetingMember member : memberList) {
memberAddressSet.add(AddressTool.getStringAddressByBytes(member.getPackingAddress()));
                    }
                    Iterator<String> iterator = tempInfoPo.getAddressSet().iterator();
                    while (iterator.hasNext()) {
                        String address = iterator.next();
                        if (!memberAddressSet.contains(address)) {
                            iterator.remove();
                        }
                    }
                    //
                    //
```

```
                    Result<BlockHeader> result =
getBlockService().getBlockHeader(header.getPreHash());
                    BlockHeader preHeader = result.getData();
                    BlockExtendsData preExtendsData = new
BlockExtendsData(preHeader.getExtend());
                    int rate = calcRate(tempInfoPo, preExtendsData);
                    tempInfoPo.setCurrentPercent(rate);
                    if (rate < tempInfoPo.getPercent()) {
                        tempInfoPo.setCurrentDelay(0);
                        tempInfoPo.setStatus(ProtocolContainer.INVALID);
                    }
                }
                tempInfoPo.setPrePercent(tempInfoPo.getCurrentPercent());
                //tempInfoPo.getAddressSet().clear();
                tempInfoPo.setRoundIndex(extendsData.getRoundIndex());
                getVersionManagerStorageService().saveProtocolTempInfoPo(tempInfoPo);
            }
        }
    }
}

/**
 *
 *
 * @param container
 * @param extendsData
 */
private void calcNewProtocolCoverageRate(ProtocolContainer container, BlockExtendsData
extendsData, BlockHeader blockHeader) {
container.getAddressSet().add(AddressTool.getStringAddressByBytes(blockHeader.getPackingAd
dress()));
    container.setRoundIndex(extendsData.getRoundIndex());
    int rate = calcRate(container, extendsData);
    container.setCurrentPercent(rate);
    //
    if (container.getStatus() == ProtocolContainer.INVALID) {
        //
        if (rate >= container.getPercent()) {
            container.setStatus(ProtocolContainer.DELAY_LOCK);
            container.setCurrentDelay(1);
        }
        Log.info("==========  ==========");
```

```
            Log.info("========== " + rate + " -->>> " + container.getPercent());
            Log.info("========== " + container.getPrePercent());
            Log.info("========== version" + container.getVersion());
            Log.info("========== " + blockHeader.getHeight());
            Log.info("========== hash" + blockHeader.getHash());
            Log.info("========== " + container.getStatus());
            Log.info("========== " + container.getCurrentDelay());
            Log.info("========== " + container.getRoundIndex());
            Log.info("========== AddressSet" +
Arrays.toString(container.getAddressSet().toArray()));
        } else if (container.getStatus() == ProtocolContainer.DELAY_LOCK) {
            //
            container.setCurrentDelay(container.getCurrentDelay() + 1);

            //
            if (container.getCurrentDelay() >= container.getDelay()) {
                container.setStatus(ProtocolContainer.VALID);
                container.setEffectiveHeight(blockHeader.getHeight() + 1);
                upgradeProtocol(container);
                clearIncompatibleTx();
                Log.info("**********  **********");
                Log.info("**********  **********");
                Log.info("**********  **********");
                Log.info("**********  **********");
                Log.info("********** version" + container.getVersion());
                Log.info("********** " + container.getEffectiveHeight());
                Log.info("********** " + container.getStatus());
                Log.info("********** " + container.getCurrentDelay());
                Log.info("********** " + container.getRoundIndex());
                Log.info("********** AddressSet" + Arrays.toString(container.getAddressSet().toArray()));
            } else {
                Log.info("==========  ==========");
                Log.info("========== version" + container.getVersion());
                Log.info("========== " + blockHeader.getHeight());
                Log.info("========== hash" + blockHeader.getHash());
                Log.info("========== " + container.getStatus());
                Log.info("========== " + container.getCurrentDelay());
                Log.info("========== " + container.getRoundIndex());
                Log.info("========== AddressSet" +
Arrays.toString(container.getAddressSet().toArray()));
            }
        }
```

```
        saveProtocolInfo(container);
        saveBlockProtocolInfo(blockHeader, container);
    }

    private void calcTempProtocolCoverageRate(ProtocolTempInfoPo tempInfoPo,
BlockExtendsData extendsData, BlockHeader blockHeader) {
tempInfoPo.getAddressSet().add(AddressTool.getStringAddressByBytes(blockHeader.getPacking
Address()));
        tempInfoPo.setRoundIndex(extendsData.getRoundIndex());
        int rate = calcRate(tempInfoPo, extendsData);
        tempInfoPo.setCurrentPercent(rate);
        MeetingRound currentRound =
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound();
        //
        if (tempInfoPo.getStatus() == ProtocolContainer.INVALID) {
            //
            if (rate >= tempInfoPo.getPercent()) {
                tempInfoPo.setStatus(ProtocolContainer.DELAY_LOCK);
                tempInfoPo.setCurrentDelay(1);
            }
            getVersionManagerStorageService().saveProtocolTempInfoPo(tempInfoPo);
            Log.info("========== Temp  ==========");
            Log.info("========== " + rate + " -->>>" + tempInfoPo.getPercent());
            Log.info("========== " + tempInfoPo.getPrePercent());
            Log.info("========== version" + tempInfoPo.getVersion());
            Log.info("========== " + blockHeader.getHeight());
            Log.info("========== hash" + blockHeader.getHash());
            Log.info("========== " + tempInfoPo.getStatus());
            Log.info("========== " + tempInfoPo.getCurrentDelay());
            Log.info("========== " + tempInfoPo.getRoundIndex());
            Log.info("========== AddressSet" +
Arrays.toString(tempInfoPo.getAddressSet().toArray()));
            Log.info("========== " + currentRound.getMemberCount());
            Log.info("========== " + Arrays.toString(currentRound.getMemberList().toArray()));
        } else if (tempInfoPo.getStatus() == ProtocolContainer.DELAY_LOCK) {
            //
            tempInfoPo.setCurrentDelay(tempInfoPo.getCurrentDelay() + 1);

            //
            if (tempInfoPo.getCurrentDelay() >= tempInfoPo.getDelay()) {
                tempInfoPo.setStatus(ProtocolContainer.VALID);
                tempInfoPo.setEffectiveHeight(blockHeader.getHeight() + 1);
```

```
            getVersionManagerStorageService().saveProtocolTempInfoPo(tempInfoPo);
            Log.info("**********  **********");
            Log.info("********** version" + tempInfoPo.getVersion());
            Log.info("********** " + tempInfoPo.getEffectiveHeight());
            Log.info("********** " + tempInfoPo.getStatus());
            Log.info("********** " + tempInfoPo.getCurrentDelay());
            Log.info("********** " + tempInfoPo.getRoundIndex());
            Log.info("********** AddressSet" +
Arrays.toString(tempInfoPo.getAddressSet().toArray()));
            //upgradeProtocol(container, blockHeader);
            //linuxtrue
            if (System.getProperties().getProperty("os.name").toUpperCase().indexOf("LINUX") != -
1) {
                saveBlockTempProtocolInfo(blockHeader, tempInfoPo);
                Log.error(">>>>>> The new protocol version has taken effect, this program version is
too low has stopped automatically, please upgrade immediately **********");
                Log.error(">>>>>> The new protocol version has taken effect, this program version is
too low has stopped automatically, please upgrade immediately **********");
                NulsContext.getInstance().exit(1);
            } else {
                NulsContext.mastUpGrade = true;
            }
        } else {
            getVersionManagerStorageService().saveProtocolTempInfoPo(tempInfoPo);
            Log.info("========== Temp  ==========");
            Log.info("========== version" + tempInfoPo.getVersion());
            Log.info("========== " + blockHeader.getHeight());
            Log.info("========== hash" + blockHeader.getHash());
            Log.info("========== " + tempInfoPo.getStatus());
            Log.info("========== " + tempInfoPo.getCurrentDelay());
            Log.info("========== " + tempInfoPo.getRoundIndex());
            Log.info("========== AddressSet" +
Arrays.toString(tempInfoPo.getAddressSet().toArray()));
            Log.info("========== " + currentRound.getMemberCount());
            Log.info("========== " + Arrays.toString(currentRound.getMemberList().toArray()));
        }
    }
    saveBlockTempProtocolInfo(blockHeader, tempInfoPo);
}


private void calcDelay(BlockHeader blockHeader, BlockExtendsData extendsData) {
```

```
        for (ProtocolContainer container : NulsVersionManager.getAllProtocolContainers().values()) {
            if (extendsData.getCurrentVersion() <= NulsVersionManager.getMainVersion()) {
                continue;
            }
            if (container.getVersion().intValue() != extendsData.getCurrentVersion().intValue()
                    && container.getStatus() != ProtocolContainer.VALID) {
                String packingAddress =
AddressTool.getStringAddressByBytes(blockHeader.getPackingAddress());
                if (container.getAddressSet().contains(packingAddress)) {
                    container.getAddressSet().remove(packingAddress);
                }
            }
            if (container.getStatus() == ProtocolContainer.DELAY_LOCK) {
                //
                container.setCurrentDelay(container.getCurrentDelay() + 1);

                //
                if (container.getCurrentDelay() >= container.getDelay()) {
                    container.setStatus(ProtocolContainer.VALID);
                    container.setEffectiveHeight(blockHeader.getHeight() + 1);
                    upgradeProtocol(container);
                    clearIncompatibleTx();
                    Log.info("**********  **********");
                    Log.info("**********  **********");
                    Log.info("**********  **********");
                    Log.info("**********  **********");
                    Log.info("********** version" + container.getVersion());
                    Log.info("********** " + container.getEffectiveHeight());
                    Log.info("********** " + container.getStatus());
                    Log.info("********** " + container.getCurrentDelay());
                    Log.info("********** " + container.getRoundIndex());
                    Log.info("********** AddressSet" +
Arrays.toString(container.getAddressSet().toArray()));
                } else {
                    Log.info("==========  ==========");
                    Log.info("========== version" + container.getVersion());
                    Log.info("========== " + blockHeader.getHeight());
                    Log.info("========== hash" + blockHeader.getHash());
                    Log.info("========== " + container.getStatus());
                    Log.info("========== " + container.getCurrentDelay());
                    Log.info("========== " + container.getRoundIndex());
                    Log.info("========== AddressSet" +
```

```
Arrays.toString(container.getAddressSet().toArray()));
            }
            saveProtocolInfo(container);
            saveBlockProtocolInfo(blockHeader, container);
        }
    }
}

    private void calcTempDelay(BlockHeader blockHeader, BlockExtendsData extendsData) {
        for (ProtocolTempInfoPo tempInfoPo :
getVersionManagerStorageService().getProtocolTempMap().values()) {
            if (extendsData.getCurrentVersion() <= NulsVersionManager.getMainVersion()) {
                continue;
            }
            if (tempInfoPo.getVersion() != extendsData.getCurrentVersion().intValue()
                    && tempInfoPo.getStatus() != ProtocolContainer.VALID) {
                String packingAddress =
AddressTool.getStringAddressByBytes(blockHeader.getPackingAddress());
                if (tempInfoPo.getAddressSet().contains(packingAddress)) {
                    tempInfoPo.getAddressSet().remove(packingAddress);
                }
            }
            if (tempInfoPo.getStatus() == ProtocolContainer.DELAY_LOCK) {
                //
                tempInfoPo.setCurrentDelay(tempInfoPo.getCurrentDelay() + 1);

                //
                if (tempInfoPo.getCurrentDelay() >= tempInfoPo.getDelay()) {
                    tempInfoPo.setStatus(ProtocolContainer.VALID);
                    tempInfoPo.setEffectiveHeight(blockHeader.getHeight() + 1);
                    getVersionManagerStorageService().saveProtocolTempInfoPo(tempInfoPo);
                    Log.info("**********  **********");
                    Log.info("********** version" + tempInfoPo.getVersion());
                    Log.info("********** " + tempInfoPo.getEffectiveHeight());
                    Log.info("********** " + tempInfoPo.getStatus());
                    Log.info("********** " + tempInfoPo.getCurrentDelay());
                    Log.info("********** " + tempInfoPo.getRoundIndex());
                    Log.info("********** AddressSet" +
Arrays.toString(tempInfoPo.getAddressSet().toArray()));
                    //upgradeProtocol(container, blockHeader);
                    //linuxtrue
                    if (System.getProperties().getProperty("os.name").toUpperCase().indexOf("LINUX") !=
```

```
-1) {
                    saveBlockTempProtocolInfo(blockHeader, tempInfoPo);
                    Log.error(">>>>>> The new protocol version has taken effect, this program version is too low has stopped automatically, please upgrade immediately **********");
                    Log.error(">>>>>> The new protocol version has taken effect, this program version is too low has stopped automatically, please upgrade immediately **********");
                    NulsContext.getInstance().exit(1);
                } else {
                    NulsContext.mastUpGrade = true;
                }
            } else {
                getVersionManagerStorageService().saveProtocolTempInfoPo(tempInfoPo);
                Log.info("========== Temp  ==========");
                Log.info("========== version" + tempInfoPo.getVersion());
                Log.info("========== " + blockHeader.getHeight());
                Log.info("========== hash" + blockHeader.getHash());
                Log.info("========== " + tempInfoPo.getStatus());
                Log.info("========== " + tempInfoPo.getCurrentDelay());
                Log.info("========== " + tempInfoPo.getRoundIndex());
                Log.info("========== AddressSet" + Arrays.toString(tempInfoPo.getAddressSet().toArray()));
            }
        }
        saveBlockTempProtocolInfo(blockHeader, tempInfoPo);
    }
}

/**
 *
 *
 * @param container
 */
private void upgradeProtocol(ProtocolContainer container) {
    NulsContext.MAIN_NET_VERSION = container.getVersion();
    getVersionManagerStorageService().saveMainVersion(container.getVersion());
    if (container.getVersion() == 2) {
        getVersionManagerStorageService().saveChangeTxHashBlockHeight(container.getEffectiveHeight());
        NulsContext.CHANGE_HASH_SERIALIZE_HEIGHT = container.getEffectiveHeight();
    }
}
```

```
    /**
     *
     *
     * @param tempInfoPo
     * @return
     */
    private int calcRate(ProtocolTempInfoPo tempInfoPo, BlockExtendsData extendsData) {
        int memberCount = extendsData.getConsensusMemberCount();
        int addressCount = tempInfoPo.getAddressSet().size();
        return calcRate(addressCount, memberCount);
    }

    private int calcRate(ProtocolContainer protocolContainer, BlockExtendsData extendsData) {
        int memberCount = extendsData.getConsensusMemberCount();
        int addressCount = protocolContainer.getAddressSet().size();

        return calcRate(addressCount, memberCount);
    }

    private int calcRate(int addressCount, int memeberCount) {
        BigDecimal b1 = new BigDecimal(addressCount);
        BigDecimal b2 = new BigDecimal(memeberCount);
        int rate = b1.divide(b2, 2, BigDecimal.ROUND_DOWN).movePointRight(2).intValue();
        return rate;
    }

    private void saveProtocolInfo(ProtocolContainer container) {
        ProtocolInfoPo infoPo = ProtocolTransferTool.toProtocolInfoPo(container);
        getVersionManagerStorageService().saveProtocolInfoPo(infoPo);
    }

    private void saveBlockProtocolInfo(BlockHeader blockHeader, ProtocolContainer container) {
        BlockProtocolInfoPo infoPo = ProtocolTransferTool.toBlockProtocolInfoPo(blockHeader,
container);
        getVersionManagerStorageService().saveBlockProtocolInfoPo(infoPo);
    }

    private void saveBlockTempProtocolInfo(BlockHeader blockHeader, ProtocolTempInfoPo
tempInfoPo) {
        BlockProtocolInfoPo infoPo = ProtocolTransferTool.toBlockProtocolInfoPo(blockHeader,
tempInfoPo);
        getVersionManagerStorageService().saveBlockProtocolTempInfoPo(infoPo);
```

```
    }

    private VersionManagerStorageService getVersionManagerStorageService() {
        if (versionManagerStorageService == null) {
            versionManagerStorageService =
NulsContext.getServiceBean(VersionManagerStorageService.class);
        }
        return versionManagerStorageService;
    }

    private BlockService getBlockService() {
        if (null == blockService) {
            blockService = NulsContext.getServiceBean(BlockService.class);
        }
        return blockService;
    }

    public void processProtocolRollback(BlockHeader blockHeader) {
        BlockExtendsData extendsData = new BlockExtendsData(blockHeader.getExtend());
        //
        if (extendsData.getCurrentVersion() == null) {
            return;
        }
        //
        ProtocolContainer protocolContainer =
NulsVersionManager.getProtocolContainer(extendsData.getCurrentVersion());
        if (protocolContainer != null) {
            if (protocolContainer.getStatus() == ProtocolContainer.VALID &&
protocolContainer.getEffectiveHeight() < blockHeader.getHeight()) {
                //block
                return;
            }
            //
            List<Long> blockHeightIndex =
getVersionManagerStorageService().getBlockProtocolIndex(protocolContainer.getVersion());
            //1container
            if (blockHeightIndex == null || blockHeightIndex.size() == 1) {
                protocolContainer.reset();
                getVersionManagerStorageService().clearBlockProtocol(blockHeader.getHeight(),
protocolContainer.getVersion());
            } else {
                if (blockHeader.getHeight() == blockHeightIndex.get(blockHeightIndex.size() - 1)) {
```

```
                    blockHeightIndex.remove(blockHeightIndex.size() - 1);
getVersionManagerStorageService().saveBlockProtocolIndex(protocolContainer.getVersion(), bloc
kHeightIndex);
                    getVersionManagerStorageService().deleteBlockProtocol(blockHeader.getHeight());
                    BlockProtocolInfoPo blockProtocolInfoPo =
getVersionManagerStorageService().getBlockProtocolInfoPo(blockHeightIndex.get(blockHeightInd
ex.size() - 1));
                    if (blockProtocolInfoPo != null) {
                        ProtocolTransferTool.copyFromBlockProtocolInfoPo(blockProtocolInfoPo,
protocolContainer);
                    }
                    /**
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@   */
                    Log.info("@@@@@@@@@@@@@@@   @@@@@@@@@@@@@@@");
                    Log.info("@@@@@@@ version" + protocolContainer.getVersion());
                    Log.info("@@@@@@@ " + blockHeader.getHeight());
                    Log.info("@@@@@@@ hash" + blockHeader.getHash());
                    Log.info("@@@@@@@ " + blockProtocolInfoPo.getBlockHeight());
                }
            }
            //2
            if (protocolContainer.getVersion() == 2 && protocolContainer.getStatus() !=
ProtocolContainer.VALID) {
                getVersionManagerStorageService().deleteChangeTxHashBlockHeight();
                NulsContext.CHANGE_HASH_SERIALIZE_HEIGHT = null;
            }
            rollbackMainVersion();
            saveProtocolInfo(protocolContainer);

            Log.info("@@@@@@@ " + protocolContainer.getStatus());
            Log.info("@@@@@@@ " + protocolContainer.getCurrentDelay());
            Log.info("@@@@@@@ " + protocolContainer.getRoundIndex());
            Log.info("@@@@@@@ AddressSet" +
Arrays.toString(protocolContainer.getAddressSet().toArray()));
            /**
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@   */
        } else {
            ProtocolTempInfoPo protocolTempInfoPo =
getVersionManagerStorageService().getProtocolTempInfoPo(extendsData.getProtocolKey());
            if (protocolTempInfoPo != null) {
```

```
            if (protocolTempInfoPo.getStatus() == ProtocolContainer.VALID &&
protocolTempInfoPo.getEffectiveHeight() < blockHeader.getHeight()) {
                //block
                return;
            }
            //
            List<Long> blockHeightIndex =
getVersionManagerStorageService().getBlockTempProtocolIndex(protocolTempInfoPo.getVersion
());
            //1container
            if (blockHeightIndex == null || blockHeightIndex.size() == 1) {
                protocolTempInfoPo.reset();
getVersionManagerStorageService().clearTempBlockProtocol(blockHeader.getHeight(), protocolT
empInfoPo.getVersion());
            } else {
                if (blockHeader.getHeight() == blockHeightIndex.get(blockHeightIndex.size() - 1)) {
                    blockHeightIndex.remove(blockHeightIndex.size() - 1);
getVersionManagerStorageService().saveTempBlockProtocolIndex(protocolTempInfoPo.getVersio
n(), blockHeightIndex);
getVersionManagerStorageService().deleteBlockTempProtocol(blockHeader.getHeight());
        BlockProtocolInfoPo blockProtocolInfoPo =
getVersionManagerStorageService().getBlockTempProtocolInfoPo(blockHeightIndex.get(blockHei
ghtIndex.size() - 1));
                    if (blockProtocolInfoPo != null) {
                        ProtocolTransferTool.copyFromBlockProtocolTempInfoPo(blockProtocolInfoPo,
protocolTempInfoPo);
                    }
                    /**
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@   */
                    Log.info("@@@@@@@@@@@@@@@@  Temp
@@@@@@@@@@@@@@");
                    Log.info("@@@@@@@ version" + protocolTempInfoPo.getVersion());
                    Log.info("@@@@@@@ " + blockHeader.getHeight());
                    Log.info("@@@@@@@ hash" + blockHeader.getHash());
                    Log.info("@@@@@@@ " + blockProtocolInfoPo.getBlockHeight());
                }

            }
            getVersionManagerStorageService().saveProtocolTempInfoPo(protocolTempInfoPo);
        }
        Log.info("@@@@@@@ " + protocolTempInfoPo.getStatus());
```

```
            Log.info("@@@@@@@ " + protocolTempInfoPo.getCurrentDelay());
            Log.info("@@@@@@@ " + protocolTempInfoPo.getRoundIndex());
            Log.info("@@@@@@@ AddressSet" +
Arrays.toString(protocolTempInfoPo.getAddressSet().toArray()));
            /**
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@   */
        }
    }

    private void rollbackMainVersion() {
        int validVersion = 1;
        for (ProtocolContainer container : NulsVersionManager.getAllProtocolContainers().values()) {
            if (container.getStatus() == ProtocolContainer.VALID) {
                if (container.getVersion() > validVersion) {
                    validVersion = container.getVersion();
                }
            }
        }
        NulsContext.MAIN_NET_VERSION = validVersion;
        getVersionManagerStorageService().saveMainVersion(validVersion);
    }

    public void clearIncompatibleTx() {
        TransactionQueueStorageService tqs =
NulsContext.getServiceBean(TransactionQueueStorageService.class);
        while (tqs.pollTx() != null) {
        }
        TransactionCacheStorageService tcs =
NulsContext.getServiceBean(TransactionCacheStorageService.class);
        Transaction tx = null;
        while ((tx = tcs.pollTx()) != null) {
            tcs.removeTx(tx.getHash());
        }
        TxMemoryPool.getInstance().clear();
    }

}
```

35:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\process\OrphanBlockProcess.java

 *

```
 */

package io.nuls.consensus.poc.process;

import io.nuls.consensus.poc.constant.ConsensusStatus;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.container.BlockContainer;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.consensus.poc.manager.ChainManager;
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.constant.BlockContainerStatus;
import io.nuls.consensus.poc.container.ChainContainer;
import io.nuls.consensus.poc.provider.OrphanBlockProvider;
import io.nuls.core.tools.log.ChainLog;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.model.Node;
import io.nuls.network.service.NetworkService;
import io.nuls.protocol.service.DownloadService;

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

/**
 * @author ln
 */
public class OrphanBlockProcess implements Runnable {

    private DownloadService downloadService =
NulsContext.getServiceBean(DownloadService.class);
    private NetworkService networkService = NulsContext.getServiceBean(NetworkService.class);

    private ChainManager chainManager;
    private OrphanBlockProvider orphanBlockProvider;

    private boolean running = true;

    public OrphanBlockProcess(ChainManager chainManager, OrphanBlockProvider
```

```
orphanBlockProvider) {
        this.chainManager = chainManager;
        this.orphanBlockProvider = orphanBlockProvider;
    }

    public void start() {
        TaskManager.createAndRunThread(ConsensusConstant.MODULE_ID_CONSENSUS,
"process-orphan-thread", this);
    }

    @Override
    public void run() {
        while (running) {
            try {
                process();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                Thread.sleep(1000L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public void process() throws IOException {
        BlockContainer blockContainer;
        while ((blockContainer = orphanBlockProvider.get()) != null) {
            process(blockContainer);
        }
    }

    public void process(BlockContainer blockContainer) throws IOException {
        if (ConsensusStatusContext.getConsensusStatus().ordinal() <
ConsensusStatus.RUNNING.ordinal()) {
            return;
        }
        Block block = blockContainer.getBlock();

        //
        long bestBlockHeight = chainManager.getBestBlockHeight();
```

```
        if (Math.abs(bestBlockHeight - block.getHeader().getHeight()) >
PocConsensusConstant.MAX_ISOLATED_BLOCK_COUNT) {
            return;
        }

        // Because it is not possible to ensure that there will be repeated reception, priority is given to
        //
        boolean hasExist = checkHasExist(block.getHeader().getHash());
        if (hasExist) {
            return;
        }

        ChainLog.debug("process isolated block, bestblockheight:{}, isolated {} - {}", bestBlockHeight,
block.getHeader().getHeight(), block.getHeader().getHash().getDigestHex());

        // Checks if the current orphaned block is connected to an existing orphaned chain
        //
        boolean success = chainManager.checkIsBeforeOrphanChainAndAdd(block);

        ChainLog.debug("checkIsBeforeIsolatedChainAndAdd: {} , block {} - {}", success,
block.getHeader().getHeight(), block.getHeader().getHash().getDigestHex());

        if (success) {
            // Connect to the current isolated chain and continue to find the previous block
            //
            foundAndProcessPreviousBlock(blockContainer);
            return;
        }
        success = chainManager.checkIsAfterOrphanChainAndAdd(block);

        ChainLog.debug("checkIsAfterIsolatedChainAndAdd: {} , block {} - {}", success,
block.getHeader().getHeight(), block.getHeader().getHash().getDigestHex());

        if (success) {
            // Successfully found and connected, no action required
            //
            return;
        }

        // Not found, then create a new isolated chain, then find the previous block
        //
```

```
        chainManager.newOrphanChain(block);

        ChainLog.debug("new a isolated chain , block {} - {}", success, block.getHeader().getHeight(),
block.getHeader().getHash().getDigestHex());

        foundAndProcessPreviousBlock(blockContainer);
    }

    /*
     * Check the existence of this block from the forked chain and the isolated chain
     *
     */
    private boolean checkHasExist(NulsDigestData blockHash) {

        for (Iterator<ChainContainer> it = chainManager.getOrphanChains().iterator() ; it.hasNext() ; )
{
            ChainContainer chainContainer = it.next();
            for (BlockHeader header : chainContainer.getChain().getBlockHeaderList()) {
                if (header.getHash().equals(blockHash)) {
                    return true;
                }
            }
        }

        for (Iterator<ChainContainer> it = chainManager.getChains().iterator() ; it.hasNext() ; ) {
            ChainContainer chainContainer = it.next();
            for (BlockHeader header : chainContainer.getChain().getBlockHeaderList()) {
                if (header.getHash().equals(blockHash)) {
                    return true;
                }
            }
        }

        List<BlockHeader> masterChainBlockHeaderList =
chainManager.getMasterChain().getChain().getBlockHeaderList();
        int size = (int) (masterChainBlockHeaderList.size() -
PocConsensusConstant.MAX_ISOLATED_BLOCK_COUNT * 1.05);
        if (size < 0) {
            size = 0;
        }
        for (int i = masterChainBlockHeaderList.size() - 1; i >= size; i--) {
            if (blockHash.equals(masterChainBlockHeaderList.get(i).getHash())) {
```

```
                return true;
            }
        }

        return false;
    }

    private void foundAndProcessPreviousBlock(BlockContainer blockContainer) {

        BlockHeader blockHeader = blockContainer.getBlock().getHeader();

        // Determine whether the previous block already exists. If it already exists, it will not be
downloaded.
        //
        boolean hasExist = checkHasExist(blockHeader.getPreHash());
        if (hasExist) {
            return;
        }

        Block preBlock = downloadService.downloadBlock(blockHeader.getPreHash(),
blockContainer.getNode()).getData();

        if (preBlock != null) {
            ChainLog.debug("get pre block success {} - {}", preBlock.getHeader().getHeight(),
preBlock.getHeader().getHash());
            orphanBlockProvider.addBlock(new BlockContainer(preBlock, blockContainer.getNode(),
BlockContainerStatus.DOWNLOADING));
        } else {
            ChainLog.debug("get pre block fail {} - {}", blockHeader.getHeight() - 1,
blockHeader.getPreHash());

            //
            for (Node node : networkService.getAvailableNodes()) {
                preBlock = downloadService.downloadBlock(blockHeader.getPreHash(),
node).getData();
                if (preBlock != null) {
                    orphanBlockProvider.addBlock(new BlockContainer(preBlock, node,
BlockContainerStatus.DOWNLOADING));
                    ChainLog.debug("get pre block retry success {} - {}",
preBlock.getHeader().getHeight() - 1, preBlock.getHeader().getPreHash());
                    return;
                }
```

```
            }
            ChainLog.debug("get pre block complete failure {} - {}", blockHeader.getHeight() - 1,
blockHeader.getPreHash());
        }
    }

    public void stop() {
        running = false;
    }
}
```

36:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\process\RewardStatisticsProcess.java
*/

```
package io.nuls.consensus.poc.process;

import io.nuls.account.service.AccountService;
import io.nuls.consensus.poc.model.RewardStatisticsParam;
import io.nuls.consensus.poc.service.impl.PocRewardCacheService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Block;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.DownloadService;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingDeque;

/**
 * @author: Niels Wang
 */
@Component
public class RewardStatisticsProcess {

    @Autowired
    private DownloadService downloadService;
```

```
    @Autowired
    private BlockService blockService;

    @Autowired
    private AccountService accountService;

    private static BlockingQueue<RewardStatisticsParam> queue = new
LinkedBlockingDeque<>(1000);

    @Autowired
    private PocRewardCacheService service;

    private void initProcess() {
        if (downloadService.isDownloadSuccess().isFailed()) {
            try {
                Thread.sleep(ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS);
            } catch (InterruptedException e) {
                Log.error(e);
            }
            initProcess();
            return;
        }
        service.initCache();



    }

    public void doProcess() {
        this.initProcess();
        while (true) {
            try {
                RewardStatisticsParam param = queue.take();
                if (param.getType() == 0) {
                    service.addBlock(param.getBlock());
                } else if (param.getType() == 1) {
                    service.rollback(param.getBlock());
                }
            } catch (Exception e) {
                Log.error(e);
            }
        }
    }
```

```
    public static void addBlock(Block block) {
        if (NulsContext.getServiceBean(DownloadService.class).isDownloadSuccess().isFailed()) {
            return;
        }
        queue.add(new RewardStatisticsParam(0, block));
    }

    public static void rollbackBlock(Block block) {
        queue.add(new RewardStatisticsParam(1, block));
    }

    public void calculate() {
        service.calcRewards();
    }
}
```

37:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\provider\BlockQueueProvider.java

```
 *
 */

package io.nuls.consensus.poc.provider;

import io.nuls.consensus.poc.constant.BlockContainerStatus;
import io.nuls.consensus.poc.container.BlockContainer;
import io.nuls.kernel.context.NulsContext;
import io.nuls.protocol.service.DownloadService;

import java.util.Queue;
import java.util.concurrent.LinkedBlockingDeque;

/**
 * @author ln
 */
public class BlockQueueProvider {

    private final static BlockQueueProvider INSTANCE = new BlockQueueProvider();

    private Queue<BlockContainer> blockQueue;
    private Queue<BlockContainer> downloadBlockQueue;
```

```
private DownloadService downloadService;

private boolean downloadBlockQueueHasDestory;

private BlockQueueProvider() {
    blockQueue = new LinkedBlockingDeque<>();
    downloadBlockQueue = new LinkedBlockingDeque<>();
    initDownloadQueue();
}

public static BlockQueueProvider getInstance() {
    return INSTANCE;
}

public boolean put(BlockContainer blockContainer) {

    boolean receive = (blockContainer.getStatus() == BlockContainerStatus.RECEIVED);

    if (receive) {
        int status = BlockContainerStatus.RECEIVED;
        checkDownloadService();
        if (!downloadService.isDownloadSuccess().isSuccess()) {
            status = BlockContainerStatus.DOWNLOADING;
        }
        blockContainer.setStatus(status);

        blockQueue.offer(blockContainer);
    } else {
        if (downloadBlockQueueHasDestory) {
            initDownloadQueue();
        }
        downloadBlockQueue.offer(blockContainer);
    }
    return true;
}

private void checkDownloadService() {
    if (null == downloadService) {
        downloadService = NulsContext.getServiceBean(DownloadService.class);
    }
}
```

```
    public BlockContainer get() {

        BlockContainer blockContainer = null;

        //check can destory the download queue
        if (!downloadBlockQueueHasDestory) {
            blockContainer = downloadBlockQueue.poll();
        }
        checkDownloadService();
        boolean hasDownloadSuccess = downloadService.isDownloadSuccess().isSuccess();
        if (blockContainer == null && hasDownloadSuccess && !downloadBlockQueueHasDestory) {
            downloadBlockQueueHasDestory = true;
            if (blockContainer == null) {
                blockContainer = blockQueue.poll();
            }
        } else if (hasDownloadSuccess && blockContainer == null) {
            blockContainer = blockQueue.poll();
        }
        return blockContainer;
    }

    public long size() {
        long size = blockQueue.size();

        if (!downloadBlockQueueHasDestory) {
            size += downloadBlockQueue.size();
        }

        return size;
    }

    public void clear() {
        blockQueue.clear();
        downloadBlockQueue.clear();
        downloadBlockQueueHasDestory = false;
    }

    private void initDownloadQueue() {
        downloadBlockQueueHasDestory = false;
    }
}
```

38:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\provider\OrphanBlockProvider.java

```
 *
 */

package io.nuls.consensus.poc.provider;

import io.nuls.consensus.poc.container.BlockContainer;

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

/**
 *
 * @author ln
 */
public class OrphanBlockProvider {

    // Orphaned block caching, isolated block refers to the case where the previous block was not found
    //
    private List<BlockContainer> orphanBlockList = new ArrayList<BlockContainer>();

    public boolean addBlock(BlockContainer block) {
        return orphanBlockList.add(block);
    }

    public BlockContainer get() {
        if(orphanBlockList == null || orphanBlockList.size() == 0) {
            return null;
        }
        return orphanBlockList.remove(0);
    }

    public int size() {
        return orphanBlockList.size();
    }
}
```

39:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\scheduler\ConsensusScheduler.java

```
 *
 */

package io.nuls.consensus.poc.scheduler;

import io.nuls.consensus.poc.constant.ConsensusStatus;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.manager.CacheManager;
import io.nuls.consensus.poc.manager.ChainManager;
import io.nuls.consensus.poc.process.*;
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.provider.OrphanBlockProvider;
import io.nuls.consensus.poc.task.*;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.service.BlockService;

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

/**
 * @author ln
 */
public class ConsensusScheduler {

    private static ConsensusScheduler INSTANCE = new ConsensusScheduler();

    private ScheduledThreadPoolExecutor threadPool;

    private OrphanBlockProcess orphanBlockProcess;

    private CacheManager cacheManager;

    private ConsensusScheduler() {
```

```
    }

    public static ConsensusScheduler getInstance() {
        return INSTANCE;
    }

    public boolean start() {

        ChainManager chainManager = new ChainManager();
        OrphanBlockProvider orphanBlockProvider = new OrphanBlockProvider();

        PocConsensusContext.setChainManager(chainManager);

        cacheManager = new CacheManager(chainManager);
        try {
            initDatas();
        } catch (Exception e) {
            Log.warn(e.getMessage());
        }

        threadPool = TaskManager.createScheduledThreadPool(6,
                new NulsThreadFactory(ConsensusConstant.MODULE_ID_CONSENSUS, "consensus-poll-control"));

        BlockProcess blockProcess = new BlockProcess(chainManager, orphanBlockProvider);
        threadPool.scheduleAtFixedRate(new BlockProcessTask(blockProcess), 1000L, 300L, TimeUnit.MILLISECONDS);

        ForkChainProcess forkChainProcess = new ForkChainProcess(chainManager);
        threadPool.scheduleAtFixedRate(new ForkChainProcessTask(forkChainProcess), 1000L, 1000L, TimeUnit.MILLISECONDS);

        ConsensusProcess consensusProcess = new ConsensusProcess(chainManager);
        threadPool.scheduleAtFixedRate(new ConsensusProcessTask(consensusProcess), 1000L, 1000L, TimeUnit.MILLISECONDS);

        orphanBlockProcess = new OrphanBlockProcess(chainManager, orphanBlockProvider);
        orphanBlockProcess.start();

        threadPool.scheduleAtFixedRate(new BlockMonitorProcessTask(new BlockMonitorProcess(chainManager)), 60, 60, TimeUnit.SECONDS);
```

```
        TaskManager.createAndRunThread(ConsensusConstant.MODULE_ID_CONSENSUS, "poc-reward-cache", new RewardStatisticsProcessTask(NulsContext.getServiceBean(RewardStatisticsProcess.class)));

        threadPool.scheduleAtFixedRate(new RewardCalculatorTask(NulsContext.getServiceBean(RewardStatisticsProcess.class)), ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND, ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND, TimeUnit.SECONDS);

        threadPool.scheduleAtFixedRate(new TxProcessTask(), 5, 1, TimeUnit.SECONDS);
        return true;
    }

    public boolean restart() {
        clear();
        initDatas();

        return true;
    }

    public boolean stop() {

        clear();

        orphanBlockProcess.stop();
        threadPool.shutdown();

        return true;
    }

    private void initDatas() {
        try {
            ConsensusStatusContext.setConsensusStatus(ConsensusStatus.LOADING_CACHE);
            cacheManager.load();

            ConsensusStatusContext.setConsensusStatus(ConsensusStatus.WAIT_RUNNING);
        } catch (Exception e) {
            throw new NulsRuntimeException(e);
        }
    }

    private void clear() {
```

```
        cacheManager.clear();
    }

}

40:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\service\impl\ConsensusPocServiceImpl.java
 *
 */

package io.nuls.consensus.poc.service.impl;

import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.cache.TxMemoryPool;
import io.nuls.consensus.poc.constant.BlockContainerStatus;
import io.nuls.consensus.poc.container.BlockContainer;
import io.nuls.consensus.poc.locker.Lockers;
import io.nuls.consensus.poc.process.NulsProtocolProcess;
import io.nuls.consensus.poc.process.RewardStatisticsProcess;
import io.nuls.consensus.poc.provider.BlockQueueProvider;
import io.nuls.consensus.poc.scheduler.ConsensusScheduler;
import io.nuls.consensus.poc.storage.service.TransactionCacheStorageService;
import io.nuls.consensus.poc.storage.service.TransactionQueueStorageService;
import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.*;
import io.nuls.ledger.service.LedgerService;
import io.nuls.network.model.Node;
import io.nuls.network.service.NetworkService;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.DownloadService;

import java.util.List;

/**
 * @author ln
 */
@Service
```

```
public class ConsensusPocServiceImpl implements ConsensusService {

    private TxMemoryPool txMemoryPool = TxMemoryPool.getInstance();

    private BlockQueueProvider blockQueueProvider = BlockQueueProvider.getInstance();

    private NulsProtocolProcess nulsProtocolProcess = NulsProtocolProcess.getInstance();

    @Autowired
    private BlockService blockService;
    @Autowired
    private LedgerService ledgerService;
    @Autowired
    private TransactionQueueStorageService transactionQueueStorageService;
    @Autowired
    private TransactionCacheStorageService transactionCacheStorageService;

    @Override
    public Result newTx(Transaction<? extends BaseNulsData> tx) {
        // Validate the transaction. If the verification is passed, it will be put into the transaction
memory pool.
        // If the verification is an isolated transaction, it will be put in the isolated transaction pool.
Other failures will directly discard the transaction.
        //
//        ValidateResult verifyResult = tx.verify();
//        boolean success = false;
//        if (verifyResult.isSuccess()) {
//            success = txMemoryPool.add(new TxContainer(tx), false);
//        } else if (verifyResult.isFailed() && TransactionErrorCode.ORPHAN_TX ==
verifyResult.getErrorCode()) {
//            success = txMemoryPool.add(new TxContainer(tx), true);
//        }

//        boolean success = txMemoryPool.add(new TxContainer(tx), false);
        boolean success =  transactionQueueStorageService.putTx(tx);
        return new Result(success, null);
    }

    @Override
    public Result newBlock(Block block) {
        return newBlock(block, null);
    }
```

```
    @Override
    public Result newBlock(Block block, Node node) {
        BlockContainer blockContainer = new BlockContainer(block, node,
BlockContainerStatus.RECEIVED);
        boolean success = blockQueueProvider.put(blockContainer);
        return new Result(success, null);
    }

    @Override
    public Result addBlock(Block block) {
        BlockContainer blockContainer = new BlockContainer(block,
BlockContainerStatus.DOWNLOADING);
        boolean success = blockQueueProvider.put(blockContainer);
        return new Result(success, null);
    }

    @Override
    public Result rollbackBlock(Block block) throws NulsException {

        boolean success;
        Lockers.CHAIN_LOCK.lock();
        try {
            success = PocConsensusContext.getChainManager().getMasterChain().rollback(block);
        } finally {
            Lockers.CHAIN_LOCK.unlock();
        }
        if (success) {
            success = blockService.rollbackBlock(block).isSuccess();
            if (!success) {
                PocConsensusContext.getChainManager().getMasterChain().addBlock(block);
            } else {
                //
                nulsProtocolProcess.processProtocolRollback(block.getHeader());
                RewardStatisticsProcess.rollbackBlock(block);
NulsContext.getInstance().setBestBlock(PocConsensusContext.getChainManager().getMasterCha
in().getBestBlock());
            }
        }
        return new Result(success, null);
    }
```

```
    @Override
    public List<Transaction> getMemoryTxs() {
        return txMemoryPool.getAll();
    }

    @Override
    public Transaction getTx(NulsDigestData hash) {
        Transaction tx = transactionCacheStorageService.getTx(hash);
        if(tx == null) {
            tx = ledgerService.getTx(hash);
        }
        return tx;
    }

    /**
     * Reset consensus module, restart, load memory data, reinitialize all states
     * <p>
     *
     *
     * @return Result
     */
    public Result reset() {
        Log.warn("Consensus restart...");
        boolean success = ConsensusScheduler.getInstance().restart();
        if (success) {
            NulsContext.getServiceBean(NetworkService.class).reset();
            NulsContext.getServiceBean(DownloadService.class).reset();
        }
        return new Result(success, null);
    }
}

41:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\service\impl\PocRewardCacheService.java
 */

package io.nuls.consensus.poc.service.impl;

import io.nuls.account.ledger.model.TransactionInfo;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.Account;
import io.nuls.account.service.AccountService;
```

```
import io.nuls.consensus.poc.model.RewardItem;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.tx.CoinBaseTransaction;
import io.nuls.protocol.service.BlockService;

import java.util.*;

/**
 * @author: Niels Wang
 */
@Component
public class PocRewardCacheService {

    private long endHeight;
    private long totalRewardHeight = Long.MAX_VALUE;
    private Na totalReward = Na.ZERO;
    private Map<String, Na> totalMap = new HashMap<>();

    private Na todayReward = Na.ZERO;
    private Map<String, Na> todayMap = new HashMap<>();

    private Map<String, Map<Long, RewardItem>> todayRewardMap = new HashMap<>();

    @Autowired
    private AccountLedgerService accountLedgerService;
    @Autowired
    private AccountService accountService;
    @Autowired
    private BlockService blockService;
    @Autowired
    private LedgerService ledgerService;
```

```
    public void initCache() {
        Collection<Account> accountList = accountService.getAccountList().getData();
        if (null == accountList || accountList.isEmpty()) {
            return;
        }

        long startTime = TimeService.currentTimeMillis() - 24 * 3600000L;
        long startHeight = NulsContext.getInstance().getBestHeight() - (24 * 3600 /
ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND);

        if (startHeight <= 0) {
            startHeight = 1;
        }
        long index = startHeight;
        while (true) {
            Block block = blockService.getBlock(index++).getData();
            if (null == block) {
                break;
            }
            if (block.getHeader().getHeight() < 1) {
                break;
            }
            if (block.getHeader().getTime() < startTime) {
                continue;
            }
            this.addBlock(block);
        }
        //
        for (Account account : accountList) {
            List<TransactionInfo> list =
accountLedgerService.getTxInfoList(account.getAddress().getAddressBytes()).getData();
            if (list == null || list.isEmpty()) {
                continue;
            }
            calcRewardHistory(account.getAddress().getBase58(), list, startHeight);
        }

        long totalValue = ledgerService.getWholeUTXO();
        this.totalRewardHeight = NulsContext.getInstance().getBestHeight();
        this.totalReward = Na.valueOf(totalValue - Na.MAX_NA_VALUE);
    }
```

```
private void calcRewardHistory(String address, List<TransactionInfo> list, long startHeight) {
    byte[] addressByte = AddressTool.getAddress(address);
    for (TransactionInfo info : list) {
        if (info.getTxType() != ProtocolConstant.TX_TYPE_COINBASE) {
            continue;
        }
        CoinBaseTransaction tx = (CoinBaseTransaction) ledgerService.getTx(info.getTxHash());
        if (null == tx) {
            continue;
        }
        if (info.getBlockHeight() >= startHeight) {
            continue;
        }
        if (null == tx.getCoinData().getTo() || tx.getCoinData().getTo().isEmpty()) {
            continue;
        }
        for (Coin coin : tx.getCoinData().getTo()) {
            //if (!Arrays.equals(addressByte, coin.()))
            if (!Arrays.equals(addressByte, coin.getAddress()))
            {
                continue;
            }
            Na na = totalMap.get(address);
            if (na == null) {
                na = Na.ZERO;
            }
            na = na.add(coin.getNa());
            totalMap.put(address, na);
        }
    }

}

public void addBlock(Block block) {
    if (block.getHeader().getHeight() <= endHeight) {
        return;
    }

    CoinBaseTransaction tx = (CoinBaseTransaction) block.getTxs().get(0);
    this.calcReward(block.getHeader().getHeight(), tx);
}
```

```
private void calcReward(long height, CoinBaseTransaction tx) {
    if (null != tx.getCoinData().getTo() && !tx.getCoinData().getTo().isEmpty()) {
        long startTime = TimeService.currentTimeMillis() - 24 * 3600000L;
        for (Coin coin : tx.getCoinData().getTo()) {
            addRewardItem(height, tx.getTime(), coin, startTime);
        }
    }
    if (height > endHeight) {
        endHeight = height;
    }

}


private void addRewardItem(long height, long time, Coin coin, long startTime) {
    //String address = AddressTool.getStringAddressByBytes(coin.());
    String address = AddressTool.getStringAddressByBytes(coin.getAddress());
    Map<Long, RewardItem> map = todayRewardMap.get(address);
    if (null == map) {
        map = new HashMap<>();
        todayRewardMap.put(address, map);
    }
    if (time > startTime) {
        map.put(height, new RewardItem(time, coin.getNa()));
    }

    Na tna = todayMap.get(address);
    if (tna == null) {
        tna = Na.ZERO;
    }
    tna = tna.add(coin.getNa());
    todayMap.put(address, tna);
    Na na = totalMap.get(address);
    if (na == null) {
        na = Na.ZERO;
    }
    na = na.add(coin.getNa());
    totalMap.put(address, na);
    if (height > totalRewardHeight) {
        totalReward = totalReward.add(coin.getNa());
    }
```

```
}

public void rollback(Block block) {
    if (block.getHeader().getHeight() > endHeight) {
        return;
    }

    CoinBaseTransaction tx = (CoinBaseTransaction) block.getTxs().get(0);
    if (null != tx.getCoinData().getTo() && !tx.getCoinData().getTo().isEmpty()) {
        for (Coin coin : tx.getCoinData().getTo()) {
            //String address = AddressTool.getStringAddressByBytes(coin.());
            String address = AddressTool.getStringAddressByBytes(coin.getAddress());
            Map<Long, RewardItem> map = todayRewardMap.get(address);
            if (null == map) {
                continue;
            }
            map.remove(block.getHeader().getHeight());

            Na na = totalMap.get(address);
            if (na == null) {
                continue;
            }
            na = na.subtract(coin.getNa());
            todayMap.put(address, na);
        }
    }
    if (endHeight == block.getHeader().getHeight()) {
        endHeight = block.getHeader().getHeight() - 1;
    }
}

public Na getReward(String address) {
    Na na = totalMap.get(address);
    if (null == na) {
        na = Na.ZERO;
    }
    return na;
}

public Na getAllReward() {
    return totalReward;
}
```

```
    public Na getAllRewardToday() {
        return todayReward;
    }

    public Na getRewardToday(String address) {
        Na na = todayMap.get(address);
        if (null == na) {
            na = Na.ZERO;
        }
        return na;
    }

    public void calcRewards() {
        List<String> list = new ArrayList<>(todayRewardMap.keySet());
        Na na = Na.ZERO;
        Map<String, Na> resultMap = new HashMap<>();
        long startTime = TimeService.currentTimeMillis() - 24 * 3600000L;
        for (String address : list) {
            Na reward = Na.ZERO;
            List<RewardItem> rewardItems = new
ArrayList<>(todayRewardMap.get(address).values());
            for (RewardItem item : rewardItems) {
                if (item.getTime() < startTime) {
                    continue;
                }
                reward = reward.add(item.getNa());
            }
            na = na.add(reward);
            resultMap.put(address, reward);
        }
        this.todayReward = na;
        this.todayMap = resultMap;
    }
}

42:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\task\BlockMonitorProcessTask.java
 */

package io.nuls.consensus.poc.task;
```

```
import io.nuls.consensus.poc.process.BlockMonitorProcess;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.protocol.service.DownloadService;

/**
 * @author Niels
 */
public class BlockMonitorProcessTask implements Runnable {

    private final BlockMonitorProcess process;
    private DownloadService downloadService =
NulsContext.getServiceBean(DownloadService.class);

    public BlockMonitorProcessTask(BlockMonitorProcess process) {
        this.process = process;
    }

    @Override
    public void run() {
        try {
            if(!downloadService.isDownloadSuccess().isSuccess()) {
                return;
            }
            process.doProcess();
        } catch (Exception e) {
            Log.error(e);
        }
    }
}

43:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\task\BlockProcessTask.java
 *
 */

package io.nuls.consensus.poc.task;

import io.nuls.consensus.poc.container.BlockContainer;
import io.nuls.consensus.poc.locker.Lockers;
import io.nuls.consensus.poc.process.BlockProcess;
```

```
import io.nuls.consensus.poc.constant.BlockContainerStatus;
import io.nuls.consensus.poc.constant.ConsensusStatus;
import io.nuls.consensus.poc.provider.BlockQueueProvider;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Transaction;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.service.DownloadService;
import io.nuls.protocol.service.TransactionService;

import java.util.List;

/**
 * @author ln
 */
public class BlockProcessTask implements Runnable {

    private DownloadService downloadService =
NulsContext.getServiceBean(DownloadService.class);

    private BlockProcess blockProcess;
    private BlockQueueProvider blockQueueProvider = BlockQueueProvider.getInstance();

    private boolean first = true;

    private LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);

    private TransactionService transactionService =
NulsContext.getServiceBean(TransactionService.class);

    public BlockProcessTask(BlockProcess blockProcess) {
        this.blockProcess = blockProcess;
    }

    @Override
    public void run() {
        Lockers.CHAIN_LOCK.lock();
        try {
            doTask();
        } catch (Exception e) {
            Log.error(e);
```

```
        } catch (Error e) {
            Log.error(e);
        } catch (Throwable e) {
            Log.error(e);
        } finally {
            Lockers.CHAIN_LOCK.unlock();
        }
    }

    private void doTask() {

        //wait consensus ready running
        if (ConsensusStatusContext.getConsensusStatus().ordinal() <=
ConsensusStatus.LOADING_CACHE.ordinal()) {
            return;
        }
        BlockContainer blockContainer;
        while ((blockContainer = blockQueueProvider.get()) != null) {
            try {
                if (first) {
                    List<Transaction> txList = blockContainer.getBlock().getTxs();
                    for (int index = txList.size() - 1; index >= 0; index--) {
                        Transaction tx = blockContainer.getBlock().getTxs().get(index);
                        Transaction localTx = ledgerService.getTx(tx.getHash());

                        if (null == localTx || tx.getBlockHeight() != localTx.getBlockHeight()) {
                            continue;
                        }
                        try {
                            transactionService.rollbackTx(tx,blockContainer.getBlock().getHeader());
                        } catch (Exception e) {
                            Log.error(e);
                        }
                    }
                    first = false;
                }
                //long time = System.currentTimeMillis();
                blockProcess.addBlock(blockContainer);
                //Log.info("add  " + blockContainer.getBlock().getHeader().getHeight() + "  " +
(System.currentTimeMillis() - time) + " ms , tx count : " +
blockContainer.getBlock().getHeader().getTxCount());
            } catch (Exception e) {
```

```
                e.printStackTrace();
                Log.error("add block fail , error : " + e.getMessage(), e);
            }
        }

        // The system starts up. The local height and the network height are the same. When the
block is not to be downloaded, the system needs to know and set the consensus status to running.
        //
        if (downloadService.isDownloadSuccess().isSuccess() &&
ConsensusStatusContext.getConsensusStatus() == ConsensusStatus.WAIT_RUNNING &&
                (blockContainer == null || blockContainer.getStatus() ==
BlockContainerStatus.RECEIVED)) {
            ConsensusStatusContext.setConsensusStatus(ConsensusStatus.RUNNING);
        }
    }
}
```

44:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\task\ConsensusProcessTask.java

```
 *
 */
package io.nuls.consensus.poc.task;

import io.nuls.consensus.poc.process.ConsensusProcess;
import io.nuls.core.tools.log.Log;

/**
 *
 * @author ln
 */
public class ConsensusProcessTask implements Runnable {

    private ConsensusProcess consensusProcess;

    public ConsensusProcessTask(ConsensusProcess consensusProcess) {
        this.consensusProcess = consensusProcess;
    }

    @Override
    public void run() {
        try {
            consensusProcess.process();
```

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

45:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\task\ForkChainProcessTask.java
 *
 */

package io.nuls.consensus.poc.task;

import io.nuls.consensus.poc.process.ForkChainProcess;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;

import java.io.IOException;

/**
 *
 * @author ln
 */
public class ForkChainProcessTask implements Runnable {

    private ForkChainProcess forkChainProcess;

    public ForkChainProcessTask(ForkChainProcess forkChainProcess) {
        this.forkChainProcess = forkChainProcess;
    }

    @Override
    public void run() {
        try {
            doTask();
        } catch (Exception e) {
            Log.error(e);
        }
    }

    private void doTask() throws IOException, NulsException {
        forkChainProcess.doProcess();
```

```
    }
}
```

46:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\task\RewardCalculatorTask.java

```
 */

package io.nuls.consensus.poc.task;

import io.nuls.consensus.poc.process.RewardStatisticsProcess; /**
 * @author: Niels Wang
 */
public class RewardCalculatorTask implements Runnable {

    private final RewardStatisticsProcess process;

    public RewardCalculatorTask(RewardStatisticsProcess process) {
        this.process = process;

    }

    @Override
    public void run() {
        process.calculate();
    }
}
```

47:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\task\RewardStatisticsProcessTask.java

```
 */

package io.nuls.consensus.poc.task;

import io.nuls.consensus.poc.process.RewardStatisticsProcess;
import io.nuls.core.tools.log.Log;

/**
 * @author Niels
 */
public class RewardStatisticsProcessTask implements Runnable {
```

```
    private final RewardStatisticsProcess process;

    public RewardStatisticsProcessTask(RewardStatisticsProcess process) {
        this.process = process;
    }

    @Override
    public void run() {
        try {
            process.doProcess();
        } catch (Exception e) {
            Log.error(e);
        }
    }
}

48:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\task\TxProcessTask.java
 */

package io.nuls.consensus.poc.task;

import io.nuls.consensus.poc.cache.TxMemoryPool;
import io.nuls.consensus.poc.storage.service.TransactionCacheStorageService;
import io.nuls.consensus.poc.storage.service.TransactionQueueStorageService;
import io.nuls.core.tools.log.Log;
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.model.Coin;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;
import io.nuls.ledger.util.LedgerUtil;
import io.nuls.protocol.service.TransactionService;
import io.nuls.protocol.utils.TransactionTimeComparator;

import java.util.*;

/**
```

```
 * @author: Niels Wang
 * @date: 2018/7/5
 */
public class TxProcessTask implements Runnable {

    private TxMemoryPool pool = TxMemoryPool.getInstance();

    private LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);
    private TransactionCacheStorageService transactionCacheStorageService =
NulsContext.getServiceBean(TransactionCacheStorageService.class);
    private TransactionQueueStorageService transactionQueueStorageService =
NulsContext.getServiceBean(TransactionQueueStorageService.class);

    private TransactionService transactionService =
NulsContext.getServiceBean(TransactionService.class);

    private TransactionTimeComparator txComparator =
TransactionTimeComparator.getInstance();

    private Map<String, Coin> temporaryToMap = new HashMap<>();
    private Set<String> temporaryFromSet = new HashSet<>();

    private List<Transaction> orphanTxList = new ArrayList<>();

    private static int maxOrphanSize = 200000;

//    int count = 0;
//    int size = 0;

    @Override
    public void run() {
        try {
            doTask();
        } catch (Exception e) {
            Log.error(e);
        }
        try {
            doOrphanTxTask();
        } catch (Exception e) {
            Log.error(e);
        }
//        System.out.println("count: " + count + " , size : " + size + " , orphan size : " +
```

```
orphanTxList.size());
    }

    private void doTask() {

        if (TxMemoryPool.getInstance().getPoolSize() >= 1000000L) {
            return;
        }

        Transaction tx = null;
        while ((tx = transactionQueueStorageService.pollTx()) != null && orphanTxList.size() <
maxOrphanSize) {
//            size++;
            processTx(tx, false);
        }
    }


    private void doOrphanTxTask() {
        orphanTxList.sort(txComparator);

        Iterator<Transaction> it = orphanTxList.iterator();
        while (it.hasNext()) {
            Transaction tx = it.next();
            boolean success = processTx(tx, true);
            if (success) {
                it.remove();
            }
        }
    }


    private boolean processTx(Transaction tx, boolean isOrphanTx) {
        try {
            Result result = tx.verify();
            if (result.isFailed()) {
                return false;
            }

            Transaction tempTx = ledgerService.getTx(tx.getHash());
            if (tempTx != null) {
                return isOrphanTx;
```

```
            }

            ValidateResult validateResult = ledgerService.verifyCoinData(tx, temporaryToMap,
temporaryFromSet);
            if (validateResult.isSuccess()) {
                pool.add(tx, false);

                List<Coin> fromCoins = tx.getCoinData().getFrom();
                for (Coin coin : fromCoins) {
                    String key = LedgerUtil.asString(coin.getOwner());
                    temporaryFromSet.remove(key);
                    temporaryToMap.remove(key);
                }
//                count++;

                transactionCacheStorageService.putTx(tx);
                transactionService.forwardTx(tx, null);

                return true;
            } else if (validateResult.getErrorCode().equals(TransactionErrorCode.ORPHAN_TX) &&
!isOrphanTx) {
                processOrphanTx(tx);
            } else if (isOrphanTx) {
                return tx.getTime() < (TimeService.currentTimeMillis() - 3600000L);
            }
        } catch (Exception e) {
            Log.error(e);
        }
        return false;
    }

    private void processOrphanTx(Transaction tx) throws NulsException {
        orphanTxList.add(tx);
    }
}

49:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\processor\CancelDepositTxProcessor.java
 */

package io.nuls.consensus.poc.tx.processor;
```

```
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.tx.CancelDepositTransaction;
import io.nuls.consensus.poc.protocol.tx.DepositTransaction;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.protocol.tx.StopAgentTransaction;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.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 io.nuls.ledger.service.LedgerService;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * @author Niels
 */
@Component
public class CancelDepositTxProcessor implements
TransactionProcessor<CancelDepositTransaction> {

    @Autowired
    private DepositStorageService depositStorageService;

    @Autowired
    private AgentStorageService agentStorageService;

    @Autowired
    private LedgerService ledgerService;
```

```
    @Override
    public Result onRollback(CancelDepositTransaction tx, Object secondaryData) {
        DepositTransaction transaction = (DepositTransaction)
ledgerService.getTx(tx.getTxData().getJoinTxHash());
        if (null == transaction) {
            return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
        }
        DepositPo po = depositStorageService.get(tx.getTxData().getJoinTxHash());
        if (null == po) {
            return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND);
        }
        if (po.getDelHeight() != tx.getBlockHeight()) {
            return Result.getFailed(PocConsensusErrorCode.DEPOSIT_NEVER_CANCELED);
        }
        po.setDelHeight(-1L);
        boolean b = depositStorageService.save(po);
        if (b) {
            return Result.getSuccess();
        }
        return Result.getFailed(TransactionErrorCode.ROLLBACK_TRANSACTION_FAILED);
    }

    @Override
    public Result onCommit(CancelDepositTransaction tx, Object secondaryData) {
        DepositTransaction transaction = (DepositTransaction)
ledgerService.getTx(tx.getTxData().getJoinTxHash());
        if (null == transaction) {
            return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST);
        }
        DepositPo po = depositStorageService.get(tx.getTxData().getJoinTxHash());
        if (null == po) {
            return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND);
        }
        tx.getTxData().setAddress(po.getAddress());
        if (po.getDelHeight() > 0L) {
            return Result.getFailed(PocConsensusErrorCode.DEPOSIT_WAS_CANCELED);
        }
        po.setDelHeight(tx.getBlockHeight());
        boolean b = depositStorageService.save(po);
        if (b) {
            return Result.getSuccess();
        }
```

```
        return Result.getFailed(TransactionErrorCode.SAVE_TX_ERROR);
    }

    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        if (txList == null || txList.size() <= 1) {
            return ValidateResult.getSuccessResult();
        }
        Set<NulsDigestData> hashSet = new HashSet<>();
        Set<NulsDigestData> agentHashSet = new HashSet<>();
        Set<String> addressSet = new HashSet<>();

        for (Transaction tx : txList) {
            if (tx.getType() == ConsensusConstant.TX_TYPE_RED_PUNISH) {
                RedPunishTransaction transaction = (RedPunishTransaction) tx;
addressSet.add(AddressTool.getStringAddressByBytes(transaction.getTxData().getAddress()));
       } else if (tx.getType() == ConsensusConstant.TX_TYPE_STOP_AGENT) {
                StopAgentTransaction transaction = (StopAgentTransaction) tx;
                agentHashSet.add(transaction.getTxData().getCreateTxHash());
            }
        }
        for (Transaction tx : txList) {
            if (tx.getType() == ConsensusConstant.TX_TYPE_CANCEL_DEPOSIT) {
                CancelDepositTransaction transaction = (CancelDepositTransaction) tx;
                if (!hashSet.add(transaction.getTxData().getJoinTxHash())) {
                    return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TRANSACTION_REPEATED);
                }
                if (agentHashSet.contains(transaction.getTxData().getJoinTxHash())) {
                    return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_STOPPED);
                }
                DepositPo depositPo =
depositStorageService.get(transaction.getTxData().getJoinTxHash());
                AgentPo agentPo = agentStorageService.get(depositPo.getAgentHash());
                if (null == agentPo) {
                    return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_NOT_EXIST);
                }
                if
(addressSet.contains(AddressTool.getStringAddressByBytes(agentPo.getAgentAddress()))) {
                    return ValidateResult.getFailedResult(this.getClass().getName(),
```

```
PocConsensusErrorCode.AGENT_PUNISHED);
                }
            }
        }

        return ValidateResult.getSuccessResult();
    }
}
```

50:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\tx\processor\CreateAgentTxProcessor.java
*/

```
package io.nuls.consensus.poc.tx.processor;

import io.nuls.account.service.AccountService;
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.protocol.util.PoConvertUtil;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
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.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.processor.TransactionProcessor;
import io.nuls.kernel.validate.ValidateResult;

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

```
import java.util.Set;

/**
 * @author ln
 */
@Component
public class CreateAgentTxProcessor implements
TransactionProcessor<CreateAgentTransaction> {

    @Autowired
    private AgentStorageService agentStorageService;

    @Override
    public Result onRollback(CreateAgentTransaction tx, Object secondaryData) {
        Agent agent = tx.getTxData();
        agent.setTxHash(tx.getHash());

        boolean success = agentStorageService.delete(agent.getTxHash());
        return new Result(success, null);
    }

    @Override
    public Result onCommit(CreateAgentTransaction tx, Object secondaryData) {

        Agent agent = tx.getTxData();
        BlockHeader header = (BlockHeader) secondaryData;
        agent.setTxHash(tx.getHash());
        agent.setBlockHeight(header.getHeight());
        agent.setTime(tx.getTime());

        AgentPo agentPo = PoConvertUtil.agentToPo(agent);

        boolean success = agentStorageService.save(agentPo);

        return new Result(success, null);
    }

    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        // Conflict detection, detecting whether the client and the packager repeat
        //
        if (null == txList || txList.isEmpty()) {
```

```
            return ValidateResult.getSuccessResult();
        }

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

        for (Transaction transaction : txList) {
            if (transaction.getType() == ConsensusConstant.TX_TYPE_REGISTER_AGENT) {
                CreateAgentTransaction createAgentTransaction = (CreateAgentTransaction)
transaction;

                Agent agent = createAgentTransaction.getTxData();

                String agentAddressHex = Hex.encode(agent.getAgentAddress());
                String packAddressHex = Hex.encode(agent.getPackingAddress());

                if (!addressHexSet.add(agentAddressHex) || !addressHexSet.add(packAddressHex)) {
                    return (ValidateResult) ValidateResult.getFailedResult(getClass().getName(),
PocConsensusErrorCode.AGENT_EXIST).setData(transaction);
                }
                break;
            } else if (transaction.getType() == ConsensusConstant.TX_TYPE_RED_PUNISH) {
                RedPunishTransaction redPunishTransaction = (RedPunishTransaction) transaction;
                RedPunishData redPunishData = redPunishTransaction.getTxData();
                String addressHex = Hex.encode(redPunishData.getAddress());
                if (!addressHexSet.add(addressHex)) {
                    return (ValidateResult) ValidateResult.getFailedResult(getClass().getName(),
PocConsensusErrorCode.LACK_OF_CREDIT).setData(transaction);
                }
                break;
            }
        }

        return ValidateResult.getSuccessResult();
    }
}

51:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\processor\DepositTxProcessor.java
 */

package io.nuls.consensus.poc.tx.processor;
```

```
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.consensus.poc.protocol.tx.DepositTransaction;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.protocol.tx.StopAgentTransaction;
import io.nuls.consensus.poc.protocol.util.PoConvertUtil;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
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.*;
import io.nuls.kernel.processor.TransactionProcessor;
import io.nuls.kernel.validate.ValidateResult;

import java.util.*;

/**
 * @author: Niels Wang
 */
@Component
public class DepositTxProcessor implements TransactionProcessor<DepositTransaction> {

    @Autowired
    private DepositStorageService depositStorageService;

    @Autowired
    private AgentStorageService agentStorageService;

    /**
     *
     * This method is called when the transaction rolls back.
     *
     * @param tx            The transaction to roll back.
     * @param secondaryData Secondary data, depending on the business needs to be passed.
```

```
     */
    @Override
    public Result onRollback(DepositTransaction tx, Object secondaryData) {
        Deposit deposit = tx.getTxData();
        deposit.setTxHash(tx.getHash());

        boolean success = depositStorageService.delete(deposit.getTxHash());
        return new Result(success, null);
    }

    /**
     *
     * This method is called when the transaction save.
     *
     * @param tx            The transaction to save;
     * @param secondaryData Secondary data, depending on the business needs to be passed.
     */
    @Override
    public Result onCommit(DepositTransaction tx, Object secondaryData) {
        Deposit deposit = tx.getTxData();
        BlockHeader header = (BlockHeader) secondaryData;
        deposit.setTxHash(tx.getHash());
        deposit.setTime(tx.getTime());
        deposit.setBlockHeight(header.getHeight());

        DepositPo depositPo = PoConvertUtil.depositToPo(deposit);

        boolean success = depositStorageService.save(depositPo);
        return new Result(success, null);
    }

    /**
     *
     *
     * <p>
     * Conflict detection, which detects conflicting transactions in the incoming transaction list, returns failure,
     * indicating the cause of failure and all the list of trades that should be discarded.
     * This method does not check the double flower conflict, the double flower is realized by the accounting interface.
     *
     * @param txList /A list of transactions to be checked.
```

```
     * @return successResultdatamsg
     * Operation result: success returns successResult. When failure, data returns the discard list, and MSG returns the cause of conflict.
     */
    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        if (null == txList || txList.isEmpty()) {
            return ValidateResult.getSuccessResult();
        }

        Set<NulsDigestData> outAgentHash = new HashSet<>();
        Map<NulsDigestData, Na> naMap = new HashMap<>();
        List<DepositTransaction> dTxList = new ArrayList<>();
        for (Transaction transaction : txList) {
            switch (transaction.getType()) {
                case ConsensusConstant.TX_TYPE_STOP_AGENT:
                    StopAgentTransaction stopAgentTransaction = (StopAgentTransaction) transaction;
                    outAgentHash.add(stopAgentTransaction.getTxData().getCreateTxHash());
                    break;
                case ConsensusConstant.TX_TYPE_JOIN_CONSENSUS:
                    DepositTransaction depositTransaction = (DepositTransaction) transaction;
                    Na na = naMap.get(depositTransaction.getTxData().getAgentHash());
                    if (null == na) {
                        na = getAgentTotalDeposit(depositTransaction.getTxData().getAgentHash());
                    }
                    if (na == null) {
                        na = depositTransaction.getTxData().getDeposit();
                    } else {
                        na = na.add(depositTransaction.getTxData().getDeposit());
                    }
                    if (na.isGreaterThan(PocConsensusProtocolConstant.SUM_OF_DEPOSIT_OF_AGENT_UPPER_LIMIT)) {
                        ValidateResult validateResult = ValidateResult.getFailedResult(this.getClass().getName(), PocConsensusErrorCode.DEPOSIT_TOO_MUCH);
                        validateResult.setData(transaction);
                        return validateResult;
                    } else {
                        naMap.put(depositTransaction.getTxData().getAgentHash(), na);
                    }
                    dTxList.add(depositTransaction);
```

```
                break;
            case ConsensusConstant.TX_TYPE_RED_PUNISH:
                RedPunishTransaction redPunishTransaction = (RedPunishTransaction) transaction;
                RedPunishData redPunishData = redPunishTransaction.getTxData();
                AgentPo agent = this.getAgentByAddress(redPunishData.getAddress());
                if (null != agent) {
                    outAgentHash.add(agent.getHash());
                }
                break;
            default:
                continue;
        }
    }

    if (dTxList.isEmpty() || outAgentHash.isEmpty()) {
        return ValidateResult.getSuccessResult();
    }
    for (DepositTransaction depositTransaction : dTxList) {
        if (outAgentHash.contains(depositTransaction.getTxData().getAgentHash())) {
            ValidateResult validateResult =
ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_STOPPED);
            validateResult.setData(depositTransaction);
            return validateResult;
        }
    }
    return ValidateResult.getSuccessResult();
}

private AgentPo getAgentByAddress(byte[] address) {
    List<AgentPo> agentList = agentStorageService.getList();
    long startBlockHeight = NulsContext.getInstance().getBestHeight();
    for (AgentPo agent : agentList) {
        if (agent.getDelHeight() != -1L && agent.getDelHeight() <= startBlockHeight) {
            continue;
        }
        if (agent.getBlockHeight() > startBlockHeight || agent.getBlockHeight() < 0L) {
            continue;
        }
        if (Arrays.equals(address, agent.getAgentAddress())) {
            return agent;
        }
```

```
        }
        return null;
    }

    private Na getAgentTotalDeposit(NulsDigestData hash) {
        List<DepositPo> depositList = depositStorageService.getList();
        long startBlockHeight = NulsContext.getInstance().getBestHeight();
        Na na = Na.ZERO;
        for (DepositPo deposit : depositList) {
            if (deposit.getDelHeight() != -1L && deposit.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (deposit.getBlockHeight() > startBlockHeight || deposit.getBlockHeight() < 0L) {
                continue;
            }
            if (!deposit.getAgentHash().equals(hash)) {
                continue;
            }
            na = na.add(deposit.getDeposit());
        }
        return na;
    }
}

52:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\processor\RedPunishTxProcessor.java
 */

package io.nuls.consensus.poc.tx.processor;

import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.constant.PunishType;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.consensus.poc.storage.service.PunishLogStorageService;
```

```
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.NulsRuntimeException;
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.SerializeUtils;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;

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

/**
 * @author Niels
 */
@Component
public class RedPunishTxProcessor implements TransactionProcessor<RedPunishTransaction> {

    @Autowired
    private PunishLogStorageService storageService;

    @Autowired
    private AgentStorageService agentStorageService;

    @Autowired
    private DepositStorageService depositStorageService;

    @Autowired
    private LedgerService ledgerService;

    @Autowired
    private AccountLedgerService accountLedgerService;

    @Override
    public Result onRollback(RedPunishTransaction tx, Object secondaryData) {
        RedPunishData punishData = tx.getTxData();
```

```
        List<AgentPo> agentList = agentStorageService.getList();
        AgentPo agent = null;
        for (AgentPo agentPo : agentList) {
            if (agentPo.getDelHeight() <= 0) {
                continue;
            }
            if (Arrays.equals(agentPo.getAgentAddress(), punishData.getAddress())) {
                agent = agentPo;
                break;
            }
        }
        if (null == agent) {
            return Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST);
        }
        List<DepositPo> depositPoList = depositStorageService.getList();
        List<DepositPo> updatedList = new ArrayList<>();
        for (DepositPo po : depositPoList) {
            po.setDelHeight(-1);
            boolean success = this.depositStorageService.save(po);
            if (!success) {
                for (DepositPo po2 : depositPoList) {
                    po2.setDelHeight(tx.getBlockHeight());
                    this.depositStorageService.save(po2);
                }
                return Result.getFailed(PocConsensusErrorCode.UPDATE_DEPOSIT_FAILED);
            }
            updatedList.add(po);
        }
        AgentPo agentPo = agent;
        agentPo.setDelHeight(-1L);
        boolean success = agentStorageService.save(agentPo);
        if (!success) {
            for (DepositPo po2 : depositPoList) {
                po2.setDelHeight(tx.getBlockHeight());
                this.depositStorageService.save(po2);
            }
            return Result.getFailed(PocConsensusErrorCode.UPDATE_AGENT_FAILED);
        }
        byte[] key = ArraysTool.concatenate(punishData.getAddress(), new
byte[]{PunishType.RED.getCode()}, SerializeUtils.uint64ToByteArray(tx.getBlockHeight()), new
byte[]{0});
```

```
        success = storageService.delete(key);
        if (!success) {
            for (DepositPo po2 : depositPoList) {
                po2.setDelHeight(tx.getBlockHeight());
                this.depositStorageService.save(po2);
            }
            agentPo.setDelHeight(tx.getBlockHeight());
            agentStorageService.save(agentPo);
            throw new
NulsRuntimeException(TransactionErrorCode.ROLLBACK_TRANSACTION_FAILED);
        }

        return Result.getSuccess();
    }


    @Override
    public Result onCommit(RedPunishTransaction tx, Object secondaryData) {
        RedPunishData punishData = tx.getTxData();
        BlockHeader header = (BlockHeader) secondaryData;
        BlockExtendsData roundData = new BlockExtendsData(header.getExtend());
        PunishLogPo punishLogPo = new PunishLogPo();
        punishLogPo.setAddress(punishData.getAddress());
        punishLogPo.setHeight(tx.getBlockHeight());
        punishLogPo.setRoundIndex(roundData.getRoundIndex());
        punishLogPo.setTime(tx.getTime());
        punishLogPo.setType(PunishType.RED.getCode());
        punishLogPo.setEvidence(punishData.getEvidence());
        punishLogPo.setReasonCode(punishData.getReasonCode());

        List<AgentPo> agentList = agentStorageService.getList();
        AgentPo agent = null;
        for (AgentPo agentPo : agentList) {
            if (agentPo.getDelHeight() > 0) {
                continue;
            }
            if (Arrays.equals(agentPo.getAgentAddress(), punishLogPo.getAddress())) {
                agent = agentPo;
                break;
            }
        }
        if (null == agent) {
```

```
            Log.error("There is no agent can be punished.");
            return Result.getSuccess();
        }


        List<DepositPo> depositPoList = depositStorageService.getList();
        List<DepositPo> updatedList = new ArrayList<>();
        for (DepositPo po : depositPoList) {
            if (po.getDelHeight() >= 0) {
                continue;
            }
            if (!po.getAgentHash().equals(agent.getHash())) {
                continue;
            }
            po.setDelHeight(tx.getBlockHeight());
            boolean b = depositStorageService.save(po);
            if (!b) {
                for (DepositPo po2 : updatedList) {
                    po2.setDelHeight(-1);
                    this.depositStorageService.save(po2);
                }
                return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.UPDATE_DEPOSIT_FAILED);
            }
            updatedList.add(po);
        }
        boolean success = storageService.save(punishLogPo);
        if (!success) {
            for (DepositPo po2 : updatedList) {
                po2.setDelHeight(-1);
                this.depositStorageService.save(po2);
            }
            throw new
NulsRuntimeException(TransactionErrorCode.ROLLBACK_TRANSACTION_FAILED);
        }
        AgentPo agentPo = agent;
        agentPo.setDelHeight(tx.getBlockHeight());
        success = agentStorageService.save(agentPo);
        if (!success) {
            for (DepositPo po2 : updatedList) {
                po2.setDelHeight(-1);
                this.depositStorageService.save(po2);
```

```
            }
            this.storageService.delete(punishLogPo.getKey());
            return Result.getFailed(PocConsensusErrorCode.UPDATE_AGENT_FAILED);
        }
        return Result.getSuccess();
    }


    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        return ValidateResult.getSuccessResult();
    }
}

53:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\processor\StopAgentTxProcessor.java
 */

package io.nuls.consensus.poc.tx.processor;

import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.protocol.tx.RedPunishTransaction;
import io.nuls.consensus.poc.protocol.tx.StopAgentTransaction;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.TransactionErrorCode;
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.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 io.nuls.ledger.service.LedgerService;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * @author Niels
 */
@Component
public class StopAgentTxProcessor implements TransactionProcessor<StopAgentTransaction> {

    @Autowired
    private AgentStorageService agentStorageService;

    @Autowired
    private LedgerService ledgerService;

    @Autowired
    private AccountLedgerService accountLedgerService;

    @Autowired
    private DepositStorageService depositStorageService;

    @Override
    public Result onRollback(StopAgentTransaction tx, Object secondaryData) {
        AgentPo agentPo = agentStorageService.get(tx.getTxData().getCreateTxHash());
        if (null == agentPo || agentPo.getDelHeight() < 0) {
            throw new NulsRuntimeException(PocConsensusErrorCode.AGENT_NOT_EXIST);
        }
        agentPo.setDelHeight(-1L);
        List<DepositPo> depositPoList = depositStorageService.getList();
        for (DepositPo depositPo : depositPoList) {
            if (depositPo.getDelHeight() != tx.getBlockHeight()) {
                continue;
            }
            if (!depositPo.getAgentHash().equals(agentPo.getHash())) {
                continue;
```

```
        }
        depositPo.setDelHeight(-1L);
        depositStorageService.save(depositPo);
    }
    boolean b = agentStorageService.save(agentPo);
    if (!b) {
        return Result.getFailed(PocConsensusErrorCode.UPDATE_AGENT_FAILED);
    }
    return Result.getSuccess();
}

@Override
public Result onCommit(StopAgentTransaction tx, Object secondaryData) {
    BlockHeader header = (BlockHeader) secondaryData;
    if (tx.getTime() < (header.getTime() - 300000L)) {
        return Result.getFailed(PocConsensusErrorCode.LOCK_TIME_NOT_REACHED);
    }
    AgentPo agentPo = agentStorageService.get(tx.getTxData().getCreateTxHash());
    if (null == agentPo || agentPo.getDelHeight() > 0) {
        throw new NulsRuntimeException(PocConsensusErrorCode.AGENT_NOT_EXIST);
    }
    List<DepositPo> depositPoList = depositStorageService.getList();
    for (DepositPo depositPo : depositPoList) {
        if (depositPo.getDelHeight() > -1L) {
            continue;
        }
        if (!depositPo.getAgentHash().equals(agentPo.getHash())) {
            continue;
        }
        depositPo.setDelHeight(tx.getBlockHeight());
        depositStorageService.save(depositPo);
    }
    agentPo.setDelHeight(tx.getBlockHeight());
    tx.getTxData().setAddress(agentPo.getAgentAddress());

    boolean b = agentStorageService.save(agentPo);
    if (!b) {
        return Result.getFailed(PocConsensusErrorCode.UPDATE_AGENT_FAILED);
    }
    return Result.getSuccess();
}
```

```
    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        if (txList == null || txList.isEmpty()) {
            return ValidateResult.getSuccessResult();
        }
        Set<NulsDigestData> hashSet = new HashSet<>();
        Set<String> addressSet = new HashSet<>();
        for (Transaction tx : txList) {
            if (tx.getType() == ConsensusConstant.TX_TYPE_RED_PUNISH) {
                RedPunishTransaction transaction = (RedPunishTransaction) tx;
addressSet.add(AddressTool.getStringAddressByBytes(transaction.getTxData().getAddress()));
       }
        }
        for (Transaction tx : txList) {
            if (tx.getType() == ConsensusConstant.TX_TYPE_STOP_AGENT) {
                StopAgentTransaction transaction = (StopAgentTransaction) tx;
                if (!hashSet.add(transaction.getTxData().getCreateTxHash())) {
                    ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TRANSACTION_REPEATED);
                    result.setData(transaction);
                    return result;
                }
                if (transaction.getTxData().getAddress() == null) {
                    CreateAgentTransaction agentTransaction = (CreateAgentTransaction)
ledgerService.getTx(transaction.getTxData().getCreateTxHash());
                    if (null == agentTransaction) {
                        ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_NOT_EXIST);
                        result.setData(transaction);
                        return result;
                    }
transaction.getTxData().setAddress(agentTransaction.getTxData().getAgentAddress());                }
                if
(addressSet.contains(AddressTool.getStringAddressByBytes(transaction.getTxData().getAddress(
)))) {
                    ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_STOPPED);
                    result.setData(transaction);
                    return result;
                }
            }
        }
```

```
        return ValidateResult.getSuccessResult();
    }
}
```

54:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\tx\processor\YellowPunishTxProcessor.java
*/

```
package io.nuls.consensus.poc.tx.processor;

import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.protocol.constant.PunishType;
import io.nuls.consensus.poc.protocol.entity.YellowPunishData;
import io.nuls.consensus.poc.protocol.tx.YellowPunishTransaction;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.storage.service.PunishLogStorageService;
import io.nuls.core.tools.array.ArraysTool;
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.BlockHeader;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.processor.TransactionProcessor;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.kernel.utils.VarInt;
import io.nuls.kernel.validate.ValidateResult;

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

/**
 * @author Niels
 */
@Component
public class YellowPunishTxProcessor implements
TransactionProcessor<YellowPunishTransaction> {

    @Autowired
    private PunishLogStorageService punishLogStorageService;
```

```
    @Override
    public Result onRollback(YellowPunishTransaction tx, Object secondaryData) {
        YellowPunishData punishData = tx.getTxData();
        List<byte[]> deletedList = new ArrayList<>();
        int deleteIndex = 1;
        for (byte[] address : punishData.getAddressList()) {
            boolean result = punishLogStorageService.delete(this.getPoKey(address,
PunishType.YELLOW.getCode(), tx.getBlockHeight(), deleteIndex++));
            if (!result) {
                BlockHeader header = (BlockHeader) secondaryData;
                BlockExtendsData roundData = new BlockExtendsData(header.getExtend());
                int index = 1;
                for (byte[] bytes : deletedList) {
                    PunishLogPo po = new PunishLogPo();
                    po.setAddress(bytes);
                    po.setHeight(tx.getBlockHeight());
                    po.setRoundIndex(roundData.getRoundIndex());
                    po.setTime(tx.getTime());
                    po.setIndex(index++);
                    po.setType(PunishType.YELLOW.getCode());
                    punishLogStorageService.save(po);
                }
                throw new
NulsRuntimeException(TransactionErrorCode.ROLLBACK_TRANSACTION_FAILED);
            } else {
                deletedList.add(address);
            }
        }
        return Result.getSuccess();
    }

    @Override
    public Result onCommit(YellowPunishTransaction tx, Object secondaryData) {
        YellowPunishData punishData = tx.getTxData();
        BlockHeader header = (BlockHeader) secondaryData;
        BlockExtendsData roundData = new BlockExtendsData(header.getExtend());
        List<PunishLogPo> savedList = new ArrayList<>();
        int index = 1;
        for (byte[] address : punishData.getAddressList()) {
            PunishLogPo po = new PunishLogPo();
            po.setAddress(address);
            po.setHeight(tx.getBlockHeight());
```

```
            po.setRoundIndex(roundData.getRoundIndex());
            po.setTime(tx.getTime());
            po.setIndex(index++);
            po.setType(PunishType.YELLOW.getCode());
            boolean result = punishLogStorageService.save(po);
            if (!result) {
                for (PunishLogPo punishLogPo : savedList) {
                    punishLogStorageService.delete(getPoKey(punishLogPo.getAddress(),
PunishType.YELLOW.getCode(), punishLogPo.getHeight(), punishLogPo.getIndex()));
                }
                throw new
NulsRuntimeException(TransactionErrorCode.ROLLBACK_TRANSACTION_FAILED);
            } else {
                savedList.add(po);
            }
        }
        return Result.getSuccess();
    }

    @Override
    public ValidateResult conflictDetect(List<Transaction> txList) {
        return ValidateResult.getSuccessResult();
    }

    /**
     * key
     */
    private byte[] getPoKey(byte[] address, byte type, long blockHeight, int index) {
        return ArraysTool.concatenate(address, new byte[]{type},
SerializeUtils.uint64ToByteArray(blockHeight), new VarInt(index).encode());
    }
}

55:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\validator\AgentAddressesValidator.java
 */

package io.nuls.consensus.poc.tx.validator;

import io.nuls.consensus.poc.config.ConsensusConfig;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
```

```
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

import java.util.Arrays;
import java.util.List;

/**
 * @author: Niels Wang
 */
@Component
public class AgentAddressesValidator extends
BaseConsensusProtocolValidator<CreateAgentTransaction> {
    /**
     * @param data
     * @return
     */
    @Override
    public ValidateResult validate(CreateAgentTransaction data) {
        Agent agent = data.getTxData();
        for (byte[] address : ConsensusConfig.getSeedNodeList()) {
            if (Arrays.equals(address, agent.getAgentAddress())) {
                return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_EXIST);
            } else if (Arrays.equals(address, agent.getPackingAddress())) {
                return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_PACKING_EXIST);
            }
        }
        long count = 0;
        try {
            count = this.getRedPunishCount(agent.getAgentAddress());
        } catch (Exception e) {
            Log.error(e);
            return ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.SYS_UNKOWN_EXCEPTION);
        }
```

```
        if (count > 0) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.LACK_OF_CREDIT);
        }
        return ValidateResult.getSuccessResult();
    }
}
```

56:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\tx\validator\AgentCountValidator.java

```
 */

package io.nuls.consensus.poc.tx.validator;

import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.cfg.NulsConfig;
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.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

import java.io.UnsupportedEncodingException;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * date 2018/3/23.
 *
 * @author Facjas
 */
@Component
public class AgentCountValidator implements NulsDataValidator<CreateAgentTransaction> {
```

```
    @Autowired
    private AgentStorageService agentStorageService;

    @Override
    public ValidateResult validate(CreateAgentTransaction tx) {
        ValidateResult result = ValidateResult.getSuccessResult();
        Agent agent = tx.getTxData();

        List<AgentPo> caList = agentStorageService.getList();
        if (caList != null) {
            Set<String> set = new HashSet<>();
            for (AgentPo ca : caList) {
                if (ca.getHash().equals(tx.getHash())) {
                    return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TRANSACTION_REPEATED);
                }
                if (ca.getDelHeight() > 0L) {
                    continue;
                }

                set.add(Hex.encode(ca.getAgentAddress()));
                set.add(Hex.encode(ca.getPackingAddress()));
            }
            boolean b = set.add(Hex.encode(agent.getAgentAddress()));
            if (!b) {
                return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_EXIST);
            }
            b = set.add(Hex.encode(agent.getPackingAddress()));
            if (!b) {
                return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_PACKING_EXIST);
            }
        }
        return result;
    }
}

57:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\validator\BaseConsensusProtocolValidator.java
 */
```

```
package io.nuls.consensus.poc.tx.validator;

import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.CoinData;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsData;
import io.nuls.kernel.validate.NulsDataValidator;

import java.util.Arrays;
import java.util.List;

/**
 * @author Niels
 */
public abstract class BaseConsensusProtocolValidator<T extends NulsData> implements
NulsDataValidator<T> {

    protected final boolean isDepositOk(Na deposit, CoinData coinData) {
        if(coinData == null || coinData.getTo().size() == 0) {
            return false;
        }
        Coin coin = coinData.getTo().get(0);
        if(!deposit.equals(coin.getNa())) {
            return false;
        }
        if(coin.getLockTime() != PocConsensusProtocolConstant.LOCK_OF_LOCK_TIME) {
            return false;
        }
        return true;
    }

    protected long getRedPunishCount(byte[] address ) {
        List<PunishLogPo> list =
PocConsensusContext.getChainManager().getMasterChain().getChain().getRedPunishList();
        if (null == list || list.isEmpty()) {
            return 0;
        }
        long count = 0;
        for (PunishLogPo po : list) {
```

```
            if (Arrays.equals(address, po.getAddress())) {
                count++;
            }
        }
        return count;
    }
}
```

58:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\tx\validator\CancelDepositTxValidator.java
*/

```
package io.nuls.consensus.poc.tx.validator;

import io.nuls.consensus.poc.protocol.tx.CancelDepositTransaction;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.SeverityLevelEnum;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;

import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.script.TransactionSignature;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

import java.util.Arrays;

/**
 * @author Niels
 */
@Component
public class CancelDepositTxValidator implements
NulsDataValidator<CancelDepositTransaction> {

    @Autowired
    private DepositStorageService depositStorageService;
```

```
    @Override
    public ValidateResult validate(CancelDepositTransaction data) throws NulsException {
        DepositPo depositPo = depositStorageService.get(data.getTxData().getJoinTxHash());
        if(null==depositPo||depositPo.getDelHeight()>0){
            return
ValidateResult.getFailedResult(this.getClass().getName(),KernelErrorCode.DATA_NOT_FOUND);
        }
        TransactionSignature sig = new TransactionSignature();
        try {
            sig.parse(data.getTransactionSignature(), 0);
        } catch (NulsException e) {
            Log.error(e);
            return ValidateResult.getFailedResult(this.getClass().getName(), e.getErrorCode());
        }
        if (!SignatureUtil.containsAddress(data,depositPo.getAddress())) {
            ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.SIGNATURE_ERROR);
            result.setLevel(SeverityLevelEnum.FLAGRANT_FOUL);
            return result;
        }
        return ValidateResult.getSuccessResult();
    }
}

59:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\validator\CreateAgentTxValidator.java
 */

package io.nuls.consensus.poc.tx.validator;

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.SeverityLevelEnum;
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.CoinData;

import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.script.TransactionSignature;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

/**
 * @author ln
 */
@Component
public class CreateAgentTxValidator extends
BaseConsensusProtocolValidator<CreateAgentTransaction> {

    //
    private static final int INSTRACTION_MAX_LENGTH = 200;

    private static final int AGENT_NAME_MAX_LENGTH = 32;

    @Override
    public ValidateResult validate(CreateAgentTransaction tx) {

        Agent agent = tx.getTxData();
        if (null == agent) {
            return ValidateResult.getFailedResult(getClass().getName(),
PocConsensusErrorCode.AGENT_NOT_EXIST);
        }
        if (!AddressTool.validNormalAddress(agent.getPackingAddress())) {
            return ValidateResult.getFailedResult(getClass().getName(),
AccountErrorCode.ADDRESS_ERROR);
        }
        if (Arrays.equals(agent.getAgentAddress(), agent.getPackingAddress())) {
            return ValidateResult.getFailedResult(getClass().getName(),
PocConsensusErrorCode.AGENTADDR_AND_PACKING_SAME);
        }
        if (Arrays.equals(agent.getRewardAddress(), agent.getPackingAddress())) {
            return ValidateResult.getFailedResult(getClass().getName(),
```

```
PocConsensusErrorCode.REWARDADDR_PACKING_SAME);
        }

        if (tx.getTime() <= 0) {
            return ValidateResult.getFailedResult(getClass().getName(),
KernelErrorCode.DATA_ERROR);
        }

        double commissionRate = agent.getCommissionRate();
        if (commissionRate < PocConsensusProtocolConstant.MIN_COMMISSION_RATE ||
commissionRate > PocConsensusProtocolConstant.MAX_COMMISSION_RATE) {
            return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
PocConsensusErrorCode.COMMISSION_RATE_OUT_OF_RANGE);
        }

        if
(PocConsensusProtocolConstant.AGENT_DEPOSIT_LOWER_LIMIT.isGreaterThan(agent.getDe
posit())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.DEPOSIT_NOT_ENOUGH);
        }
        if
(PocConsensusProtocolConstant.AGENT_DEPOSIT_UPPER_LIMIT.isLessThan(agent.getDeposi
t())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.DEPOSIT_TOO_MUCH);
        }

        if (!isDepositOk(agent.getDeposit(), tx.getCoinData())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
SeverityLevelEnum.FLAGRANT_FOUL, PocConsensusErrorCode.DEPOSIT_ERROR);
        }
        TransactionSignature sig = new TransactionSignature();
        try {
            sig.parse(tx.getTransactionSignature(), 0);
        } catch (NulsException e) {
            Log.error(e);
            return ValidateResult.getFailedResult(this.getClass().getName(), e.getErrorCode());
        }
        try {
            if (!SignatureUtil.containsAddress(tx, agent.getAgentAddress())) {
                ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
```

```
KernelErrorCode.SIGNATURE_ERROR);
                result.setLevel(SeverityLevelEnum.FLAGRANT_FOUL);
                return result;
            }
        } catch (NulsException e) {
            ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.SIGNATURE_ERROR);
            result.setLevel(SeverityLevelEnum.FLAGRANT_FOUL);
            return result;
        }
        CoinData coinData = tx.getCoinData();
        Set<String> addressSet = new HashSet<>();
        int lockCount = 0;
        for (Coin coin : coinData.getTo()) {
            if (coin.getLockTime() == PocConsensusConstant.CONSENSUS_LOCK_TIME) {
                lockCount++;
            }
            //addressSet.add(AddressTool.getStringAddressByBytes(coin.()));
            addressSet.add(AddressTool.getStringAddressByBytes(coin.getAddress()));
        }
        if (lockCount > 1) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        if (addressSet.size() > 1) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        return ValidateResult.getSuccessResult();
    }
}

60:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\tx\validator\DepositTxValidator.java
 */

package io.nuls.consensus.poc.tx.validator;/*
 * MIT License
 *
 * Copyright (c) 2017-2018 nuls.io
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
```

```
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.protocol.tx.DepositTransaction;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.consensus.poc.storage.service.PunishLogStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.SeverityLevelEnum;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Coin;
```

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

import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.script.TransactionSignature;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;

import java.util.*;

/**
 * @author ln
 */
@Component
public class DepositTxValidator extends BaseConsensusProtocolValidator<DepositTransaction> {

    @Autowired
    private PunishLogStorageService punishLogStorageService;

    @Autowired
    private AgentStorageService agentStorageService;

    @Autowired
    private DepositStorageService depositStorageService;

    @Override
    public ValidateResult validate(DepositTransaction tx) throws NulsException {
        if (null == tx || null == tx.getTxData() || null == tx.getTxData().getAgentHash() || null ==
tx.getTxData().getDeposit() || null == tx.getTxData().getAddress()) {
            return
ValidateResult.getFailedResult(this.getClass().getName(),TransactionErrorCode.TX_DATA_VALI
DATION_ERROR);
        }
        Deposit deposit = tx.getTxData();
        AgentPo agentPo = agentStorageService.get(deposit.getAgentHash());
        if (null == agentPo || agentPo.getDelHeight() > 0) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_NOT_EXIST);
        }
        List<DepositPo> poList = this.getDepositListByAgent(deposit.getAgentHash());
        if (null != poList && poList.size() >=
```

```
PocConsensusProtocolConstant.MAX_ACCEPT_NUM_OF_DEPOSIT) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.DEPOSIT_OVER_COUNT);
        }
        Na limit = PocConsensusProtocolConstant.ENTRUSTER_DEPOSIT_LOWER_LIMIT;
        Na max =
PocConsensusProtocolConstant.SUM_OF_DEPOSIT_OF_AGENT_UPPER_LIMIT;
        Na total = Na.ZERO;
        for (DepositPo cd : poList) {
            total = total.add(cd.getDeposit());
        }
        if (limit.isGreaterThan(deposit.getDeposit())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.DEPOSIT_NOT_ENOUGH);
        }
        if (max.isLessThan(total.add(deposit.getDeposit()))) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.DEPOSIT_TOO_MUCH);
        }

        if (!isDepositOk(deposit.getDeposit(), tx.getCoinData())) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
SeverityLevelEnum.FLAGRANT_FOUL, PocConsensusErrorCode.DEPOSIT_ERROR);
        }
        TransactionSignature sig = new TransactionSignature();
        try {
            sig.parse(tx.getTransactionSignature(), 0);
        } catch (NulsException e) {
            Log.error(e);
            return ValidateResult.getFailedResult(this.getClass().getName(), e.getErrorCode());
        }
        if (!SignatureUtil.containsAddress(tx,deposit.getAddress())) {
            ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.SIGNATURE_ERROR);
            result.setLevel(SeverityLevelEnum.FLAGRANT_FOUL);
            return result;
        }
        CoinData coinData = tx.getCoinData();
        Set<String> addressSet = new HashSet<>();
        int lockCount = 0;
        for (Coin coin : coinData.getTo()) {
            if (coin.getLockTime() == PocConsensusConstant.CONSENSUS_LOCK_TIME) {
```

```
                lockCount++;
            }
            //addressSet.add(AddressTool.getStringAddressByBytes(coin.()));
            addressSet.add(AddressTool.getStringAddressByBytes(coin.getAddress()));
        }
        if (lockCount > 1) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        if (addressSet.size() > 1) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        return ValidateResult.getSuccessResult();
    }

    private List<DepositPo> getDepositListByAgent(NulsDigestData agentHash) {
        List<DepositPo> depositList = depositStorageService.getList();
        long startBlockHeight = NulsContext.getInstance().getBestHeight();
        List<DepositPo> resultList = new ArrayList<>();
        for (DepositPo deposit : depositList) {
            if (deposit.getDelHeight() != -1L && deposit.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (deposit.getBlockHeight() > startBlockHeight || deposit.getBlockHeight() < 0L) {
                continue;
            }
            if (!deposit.getAgentHash().equals(agentHash)) {
                continue;
            }
            if (deposit.getAgentHash().equals(agentHash)) {
                resultList.add(deposit);
            }
        }
        return resultList;
    }


}
```

61:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\tx\validator\StopAgentTxValidator.java

```
 */

package io.nuls.consensus.poc.tx.validator;

import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.tx.StopAgentTransaction;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.SeverityLevelEnum;
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.Coin;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsDigestData;

import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.script.TransactionSignature;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

import java.util.*;

/**
 * @author Niels
 */
@Component
public class StopAgentTxValidator implements NulsDataValidator<StopAgentTransaction> {

    @Autowired
    private AgentStorageService agentStorageService;

    @Autowired
    private DepositStorageService depositStorageService;
```

```
    @Override
    public ValidateResult validate(StopAgentTransaction data) throws NulsException {
        AgentPo agentPo = agentStorageService.get(data.getTxData().getCreateTxHash());
        if (null == agentPo || agentPo.getDelHeight() > 0) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.AGENT_NOT_EXIST);
        }
        TransactionSignature sig = new TransactionSignature();
        try {
            sig.parse(data.getTransactionSignature(), 0);
        } catch (NulsException e) {
            Log.error(e);
            return ValidateResult.getFailedResult(this.getClass().getName(), e.getErrorCode());
        }

        if (!SignatureUtil.containsAddress(data,agentPo.getAgentAddress())) {
            ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.SIGNATURE_ERROR);
            result.setLevel(SeverityLevelEnum.FLAGRANT_FOUL);
            return result;
        }
        if (data.getCoinData().getTo() == null || data.getCoinData().getTo().isEmpty()) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        List<DepositPo> allDepositList = depositStorageService.getList();
        Map<NulsDigestData, DepositPo> depositMap = new HashMap<>();
        Na totalNa = agentPo.getDeposit();
        DepositPo ownDeposit = new DepositPo();
        ownDeposit.setDeposit(agentPo.getDeposit());
        ownDeposit.setAddress(agentPo.getAgentAddress());
        depositMap.put(data.getTxData().getCreateTxHash(), ownDeposit);
        for (DepositPo deposit : allDepositList) {
            if (deposit.getDelHeight() > -1L && (data.getBlockHeight() == -1L || deposit.getDelHeight()
< data.getBlockHeight())) {
                continue;
            }
            if (!deposit.getAgentHash().equals(agentPo.getHash())) {
                continue;
            }
            depositMap.put(deposit.getTxHash(), deposit);
            totalNa = totalNa.add(deposit.getDeposit());
```

```
        }

        Na fromTotal = Na.ZERO;
        Map<String, Na> verifyToMap = new HashMap<>();
        for (Coin coin : data.getCoinData().getFrom()) {
            if (coin.getLockTime() != -1L){
                return
ValidateResult.getFailedResult(this.getClass().getName(),TransactionErrorCode.TX_DATA_VALI
DATION_ERROR);
            }
                NulsDigestData txHash = new NulsDigestData();
            txHash.parse(coin.getOwner(), 0);
            DepositPo deposit = depositMap.remove(txHash);
            if (deposit == null) {
                return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
            }
            if (deposit.getAgentHash() == null && !coin.getNa().equals(agentPo.getDeposit())) {
                return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
            } else if (!deposit.getDeposit().equals(coin.getNa())) {
                return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
            }
            fromTotal = fromTotal.add(coin.getNa());
            if (deposit.getAgentHash() == null) {
                continue;
            }
            String address = AddressTool.getStringAddressByBytes(deposit.getAddress());
            Na na = verifyToMap.get(address);
            if (null == na) {
                na = deposit.getDeposit();
            } else {
                na = na.add(deposit.getDeposit());
            }
            verifyToMap.put(address, na);
        }
        if (!depositMap.isEmpty()) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        if (!totalNa.equals(fromTotal)) {
```

```
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        Na ownToCoin = ownDeposit.getDeposit().subtract(data.getFee());
        long ownLockTime = 0L;
        for (Coin coin : data.getCoinData().getTo()) {
            //String address = AddressTool.getStringAddressByBytes(coin.());
            String address = AddressTool.getStringAddressByBytes(coin.getAddress());
            Na na = verifyToMap.get(address);
            if (null != na && na.equals(coin.getNa())) {
                verifyToMap.remove(address);
                continue;
            }
            if (ownToCoin != null && Arrays.equals(coin.getAddress(), ownDeposit.getAddress()) &&
coin.getNa().equals(ownToCoin)) {
                ownToCoin = null;
                ownLockTime = coin.getLockTime();
                continue;
            } else {
                return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
            }
        }
        if (ownLockTime < (data.getTime() + PocConsensusConstant.STOP_AGENT_LOCK_TIME))
{
            return ValidateResult.getFailedResult(this.getClass().getName(),
PocConsensusErrorCode.LOCK_TIME_NOT_REACHED);
        }
        if (!verifyToMap.isEmpty()) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }

        return ValidateResult.getSuccessResult();
    }
}
```

62:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\main\java\io\nuls\consensus\poc\util\ConsensusTool.java

```
 *
 */
package io.nuls.consensus.poc.util;
```

```
import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.model.Account;
import io.nuls.account.service.AccountService;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.model.BlockData;
import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.consensus.poc.model.MeetingMember;
import io.nuls.consensus.poc.model.MeetingRound;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.protocol.entity.YellowPunishData;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.protocol.tx.DepositTransaction;
import io.nuls.consensus.poc.protocol.tx.YellowPunishTransaction;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.contract.constant.ContractConstant;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.entity.tx.ContractTransaction;
import io.nuls.contract.entity.txdata.ContractData;
import io.nuls.contract.service.ContractService;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.calc.DoubleUtils;
import io.nuls.core.tools.calc.LongUtils;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.*;

import io.nuls.kernel.script.BlockSignature;
import io.nuls.kernel.script.Script;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.ByteArrayWrapper;
import io.nuls.kernel.utils.VarInt;
import io.nuls.ledger.service.LedgerService;
```

```
import io.nuls.protocol.model.SmallBlock;
import io.nuls.protocol.model.tx.CoinBaseTransaction;

import java.io.IOException;
import java.util.*;

/**
 * @author Niels
 */
public class ConsensusTool {

    private static AccountService accountService =
NulsContext.getServiceBean(AccountService.class);
    private static AgentStorageService agentStorageService =
NulsContext.getServiceBean(AgentStorageService.class);
    private static DepositStorageService depositStorageService =
NulsContext.getServiceBean(DepositStorageService.class);
    private static LedgerService ledgerService =
NulsContext.getServiceBean(LedgerService.class);
    /**
     * pierre add
     */
    private static ContractService contractService =
NulsContext.getServiceBean(ContractService.class);

    public static SmallBlock getSmallBlock(Block block) {
        SmallBlock smallBlock = new SmallBlock();
        smallBlock.setHeader(block.getHeader());
        List<NulsDigestData> txHashList = new ArrayList<>();
        for (Transaction tx : block.getTxs()) {
            txHashList.add(tx.getHash());
            if (tx.isSystemTx()) {
                smallBlock.addBaseTx(tx);
            }
        }
        smallBlock.setTxHashList(txHashList);
        return smallBlock;
    }

    public static Block createBlock(BlockData blockData, Account account) throws NulsException {
        if (null == account) {
            throw new NulsRuntimeException(AccountErrorCode.ACCOUNT_NOT_EXIST);
```

```
        }

        // Account cannot be encrypted, otherwise it will be wrong
        //
        if (account.isEncrypted()) {
            throw new
NulsRuntimeException(AccountErrorCode.ACCOUNT_IS_ALREADY_ENCRYPTED);
        }

        Block block = new Block();
        block.setTxs(blockData.getTxList());
        BlockHeader header = new BlockHeader();
        block.setHeader(header);
        try {
//
block.getHeader().setExtend(ArraysTool.concatenate(blockData.getExtendsData().serialize(),new
byte[]{0,1,0,1,1}));
            block.getHeader().setExtend(blockData.getExtendsData().serialize());
        } catch (IOException e) {
            Log.error(e);
            throw new NulsRuntimeException(e);
        }
        header.setHeight(blockData.getHeight());
        header.setTime(blockData.getTime());
        header.setPreHash(blockData.getPreHash());
        header.setTxCount(blockData.getTxList().size());
        List<NulsDigestData> txHashList = new ArrayList<>();
        for (int i = 0; i < blockData.getTxList().size(); i++) {
            Transaction tx = blockData.getTxList().get(i);
            tx.setBlockHeight(header.getHeight());
            txHashList.add(tx.getHash());
        }
        header.setMerkleHash(NulsDigestData.calcMerkleDigestData(txHashList));
        header.setHash(NulsDigestData.calcDigestData(block.getHeader()));

        BlockSignature scriptSig = new BlockSignature();

        NulsSignData signData = accountService.signDigest(header.getHash().getDigestBytes(),
account.getEcKey());
        scriptSig.setSignData(signData);
        scriptSig.setPublicKey(account.getPubKey());
        header.setBlockSignature(scriptSig);
```

```
        //header.setStateRoot(blockData.getStateRoot());

        return block;
    }

    public static CoinBaseTransaction createCoinBaseTx(MeetingMember member,
List<Transaction> txList, MeetingRound localRound, long unlockHeight) {
        CoinData coinData = new CoinData();
        // Gas
        List<Coin> returnGasList = returnContractSenderNa(txList, unlockHeight);

        List<Coin> rewardList = calcReward(txList, member, localRound, unlockHeight);
        if (!returnGasList.isEmpty()) {
            Coin agentReward = rewardList.remove(0);
            rewardList.addAll(returnGasList);
            rewardList.sort(new Comparator<Coin>() {
                @Override
                public int compare(Coin o1, Coin o2) {
                    return Arrays.hashCode(o1.getOwner()) > Arrays.hashCode(o2.getOwner()) ? 1 : -1;
                }
            });
            rewardList.add(0, agentReward);
        }

        for (Coin coin : rewardList) {
            coinData.addTo(coin);
        }
        CoinBaseTransaction tx = new CoinBaseTransaction();
        tx.setTime(member.getPackEndTime());
        tx.setCoinData(coinData);
        try {
            tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        } catch (IOException e) {
            Log.error(e);
        }
        return tx;
    }

    private static List<Coin> returnContractSenderNa(List<Transaction> txList, long unlockHeight) {
        // , senderGasNa
        Map<ByteArrayWrapper, Na> returnMap = new HashMap<>();
        List<Coin> returnList = new ArrayList<>();
```

```
if (txList != null && txList.size() > 0) {
    int txType;
    for (Transaction tx : txList) {
        txType = tx.getType();
        if (txType == ContractConstant.TX_TYPE_CREATE_CONTRACT
                || txType == ContractConstant.TX_TYPE_CALL_CONTRACT
                || txType == ContractConstant.TX_TYPE_DELETE_CONTRACT) {
            ContractTransaction contractTx = (ContractTransaction) tx;
            ContractResult contractResult = contractTx.getContractResult();
            if (contractResult == null) {
                contractResult = contractService.getContractExecuteResult(tx.getHash());
                if (contractResult == null) {
                    Log.error("get contract tx contractResult error: " + tx.getHash().getDigestHex());
                    continue;
                }
            }
            contractTx.setContractResult(contractResult);

            // Gas
            if (txType == ContractConstant.TX_TYPE_DELETE_CONTRACT) {
                continue;
            }
            // Gas
            ContractData contractData = (ContractData) tx.getTxData();
            long realGasUsed = contractResult.getGasUsed();
            long txGasUsed = contractData.getGasLimit();
            long returnGas = 0;
            Na returnNa = Na.ZERO;
            if (txGasUsed > realGasUsed) {
                returnGas = txGasUsed - realGasUsed;
                returnNa = Na.valueOf(LongUtils.mul(returnGas, contractData.getPrice()));
                //  -> senderGas, Call,Create,DeleteContractTransaction getFee
                contractTx.setReturnNa(returnNa);

                ByteArrayWrapper sender = new ByteArrayWrapper(contractData.getSender());
                Na senderNa = returnMap.get(sender);
                if (senderNa == null) {
                    senderNa = Na.ZERO.add(returnNa);
                } else {
                    senderNa = senderNa.add(returnNa);
                }
                returnMap.put(sender, senderNa);
```

```
                }
            }
        }
        Set<Map.Entry<ByteArrayWrapper, Na>> entries = returnMap.entrySet();
        Coin returnCoin;
        for (Map.Entry<ByteArrayWrapper, Na> entry : entries) {
            returnCoin = new Coin(entry.getKey().getBytes(), entry.getValue(), unlockHeight);
            returnList.add(returnCoin);
        }
    }
    return returnList;
}

private static List<Coin> calcReward(List<Transaction> txList, MeetingMember self,
MeetingRound localRound, long unlockHeight) {
    List<Coin> rewardList = new ArrayList<>();
    if (self.getOwnDeposit().getValue() == Na.ZERO.getValue()) {
        long totalFee = 0;
        for (Transaction tx : txList) {
            totalFee += tx.getFee().getValue();
        }
        if (totalFee == 0L) {
            return rewardList;
        }
        double caReward = totalFee;
        Coin agentReword = new Coin(self.getRewardAddress(), Na.valueOf((long) caReward),
unlockHeight);
        rewardList.add(agentReword);
        return rewardList;
    }
    long totalFee = 0;
    for (Transaction tx : txList) {
        totalFee += tx.getFee().getValue();
    }
    double totalAll = DoubleUtils.mul(localRound.getMemberCount(),
PocConsensusConstant.BLOCK_REWARD.getValue());
    double commissionRate = DoubleUtils.div(self.getCommissionRate(), 100, 2);
    double agentWeight = DoubleUtils.mul(DoubleUtils.sum(self.getOwnDeposit().getValue(),
self.getTotalDeposit().getValue()), self.getCalcCreditVal());
    double blockReword = totalFee;
    if (localRound.getTotalWeight() > 0d && agentWeight > 0d) {
        blockReword = DoubleUtils.sum(blockReword, DoubleUtils.mul(totalAll,
```

```
DoubleUtils.div(agentWeight, localRound.getTotalWeight())));
        }

        if (blockReword == 0d) {
            return rewardList;
        }

        long realTotalAllDeposit = self.getOwnDeposit().getValue() +
self.getTotalDeposit().getValue();
        double caReward = DoubleUtils.mul(blockReword,
DoubleUtils.div(self.getOwnDeposit().getValue(), realTotalAllDeposit));

        for (Deposit deposit : self.getDepositList()) {
            double weight = DoubleUtils.div(deposit.getDeposit().getValue(), realTotalAllDeposit);
            if (Arrays.equals(deposit.getAddress(), self.getAgentAddress())) {
                caReward = caReward + DoubleUtils.mul(blockReword, weight);
            } else {
                double reward = DoubleUtils.mul(blockReword, weight);
                double fee = DoubleUtils.mul(reward, commissionRate);
                caReward = caReward + fee;
                double hisReward = DoubleUtils.sub(reward, fee);
                if (hisReward == 0D) {
                    continue;
                }
                Na depositReward = Na.valueOf(DoubleUtils.longValue(hisReward));

                Coin rewardCoin = null;

                for (Coin coin : rewardList) {
                    if (Arrays.equals(coin.getAddress(), deposit.getAddress())) {
                        rewardCoin = coin;
                        break;
                    }
                }
                if (rewardCoin == null) {
                    rewardCoin = new Coin(deposit.getAddress(), depositReward, unlockHeight);
                    rewardList.add(rewardCoin);
                } else {
                    rewardCoin.setNa(rewardCoin.getNa().add(depositReward));
                }
            }
        }
```

```
        rewardList.sort(new Comparator<Coin>() {
            @Override
            public int compare(Coin o1, Coin o2) {
                return Arrays.hashCode(o1.getOwner()) > Arrays.hashCode(o2.getOwner()) ? 1 : -1;
            }
        });

        Coin agentReward = new Coin(self.getRewardAddress(),
Na.valueOf(DoubleUtils.longValue(caReward)), unlockHeight);
        rewardList.add(0, agentReward);

        return rewardList;
    }


    public static YellowPunishTransaction createYellowPunishTx(Block preBlock, MeetingMember
self, MeetingRound round) throws NulsException, IOException {
        BlockExtendsData preBlockRoundData = new
BlockExtendsData(preBlock.getHeader().getExtend());
        if (self.getRoundIndex() - preBlockRoundData.getRoundIndex() > 1) {
            return null;
        }

        int yellowCount = 0;
        if (self.getRoundIndex() == preBlockRoundData.getRoundIndex() &&
self.getPackingIndexOfRound() != preBlockRoundData.getPackingIndexOfRound() + 1) {
            yellowCount = self.getPackingIndexOfRound() -
preBlockRoundData.getPackingIndexOfRound() - 1;
        }

        if (self.getRoundIndex() != preBlockRoundData.getRoundIndex() &&
(self.getPackingIndexOfRound() != 1 || preBlockRoundData.getPackingIndexOfRound() !=
preBlockRoundData.getConsensusMemberCount())) {
            yellowCount = self.getPackingIndexOfRound() +
preBlockRoundData.getConsensusMemberCount() -
preBlockRoundData.getPackingIndexOfRound() - 1;
        }

        if (yellowCount == 0) {
            return null;
        }
```

```
        List<byte[]> addressList = new ArrayList<>();
        for (int i = 1; i <= yellowCount; i++) {
            int index = self.getPackingIndexOfRound() - i;
            if (index > 0) {
                MeetingMember member = round.getMember(index);
                if (member.getAgent() == null) {
                    continue;
                } else if (member.getAgent().getDelHeight() > 0) {
                    continue;
                }
                addressList.add(member.getAgentAddress());
            } else {
                MeetingRound preRound = round.getPreRound();
                MeetingMember member = preRound.getMember(index +
preRound.getMemberCount());
                if (member.getAgent() == null || member.getAgent().getDelHeight() > 0) {
                    continue;
                }
                addressList.add(member.getAgentAddress());
            }
        }
        if (addressList.isEmpty()) {
            return null;
        }
        YellowPunishTransaction punishTx = new YellowPunishTransaction();
        YellowPunishData data = new YellowPunishData();
        data.setAddressList(addressList);
        punishTx.setTxData(data);
        punishTx.setTime(self.getPackEndTime());
        punishTx.setHash(NulsDigestData.calcDigestData(punishTx.serializeForHash()));
        return punishTx;
    }

    /**
     * coinData
     *
     * @param lockTime (+)Charlie
     */
    public static CoinData getStopAgentCoinData(Agent agent, long lockTime) throws IOException
{
        return getStopAgentCoinData(agent, lockTime, null);
```

```
    }

    public static CoinData getStopAgentCoinData(Agent agent, long lockTime, Long hight) throws
IOException {
        if (null == agent) {
            return null;
        }
        NulsDigestData createTxHash = agent.getTxHash();
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        if (agent.getAgentAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
            Script scriptPubkey = SignatureUtil.createOutputScript(agent.getAgentAddress());
            toList.add(new Coin(scriptPubkey.getProgram(), agent.getDeposit(), lockTime));
        } else {
            toList.add(new Coin(agent.getAgentAddress(), agent.getDeposit(), lockTime));
        }
        coinData.setTo(toList);
        CreateAgentTransaction transaction = (CreateAgentTransaction)
ledgerService.getTx(createTxHash);
        if (null == transaction) {
            throw new NulsRuntimeException(TransactionErrorCode.TX_NOT_EXIST);
        }
        List<Coin> fromList = new ArrayList<>();
        for (int index = 0; index < transaction.getCoinData().getTo().size(); index++) {
            Coin coin = transaction.getCoinData().getTo().get(index);
            if (coin.getNa().equals(agent.getDeposit()) && coin.getLockTime() == -1L) {
                coin.setOwner(ArraysTool.concatenate(transaction.getHash().serialize(), new
VarInt(index).encode()));
                fromList.add(coin);
                break;
            }
        }
        if (fromList.isEmpty()) {
            throw new NulsRuntimeException(KernelErrorCode.DATA_ERROR);
        }
        coinData.setFrom(fromList);

        List<Deposit> deposits =
PocConsensusContext.getChainManager().getMasterChain().getChain().getDepositList();
        List<String> addressList = new ArrayList<>();
        Map<String, Coin> toMap = new HashMap<>();
        long blockHeight = null == hight ? -1 : hight;
```

```
        for (Deposit deposit : deposits) {
            if (deposit.getDelHeight() > 0 && (blockHeight <= 0 || deposit.getDelHeight() <
blockHeight)) {
                continue;
            }
            if (!deposit.getAgentHash().equals(agent.getTxHash())) {
                continue;
            }
            DepositTransaction dtx = (DepositTransaction) ledgerService.getTx(deposit.getTxHash());
            Coin fromCoin = null;
            for (Coin coin : dtx.getCoinData().getTo()) {
                if (!coin.getNa().equals(deposit.getDeposit()) || coin.getLockTime() != -1L) {
                    continue;
                }
                fromCoin = new Coin(ArraysTool.concatenate(dtx.getHash().serialize(), new
VarInt(0).encode()), coin.getNa(), coin.getLockTime());
                fromCoin.setLockTime(-1L);
                fromList.add(fromCoin);
                break;
            }
            String address = AddressTool.getStringAddressByBytes(deposit.getAddress());
            Coin coin = toMap.get(address);
            if (null == coin) {
                if (deposit.getAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
                    Script scriptPubkey = SignatureUtil.createOutputScript(deposit.getAddress());
                    coin = new Coin(scriptPubkey.getProgram(), deposit.getDeposit(), 0);
                } else {
                    coin = new Coin(deposit.getAddress(), deposit.getDeposit(), 0);
                }
                addressList.add(address);
                toMap.put(address, coin);
            } else {
                coin.setNa(coin.getNa().add(fromCoin.getNa()));
            }
        }
        for (String address : addressList) {
            coinData.getTo().add(toMap.get(address));
        }

        return coinData;
    }
```

```
    public static CoinData getStopAgentCoinData(byte[] address, long lockTime) throws
IOException {
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        for (Agent agent : agentList) {
            if (agent.getDelHeight() > 0) {
                continue;
            }
            if (Arrays.equals(address, agent.getAgentAddress())) {
                return getStopAgentCoinData(agent, lockTime);
            }
        }
        return null;
    }

    public static byte[] getStateRoot(BlockHeader blockHeader) {
        if (blockHeader == null || blockHeader.getExtend() == null) {
            return null;
        }
        byte[] stateRoot = blockHeader.getStateRoot();
        if (stateRoot != null && stateRoot.length > 0) {
            return stateRoot;
        }
        try {
            BlockExtendsData extendsData = new BlockExtendsData(blockHeader.getExtend());
            stateRoot = extendsData.getStateRoot();
            if ((stateRoot == null || stateRoot.length == 0) && NulsContext.MAIN_NET_VERSION > 1)
{
                stateRoot = Hex.decode(NulsContext.INITIAL_STATE_ROOT);
            }
            blockHeader.setStateRoot(stateRoot);
            return stateRoot;
        } catch (Exception e) {
            Log.error("parse stateRoot of blockHeader error.", e);
            return null;
        }
    }

    /*public static CoinData getStopMutilAgentCoinData(Agent agent, long lockTime, BlockHeader
blockHeader) throws IOException {
        if (null == agent) {
            return null;
```

```
        }
        NulsDigestData createTxHash = agent.getTxHash();
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        if (agent.getAgentAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
            Script scriptPubkey = SignatureUtil.createOutputScript(agent.getAgentAddress());
            toList.add(new Coin(scriptPubkey.getProgram(), agent.getDeposit(), lockTime));
        } else {
            toList.add(new Coin(agent.getAgentAddress(), agent.getDeposit(), lockTime));
        }
        coinData.setTo(toList);
        CreateAgentTransaction transaction = (CreateAgentTransaction)
ledgerService.getTx(createTxHash);
        if (null == transaction) {
            throw new NulsRuntimeException(TransactionErrorCode.TX_NOT_EXIST);
        }
        List<Coin> fromList = new ArrayList<>();
        for (int index = 0; index < transaction.getCoinData().getTo().size(); index++) {
            Coin coin = transaction.getCoinData().getTo().get(index);
            if (coin.getNa().equals(agent.getDeposit()) && coin.getLockTime() == -1L) {
                coin.setOwner(ArraysTool.concatenate(transaction.getHash().serialize(), new
VarInt(index).encode()));
                fromList.add(coin);
                break;
            }
        }
        if (fromList.isEmpty()) {
            throw new NulsRuntimeException(KernelErrorCode.DATA_ERROR);
        }
        coinData.setFrom(fromList);
        List<Deposit> deposits =
PocConsensusContext.getChainManager().getMasterChain().getChain().getDepositList();
        List<String> addressList = new ArrayList<>();
        Map<String, Coin> toMap = new HashMap<>();
        long blockHeight = null == blockHeader ? -1 : blockHeader.getHeight();
        for (Deposit deposit : deposits) {
            if (deposit.getDelHeight() > 0 && (blockHeight <= 0 || deposit.getDelHeight() <
blockHeight)) {
                continue;
            }
            if (!deposit.getAgentHash().equals(agent.getTxHash())) {
                continue;
```

```
            }
            DepositTransaction dtx = (DepositTransaction) ledgerService.getTx(deposit.getTxHash());
            Coin fromCoin = null;
            for (Coin coin : dtx.getCoinData().getTo()) {
                if (!coin.getNa().equals(deposit.getDeposit()) || coin.getLockTime() != -1L) {
                    continue;
                }
                fromCoin = new Coin(ArraysTool.concatenate(dtx.getHash().serialize(), new
VarInt(0).encode()), coin.getNa(), coin.getLockTime());
                fromCoin.setLockTime(-1L);
                fromList.add(fromCoin);
                break;
            }
            String address = AddressTool.getStringAddressByBytes(deposit.getAddress());
            Coin coin = toMap.get(address);
            if (null == coin) {
                if (deposit.getAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
                    Script scriptPubkey = SignatureUtil.createOutputScript(deposit.getAddress());
                    toList.add(new Coin(scriptPubkey.getProgram(), deposit.getDeposit(), 0));
                } else {
                    coin = new Coin(deposit.getAddress(), deposit.getDeposit(), 0);
                }
                addressList.add(address);
                toMap.put(address, coin);
            } else {
                coin.setNa(coin.getNa().add(fromCoin.getNa()));
            }
        }
        for (String address : addressList) {
            coinData.getTo().add(toMap.get(address));
        }
        return coinData;
    }*/
}


63:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\main\java\io\nuls\consensus\poc\util\ProtocolTransferTool.java
 */
package io.nuls.consensus.poc.util;

import io.nuls.consensus.poc.model.BlockExtendsData;
```

```
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.protocol.base.version.ProtocolContainer;
import io.nuls.protocol.storage.po.BlockProtocolInfoPo;
import io.nuls.protocol.storage.po.ProtocolInfoPo;
import io.nuls.protocol.storage.po.ProtocolTempInfoPo;

public class ProtocolTransferTool {

    public static ProtocolInfoPo toProtocolInfoPo(ProtocolContainer container) {
        ProtocolInfoPo infoPo = new ProtocolInfoPo();
        infoPo.setVersion(container.getVersion());
        infoPo.setPercent(container.getPercent());
        infoPo.setDelay(container.getDelay());
        infoPo.setCurrentDelay(container.getCurrentDelay());
        infoPo.setCurrentPercent(container.getCurrentPercent());
        infoPo.setAddressSet(container.getAddressSet());
        infoPo.setStatus(container.getStatus());
        infoPo.setRoundIndex(container.getRoundIndex());
        infoPo.setEffectiveHeight(container.getEffectiveHeight());
        infoPo.setPrePercent(container.getPrePercent());
        return infoPo;
    }

    public static ProtocolTempInfoPo createProtocolTempInfoPo(BlockExtendsData extendsData) {
        ProtocolTempInfoPo infoPo = new ProtocolTempInfoPo();
        infoPo.setVersion(extendsData.getCurrentVersion());
        infoPo.setDelay(extendsData.getDelay());
        infoPo.setPercent(extendsData.getPercent());
        return infoPo;
    }


    public static BlockProtocolInfoPo toBlockProtocolInfoPo(BlockHeader header,
ProtocolContainer container) {
        BlockProtocolInfoPo infoPo = new BlockProtocolInfoPo();
        infoPo.setBlockHeight(header.getHeight());
        infoPo.setVersion(container.getVersion());
        infoPo.setCurrentDelay(container.getCurrentDelay());
        infoPo.setEffectiveHeight(container.getEffectiveHeight());
        infoPo.setRoundIndex(container.getRoundIndex());
        infoPo.setStatus(container.getStatus());
```

```
        infoPo.setAddressSet(container.getAddressSet());
        infoPo.setPrePercent(container.getPrePercent());
        return infoPo;
    }

    public static void copyFromBlockProtocolInfoPo(BlockProtocolInfoPo infoPo, ProtocolContainer
container) {
        container.setStatus(infoPo.getStatus());
        container.setEffectiveHeight(infoPo.getEffectiveHeight());
        container.setRoundIndex(infoPo.getRoundIndex());
        container.setAddressSet(infoPo.getAddressSet());
        container.setCurrentDelay(infoPo.getCurrentDelay());
        container.setPrePercent(infoPo.getPrePercent());
    }

    public static BlockProtocolInfoPo toBlockProtocolInfoPo(BlockHeader header,
ProtocolTempInfoPo tempInfoPo) {
        BlockProtocolInfoPo infoPo = new BlockProtocolInfoPo();
        infoPo.setBlockHeight(header.getHeight());
        infoPo.setVersion(tempInfoPo.getVersion());
        infoPo.setCurrentDelay(tempInfoPo.getCurrentDelay());
        infoPo.setEffectiveHeight(tempInfoPo.getEffectiveHeight());
        infoPo.setRoundIndex(tempInfoPo.getRoundIndex());
        infoPo.setStatus(tempInfoPo.getStatus());
        infoPo.setAddressSet(tempInfoPo.getAddressSet());
        infoPo.setPrePercent(tempInfoPo.getPrePercent());
        return infoPo;
    }

    public static void copyFromBlockProtocolTempInfoPo(BlockProtocolInfoPo infoPo,
ProtocolTempInfoPo tempInfoPo) {
        tempInfoPo.setStatus(infoPo.getStatus());
        tempInfoPo.setEffectiveHeight(infoPo.getEffectiveHeight());
        tempInfoPo.setRoundIndex(infoPo.getRoundIndex());
        tempInfoPo.setAddressSet(infoPo.getAddressSet());
        tempInfoPo.setCurrentDelay(infoPo.getCurrentDelay());
        tempInfoPo.setPrePercent(infoPo.getPrePercent());
    }
}
```

64:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\BaseChainTest.java

```
 *
 */

package io.nuls.consensus.poc;

import io.nuls.consensus.poc.model.*;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.container.ChainContainer;
import io.nuls.consensus.poc.protocol.tx.DepositTransaction;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.*;
import io.nuls.kernel.script.BlockSignature;

import io.nuls.kernel.utils.AddressTool;

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

import static org.junit.Assert.assertNotNull;

/**
 * Created by ln on 2018/5/7.
 */
public class BaseChainTest extends BaseTest {

    protected Chain chain;
    protected ChainContainer chainContainer;

    protected void initChain() {
        chain = new Chain();

        // new a block header
        BlockHeader blockHeader = new BlockHeader();
        blockHeader.setHeight(0);
        blockHeader.setPreHash(NulsDigestData.calcDigestData("00000000000".getBytes()));
        blockHeader.setTime(1L);
        blockHeader.setTxCount(0);
```

```
// add a round data
BlockRoundData roundData = new BlockRoundData();
roundData.setConsensusMemberCount(1);
roundData.setPackingIndexOfRound(1);
roundData.setRoundIndex(1);
roundData.setRoundStartTime(1L);
try {
    blockHeader.setExtend(roundData.serialize());
} catch (IOException e) {
    throw new NulsRuntimeException(e);
}

chain.setEndBlockHeader(blockHeader);

// new a block of height 0
Block block = new Block();
block.setHeader(blockHeader);

// add the block into chain
chain.getBlockList().add(block);
chain.setStartBlockHeader(blockHeader);
chain.setEndBlockHeader(blockHeader);

// init some agent
List<Agent> agentList = new ArrayList<>();

Transaction<Agent> agentTx = new CreateAgentTransaction();
Agent agent = new Agent();
agent.setPackingAddress(AddressTool.getAddress(ecKey.getPubKey()));
agent.setAgentAddress(AddressTool.getAddress(new ECKey().getPubKey()));
agent.setRewardAddress(AddressTool.getAddress(ecKey.getPubKey()));
agent.setTime(System.currentTimeMillis());
agent.setDeposit(Na.NA.multiply(20000));
agent.setCommissionRate(0.3d);
agent.setBlockHeight(blockHeader.getHeight());

agentTx.setTxData(agent);
agentTx.setTime(agent.getTime());
agentTx.setBlockHeight(blockHeader.getHeight());

NulsSignData signData = signDigest(agentTx.getHash().getDigestBytes(), ecKey);
```

```
    agentTx.setTransactionSignature(signData.getSignBytes());
    agentTx.getTxData().setTxHash(agentTx.getHash());

    // add the agent tx into agent list
    agentList.add(agentTx.getTxData());

    // set the agent list into chain
    chain.setAgentList(agentList);

    // new a deposit
    Deposit deposit = new Deposit();
    deposit.setAddress(AddressTool.getAddress(ecKey.getPubKey()));
    deposit.setAgentHash(agentTx.getHash());
    deposit.setTime(System.currentTimeMillis());
    deposit.setDeposit(Na.NA.multiply(200000));
    deposit.setBlockHeight(blockHeader.getHeight());

    DepositTransaction depositTx = new DepositTransaction();
    depositTx.setTime(deposit.getTime());
    depositTx.setTxData(deposit);
    depositTx.setBlockHeight(blockHeader.getHeight());

    List<Deposit> depositList = new ArrayList<>();

    depositList.add(depositTx.getTxData());

    chain.setDepositList(depositList);

    chain.setYellowPunishList(new ArrayList<>());
    chain.setRedPunishList(new ArrayList<>());

    chainContainer = new ChainContainer(chain);
}

protected Block newBlock(Block preBlock) {

    assertNotNull(preBlock);
    assertNotNull(preBlock.getHeader());

    BlockHeader blockHeader = new BlockHeader();
    blockHeader.setHeight(preBlock.getHeader().getHeight() + 1);
    blockHeader.setPreHash(preBlock.getHeader().getHash());
```

```
        blockHeader.setTxCount(1);

        MeetingRound round = chainContainer.initRound();

        BlockExtendsData nextRoundData = new BlockExtendsData();

        nextRoundData.setRoundIndex(round.getIndex() + 1);
        nextRoundData.setRoundStartTime(round.getEndTime());

        MeetingRound currentRound =
chainContainer.getRoundManager().getNextRound(nextRoundData, false);

        MeetingMember member =
currentRound.getMember(AddressTool.getAddress(ecKey.getPubKey()));
        blockHeader.setTime(member.getPackEndTime());

        // add a round data
        BlockRoundData roundData = new BlockRoundData(preBlock.getHeader().getExtend());
        roundData.setConsensusMemberCount(currentRound.getMemberCount());
        roundData.setPackingIndexOfRound(member.getPackingIndexOfRound());
        roundData.setRoundIndex(currentRound.getIndex());
        roundData.setRoundStartTime(currentRound.getStartTime());
        try {
            blockHeader.setExtend(roundData.serialize());
        } catch (IOException e) {
            throw new NulsRuntimeException(e);
        }

        // new a block of height 0
        Block block = new Block();
        block.setHeader(blockHeader);

        List<Transaction> txs = new ArrayList<>();
        block.setTxs(txs);
        txs.add(new TestTransaction());

        List<NulsDigestData> txHashList = block.getTxHashList();

        blockHeader.setMerkleHash(NulsDigestData.calcMerkleDigestData(txHashList));

        NulsSignData signData = signDigest(blockHeader.getHash().getDigestBytes(), ecKey);
```

```
        BlockSignature sig = new BlockSignature();
        sig.setSignData(signData);
        sig.setPublicKey(ecKey.getPubKey());

        blockHeader.setBlockSignature(sig);

        return block;
    }
}
```

65:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\BaseTest.java

```
 *
 */

package io.nuls.consensus.poc;

import io.nuls.consensus.poc.customer.ConsensusBlockServiceImpl;
import io.nuls.consensus.poc.customer.ConsensusDownloadServiceImpl;
import io.nuls.consensus.poc.customer.ConsensusNetworkService;
import io.nuls.consensus.poc.model.BlockRoundData;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.core.SpringLiteContext;
import io.nuls.kernel.model.*;

import io.nuls.kernel.script.BlockSignature;
import io.nuls.network.service.NetworkService;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.DownloadService;
import org.junit.BeforeClass;

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

/**
 * Created by ln on 2018/5/8.
 */
public class BaseTest {

    protected static BlockService blockService;
```

```
protected ECKey ecKey = new ECKey();

@BeforeClass
public static void initClass() {
    try {
        try {
            blockService = SpringLiteContext.getBean(BlockService.class);
        } catch (Exception e) {
            SpringLiteContext.putBean(ConsensusBlockServiceImpl.class, false);
        }
        try {
            SpringLiteContext.getBean(DownloadService.class);
        } catch (Exception e) {
            SpringLiteContext.putBean(ConsensusDownloadServiceImpl.class, false);
        }
        try {
            SpringLiteContext.getBean(NetworkService.class);
        } catch (Exception e) {
            SpringLiteContext.putBean(ConsensusNetworkService.class, false);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

protected Block createBlock() {
    // new a block header
    BlockHeader blockHeader = new BlockHeader();
    blockHeader.setHeight(0);
    blockHeader.setPreHash(NulsDigestData.calcDigestData("00000000000".getBytes()));
    blockHeader.setTime(1L);
    blockHeader.setTxCount(1);
    blockHeader.setMerkleHash(NulsDigestData.calcDigestData(new byte[20]));

    // add a round data
    BlockRoundData roundData = new BlockRoundData();
    roundData.setConsensusMemberCount(1);
    roundData.setPackingIndexOfRound(1);
    roundData.setRoundIndex(1);
    roundData.setRoundStartTime(1L);
    try {
        blockHeader.setExtend(roundData.serialize());
```

```
        } catch (IOException e) {
            throw new NulsRuntimeException(e);
        }

        // new a block of height 0
        Block block = new Block();
        block.setHeader(blockHeader);

        List<Transaction> txs = new ArrayList<>();
        block.setTxs(txs);

        Transaction tx = new TestTransaction();
        txs.add(tx);

        List<NulsDigestData> txHashList = block.getTxHashList();

        blockHeader.setMerkleHash(NulsDigestData.calcMerkleDigestData(txHashList));

        NulsSignData signData = signDigest(blockHeader.getHash().getDigestBytes(), ecKey);

        BlockSignature sig = new BlockSignature();
        sig.setSignData(signData);
        sig.setPublicKey(ecKey.getPubKey());
        blockHeader.setBlockSignature(sig);

        return block;
    }

    protected NulsSignData signDigest(byte[] digest, ECKey ecKey) {
        byte[] signbytes = ecKey.sign(digest);
        NulsSignData nulsSignData = new NulsSignData();
        nulsSignData.setSignAlgType(NulsSignData.SIGN_ALG_ECC);
        nulsSignData.setSignBytes(signbytes);

        return nulsSignData;
    }
}

66:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\cache\TxMemoryPoolTest.java
 *
 */
```

```
package io.nuls.consensus.poc.cache;

import io.nuls.consensus.poc.TestTransaction;
import io.nuls.kernel.model.Transaction;
import org.junit.Test;

import java.util.List;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/6.
 */
public class TxMemoryPoolTest {

    private TxMemoryPool txMemoryPool = TxMemoryPool.getInstance();

    @Test
    public void test() {
        assertNotNull(txMemoryPool);

        Transaction tx = new TestTransaction();

        boolean success = txMemoryPool.add(tx, false);

        assertTrue(success);

        Transaction tempTx = txMemoryPool.get();

        assertEquals(tx, tempTx);

        tempTx = txMemoryPool.get();

        assertNull(tempTx);

//        tempTx = txMemoryPool.get(tx.getTx().getHash());
//        assertNull(tempTx);
//
//        txMemoryPool.add(tx, false);
//        tempTx = txMemoryPool.get(tx.getTx().getHash());
//        assertEquals(tempTx, tx);
```

```
//
//        tempTx = txMemoryPool.get(tx.getTx().getHash());
//        assertEquals(tempTx, tx);

        Transaction tx2 = new TestTransaction();
        txMemoryPool.add(tx2, true);

        tempTx = txMemoryPool.get();

        assertNotNull(tempTx);
        assertEquals(tempTx, tx);
        tempTx = txMemoryPool.get();

        assertNotNull(tempTx);
        assertEquals(tempTx, tx2);

        tempTx = txMemoryPool.get();
        assertNull(tempTx);

        txMemoryPool.add(tx2, true);
        List<Transaction> list = txMemoryPool.getAll();
        assertEquals(list.size(), 0);

        list = txMemoryPool.getAllOrphan();
        assertEquals(list.size(), 1);

        success = txMemoryPool.exist(tx.getHash());
        assertFalse(success);

        success = txMemoryPool.exist(tx2.getHash());
        assertTrue(success);

        success = txMemoryPool.remove(tx2.getHash());
        assertTrue(success);

        success = txMemoryPool.exist(tx2.getHash());
        assertFalse(success);
    }
}
```

67:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\container\ChainContainerTest.java

```
 *
 */

package io.nuls.consensus.poc.container;

import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.BaseChainTest;
import io.nuls.consensus.poc.model.Chain;
import io.nuls.consensus.poc.protocol.tx.DepositTransaction;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.AddressTool;
import org.junit.Before;
import org.junit.Test;

import java.util.List;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/7.
 */
public class ChainContainerTest extends BaseChainTest {

    @Before
    public void initData() {
        initChain();

        chainContainer = new ChainContainer(chain);

        chainContainer.getOrResetCurrentRound(false);
    }

    @Test
    public void testInit() {
        assertNotNull(chain);
        assertNotNull(chainContainer);

        assertNotNull(chainContainer.getCurrentRound());

        assertEquals(chainContainer.getCurrentRound().getMemberCount(), 1);
```

```
}

@Test
public void testAddBlock() {
    Block bestBlock = chainContainer.getBestBlock();
    assertNotNull(bestBlock);

    Block newBlock = newBlock(bestBlock);

    Result success = chainContainer.verifyBlock(newBlock);
    assertTrue(success.isSuccess());

    bestBlock = chainContainer.getBestBlock();
    assertEquals(bestBlock.getHeader().getHeight(), 0L);

    for(int i = 0 ; i < 100 ; i++) {

        newBlock = newBlock(bestBlock);

        success = chainContainer.verifyAndAddBlock(newBlock, false, true);
        assertTrue(success.isSuccess());

        bestBlock = chainContainer.getBestBlock();
        assertEquals(bestBlock.getHeader().getHeight(), 1L + i);
    }

    System.out.println();

}

@Test
public void testAddAgent() {

    assertEquals(chainContainer.getCurrentRound().getMemberCount(), 1);

    Block bestBlock = chainContainer.getBestBlock();
    assertNotNull(bestBlock);

    Block newBlock = newBlock(bestBlock);

    addTx(newBlock);
```

```
    Result success = chainContainer.verifyAndAddBlock(newBlock, false, true);
    assertTrue(success.isSuccess());

    bestBlock = chainContainer.getBestBlock();
    newBlock = newBlock(bestBlock);
    success = chainContainer.verifyAndAddBlock(newBlock, false, true);
    assertTrue(success.isSuccess());

    bestBlock = chainContainer.getBestBlock();
    newBlock = newBlock(bestBlock);
    success = chainContainer.verifyAndAddBlock(newBlock, false, true);
    assertTrue(success.isSuccess());

    bestBlock = chainContainer.getBestBlock();
    newBlock = newBlock(bestBlock);
    success = chainContainer.verifyAndAddBlock(newBlock, false, true);
    assertTrue(success.isSuccess());


    assertEquals(chainContainer.getCurrentRound().getMemberCount(), 2);

}

@Test
public void testRollback() {
    testAddAgent();

    assertEquals(chainContainer.getCurrentRound().getMemberCount(), 2);
    for(int i = 3 ; i > 0 ; i--) {
        Block bestBlock = chainContainer.getBestBlock();
        boolean success = chainContainer.rollback(bestBlock);
        assert(success);
    }
    assertEquals(chainContainer.getCurrentRound().getMemberCount(), 1);
}

@Test
public void testGetBeforeTheForkChain() {

    Block forkBlock = null;

    for(int i = 0 ; i < 20 ; i++) {
```

```
            Block bestBlock = chainContainer.getBestBlock();
            Block newBlock = newBlock(bestBlock);

            Result success = chainContainer.verifyAndAddBlock(newBlock, false, true);
            assertTrue(success.isSuccess());

            bestBlock = chainContainer.getBestBlock();
            assertEquals(bestBlock.getHeader().getHeight(), 1L + i);

            if(i == 10) {
                forkBlock = bestBlock;
            }
        }

        Chain chain = new Chain();
        chain.setEndBlockHeader(forkBlock.getHeader());
        chain.setStartBlockHeader(forkBlock.getHeader());
        chain.getBlockList().add(forkBlock);

        Block newBlock = newBlock(forkBlock);
        chain.setEndBlockHeader(newBlock.getHeader());
        chain.getBlockList().add(newBlock);

        ChainContainer otherChainContainer = new ChainContainer(chain);
        ChainContainer newForkChainContainer =
chainContainer.getBeforeTheForkChain(otherChainContainer);
        assertEquals(newForkChainContainer.getBestBlock().getHeader().getHeight(), 10L);

    }

    @Test
    public void testGetAfterTheForkChain() {
        Block forkBlock = null;

        for(int i = 0 ; i < 30 ; i++) {

            Block bestBlock = chainContainer.getBestBlock();
            Block newBlock = newBlock(bestBlock);

            Result success = chainContainer.verifyAndAddBlock(newBlock, false, true);
            assertTrue(success.isSuccess());
```

```
            bestBlock = chainContainer.getBestBlock();
            assertEquals(bestBlock.getHeader().getHeight(), 1L + i);

            if(i == 20) {
                forkBlock = bestBlock;
            }
        }

        Chain chain = new Chain();
        chain.setEndBlockHeader(forkBlock.getHeader());
        chain.setStartBlockHeader(forkBlock.getHeader());
        chain.getBlockList().add(forkBlock);

        Block newBlock = newBlock(forkBlock);
        chain.setEndBlockHeader(newBlock.getHeader());
        chain.getBlockList().add(newBlock);

        ChainContainer otherChainContainer = new ChainContainer(chain);
        ChainContainer newForkChainContainer =
chainContainer.getAfterTheForkChain(otherChainContainer);
        assertEquals(newForkChainContainer.getBestBlock().getHeader().getHeight(), 30L);

    }

    protected void addTx(Block block) {

        BlockHeader blockHeader = block.getHeader();
        List<Transaction> txs = block.getTxs();

        Transaction<Agent> agentTx = new CreateAgentTransaction();
        Agent agent = new Agent();
        agent.setPackingAddress(AddressTool.getAddress(ecKey.getPubKey()));
        agent.setAgentAddress(AddressTool.getAddress(ecKey.getPubKey()));
        agent.setTime(System.currentTimeMillis());
        agent.setDeposit(Na.NA.multiply(20000));
        agent.setCommissionRate(0.3d);
        agent.setBlockHeight(blockHeader.getHeight());

        agentTx.setTxData(agent);
        agentTx.setTime(agent.getTime());
        agentTx.setBlockHeight(blockHeader.getHeight());
```

```
        NulsSignData signData = signDigest(agentTx.getHash().getDigestBytes(), ecKey);

        agentTx.setTransactionSignature(signData.getSignBytes());

        // add the agent tx into agent list
        txs.add(agentTx);

        // new a deposit
        Deposit deposit = new Deposit();
        deposit.setAddress(AddressTool.getAddress(ecKey.getPubKey()));
        deposit.setAgentHash(agentTx.getHash());
        deposit.setTime(System.currentTimeMillis());
        deposit.setDeposit(Na.NA.multiply(200000));
        deposit.setBlockHeight(blockHeader.getHeight());

        DepositTransaction depositTx = new DepositTransaction();
        depositTx.setTime(deposit.getTime());
        depositTx.setTxData(deposit);
        depositTx.setBlockHeight(blockHeader.getHeight());

        txs.add(depositTx);
    }
}

68:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\customer\ConsensusBlockServiceImpl.java
 *
 */

package io.nuls.consensus.poc.customer;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.network.model.Node;
import io.nuls.protocol.model.SmallBlock;
import io.nuls.protocol.service.BlockService;

import java.util.ArrayList;
```

```
/**
 * Created by ln on 2018/5/8.
 */
public class ConsensusBlockServiceImpl implements BlockService {
    @Override
    public Result<Block> getGengsisBlock() {
        return null;
    }

    @Override
    public Result<Block> getBestBlock() {
        return null;
    }

    @Override
    public Result<BlockHeader> getBestBlockHeader() {
        return null;
    }

    @Override
    public Result<BlockHeader> getBlockHeader(long height) {
        return null;
    }

    @Override
    public Result<BlockHeader> getBlockHeader(NulsDigestData hash) {
        return null;
    }

    @Override
    public Result<Block> getBlock(NulsDigestData hash) {
        Result result = new Result(true, null);

        Block block = new Block();
        BlockHeader blockHeader = new BlockHeader();
        blockHeader.setHash(hash);
        block.setHeader(blockHeader);
        block.setTxs(new ArrayList<>());

        result.setData(block);
```

```
        return result;
    }

    @Override
    public Result<Block> getBlock(NulsDigestData hash, boolean isNeedContractTransfer) {
        return null;
    }

    @Override
    public Result<Block> getBlock(long height) {
        return null;
    }

    @Override
    public Result<Block> getBlock(long height, boolean isNeedContractTransfer) {
        return null;
    }

    @Override
    public Result saveBlock(Block block) throws NulsException {
        return new Result(true, null);
    }

    @Override
    public Result rollbackBlock(Block block) throws NulsException {
        return new Result(false, null);
    }

    @Override
    public Result forwardBlock(NulsDigestData blockHash, Node excludeNode) {
        return null;
    }

    @Override
    public Result broadcastBlock(SmallBlock block) {
        return null;
    }

}
```

69:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\customer\ConsensusDownloadServiceImpl.java

```
 *
 */

package io.nuls.consensus.poc.customer;

import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.network.model.Node;
import io.nuls.protocol.model.TxGroup;
import io.nuls.protocol.service.DownloadService;

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

/**
 * Created by ln on 2018/5/8.
 */
public class ConsensusDownloadServiceImpl implements DownloadService {

    private boolean isDownloadSuccess;

    @Override
    public Result<Block> downloadBlock(NulsDigestData hash, Node node) {
        Result<Block> result = new Result<>();

        Block block = new Block();
        block.setTxs(new ArrayList<>());

        BlockHeader blockHeader = new BlockHeader();
        blockHeader.setHash(hash);
        block.setHeader(blockHeader);

        result.setData(block);

        return result;
    }

    @Override
    public Result isDownloadSuccess() {
        return new Result(isDownloadSuccess, null);
```

```
    }

    @Override
    public Result reset() {
        return null;
    }

    public void setDownloadSuccess(boolean downloadSuccess) {
        isDownloadSuccess = downloadSuccess;
    }
}

70:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\customer\ConsensusNetworkService.java
 *
 */

package io.nuls.consensus.poc.customer;

import io.nuls.kernel.model.BaseNulsData;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.model.BroadcastResult;
import io.nuls.network.model.Node;
import io.nuls.network.model.NodeGroup;
import io.nuls.network.service.NetworkService;

import java.util.Collection;
import java.util.List;
import java.util.Map;

/**
 * Created by ln on 2018/5/9.
 */
public class ConsensusNetworkService implements NetworkService {
    @Override
    public void removeNode(String nodeId) {

    }

    @Override
    public Node getNode(String nodeId) {
        return null;
```

```
    }

    @Override
    public Map<String, Node> getNodes() {
        return null;
    }

    @Override
    public Collection<Node> getAvailableNodes() {
        return null;
    }

    @Override
    public List<Node> getCanConnectNodes() {
        // todo auto-generated method stub
        return null;
    }

    @Override
    public NodeGroup getNodeGroup(String groupName) {
        return null;
    }

    @Override
    public BroadcastResult sendToAllNode(BaseNulsData event, boolean asyn, int percent) {
        return null;
    }

    @Override
    public BroadcastResult sendToAllNode(BaseNulsData event, Node excludeNode, boolean
asyn, int percent) {
        return null;
    }

    @Override
    public BroadcastResult sendToNode(BaseNulsData event, Node node, boolean asyn) {
        return null;
    }

    @Override
    public BroadcastResult sendToGroup(BaseNulsData event, String groupName, boolean asyn) {
        return null;
```

```
    }

    @Override
    public BroadcastResult sendToGroup(BaseNulsData event, String groupName, Node
excludeNode, boolean asyn) {
        return null;
    }

    @Override
    public void reset() {
    }

    @Override
    public NetworkParam getNetworkParam() {
        return null;
    }
}
```

71:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\manager\ChainManagerTest.java

```
 *
 */

package io.nuls.consensus.poc.manager;

import io.nuls.consensus.poc.BaseTest;
import io.nuls.consensus.poc.container.ChainContainer;
import io.nuls.consensus.poc.model.Chain;
import io.nuls.kernel.model.Block;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/8.
 */
public class ChainManagerTest extends BaseTest {

    private ChainManager chainManager;

    @Before
```

```
public void init() {
    chainManager = new ChainManager();
}

@Test
public void testNewOrphanChain() {

    assertNotNull(chainManager);

    Block block = createBlock();
    chainManager.newOrphanChain(block);

    assertEquals(1, chainManager.getOrphanChains().size());
}

@Test
public void testGetBestBlockHeight() {
    assertNotNull(chainManager);

    Block block = createBlock();

    ChainContainer masterChain = new ChainContainer(new Chain());
    chainManager.setMasterChain(masterChain);
    masterChain.getChain().setEndBlockHeader(block.getHeader());
    masterChain.getChain().setStartBlockHeader(block.getHeader());
    masterChain.getChain().getBlockList().add(block);
    masterChain.getChain().getBlockHeaderList().add(block.getHeader());

    assertEquals(0L, chainManager.getBestBlockHeight());
}

@Test
public void testCheckIsBeforeOrphanChainAndAdd() {

    testGetBestBlockHeight();

    Block block = createBlock();

    Block block1 = createBlock();
    block1.getHeader().setHeight(1L);
    block1.getHeader().setPreHash(block.getHeader().getHash());
```

```
    ChainContainer orphanChain = new ChainContainer(new Chain());
    orphanChain.getChain().setEndBlockHeader(block1.getHeader());
    orphanChain.getChain().setStartBlockHeader(block1.getHeader());
    orphanChain.getChain().getBlockList().add(block1);
    orphanChain.getChain().getBlockHeaderList().add(block1.getHeader());

    chainManager.getOrphanChains().add(orphanChain);

    assertEquals(1, chainManager.getOrphanChains().size());


    boolean success = chainManager.checkIsBeforeOrphanChainAndAdd(block);

    assertTrue(success);
}

@Test
public void testCheckIsAfterOrphanChainAndAdd() {

    testGetBestBlockHeight();

    Block block = createBlock();

    Block block1 = createBlock();
    block1.getHeader().setHeight(1L);
    block1.getHeader().setPreHash(block.getHeader().getHash());

    ChainContainer orphanChain = new ChainContainer(new Chain());
    orphanChain.getChain().setEndBlockHeader(block.getHeader());
    orphanChain.getChain().setStartBlockHeader(block.getHeader());
    orphanChain.getChain().getBlockList().add(block);
    orphanChain.getChain().getBlockHeaderList().add(block.getHeader());

    chainManager.getOrphanChains().add(orphanChain);

    assertEquals(1, chainManager.getOrphanChains().size());


    boolean success = chainManager.checkIsAfterOrphanChainAndAdd(block1);

    assertTrue(success);
}
```

```
}

72:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\manager\RoundManagerTest.java
 *
 */

package io.nuls.consensus.poc.manager;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.BaseChainTest;
import io.nuls.consensus.poc.model.Chain;
import io.nuls.consensus.poc.model.MeetingRound;
import io.nuls.protocol.constant.ProtocolConstant;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/7.
 */
public class RoundManagerTest extends BaseChainTest {

    private RoundManager roundManager;

    @Before
    public void initData() {
        initChain();

        roundManager = new RoundManager(chain);
    }

    @Test
    public void testAddRound() {
        assertNotNull(roundManager);
        assertNotNull(roundManager.getChain());

        assertEquals(0, roundManager.getRoundList().size());
        MeetingRound round = new MeetingRound();
        roundManager.addRound(round);
```

```
    assertEquals(1, roundManager.getRoundList().size());

    assertEquals(round, roundManager.getCurrentRound());

    MeetingRound round2 = new MeetingRound();
    roundManager.addRound(round2);

    assertEquals(2, roundManager.getRoundList().size());

    assertNotEquals(round, roundManager.getCurrentRound());

    assertEquals(round2, roundManager.getCurrentRound());
}

@Test
public void testGetRoundByIndex() {
    assertNotNull(roundManager);
    assertNotNull(roundManager.getChain());

    long index = 1002L;

    assertEquals(0, roundManager.getRoundList().size());
    MeetingRound round = new MeetingRound();
    round.setIndex(index);
    roundManager.addRound(round);

    assertEquals(1, roundManager.getRoundList().size());

    MeetingRound round2 = roundManager.getRoundByIndex(index);
    assertNotNull(round2);
    assertEquals(round, round2);

}

@Test
public void testClearRound() {
    MeetingRound round = new MeetingRound();
    round.setIndex(1l);
    roundManager.addRound(round);
    round = new MeetingRound();
    round.setIndex(2l);
```

```
    roundManager.addRound(round);
    round = new MeetingRound();
    round.setIndex(3l);
    roundManager.addRound(round);
    round = new MeetingRound();
    round.setIndex(4l);
    roundManager.addRound(round);
    round = new MeetingRound();
    round.setIndex(5l);
    roundManager.addRound(round);
    round = new MeetingRound();
    round.setIndex(6l);
    roundManager.addRound(round);
    round = new MeetingRound();
    round.setIndex(7l);
    roundManager.addRound(round);

    assertEquals(7, roundManager.getRoundList().size());
    assertEquals(7L, roundManager.getCurrentRound().getIndex());

    boolean success = roundManager.clearRound(3);
    assert(success);

    assertEquals(3, roundManager.getRoundList().size());
    assertEquals(7L, roundManager.getCurrentRound().getIndex());
}

@Test
public void testInitRound() {
    assertNotNull(roundManager);
    assertNotNull(roundManager.getChain());

    Chain chain = roundManager.getChain();

    assertNotNull(chain.getEndBlockHeader());
    assert(chain.getBlockList().size() > 0);

    MeetingRound round = roundManager.initRound();

    assertNotNull(round);

    assertEquals(round.getIndex(), 2L);
```

```
        Assert.assertEquals(round.getStartTime(),
ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS + 1L);

        MeetingRound round2 = roundManager.getNextRound(null, false);
        assertNotNull(round2);
        assertEquals(round.getIndex(), round2.getIndex());
        assertEquals(round.getStartTime(), round2.getStartTime());

        round2 = roundManager.getNextRound(null, true);
        assertNotNull(round2);
        assert(round.getIndex() < round2.getIndex());
        assert(round.getStartTime() < round2.getStartTime());
        assertEquals("", 0d, round2.getTotalWeight(), 2200000d);
    }
}

73:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\model\ChainTest.java
 *
 */

package io.nuls.consensus.poc.model;

import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/7.
 */
public class ChainTest {

    @Test
    public void test() {
        Chain chain = new Chain();

        assertNotNull(chain.getId());

        assertNull(chain.getBestBlock());
```

```
        Block block = new Block();
        BlockHeader blockHeader = new BlockHeader();
        blockHeader.setHeight(100l);
        block.setHeader(blockHeader);
        chain.getBlockList().add(block);

        Block bestBlock = chain.getBestBlock();
        assertNotNull(bestBlock);
        assertEquals(bestBlock.getHeader().getHeight(), 100l);
    }
}
```

74:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\model\MeetingMemberTest.java

```
 *
 */

package io.nuls.consensus.poc.model;

import org.junit.Test;

/**
 * Created by ln on 2018/5/7.
 */
public class MeetingMemberTest {

    @Test
    public void testSort() {
        MeetingMember member = new MeetingMember();
        member.setRoundStartTime(0l);
        member.setPackingAddress(new byte[20]);

        MeetingMember member2 = new MeetingMember();
        member2.setRoundStartTime(0l);
        member2.setPackingAddress(new byte[24]);

        assert(member.compareTo(member2) > 0);

        member.setRoundStartTime(1003l);
        member2.setRoundStartTime(1003l);

        assert(member.compareTo(member2) < 0);
```

```
    }
}

75:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\model\MeetingRoundTest.java
 *
 */

package io.nuls.consensus.poc.model;

import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.protocol.constant.ProtocolConstant;
import org.junit.Test;

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

import static org.junit.Assert.assertEquals;

/**
 * Created by ln on 2018/5/7.
 */
public class MeetingRoundTest {

    private long roundStartTime = System.currentTimeMillis();

    @Test
    public void test() {
        List<MeetingMember> meetingMemberList = getMemberList();

        MeetingRound round = new MeetingRound();
        round.setStartTime(roundStartTime);
        round.init(meetingMemberList);

        assertEquals(round.getMemberCount(), meetingMemberList.size());
        assertEquals("error", 1010d, round.getTotalWeight(), 2);
        assertEquals(meetingMemberList.size() *
ProtocolConstant.BLOCK_TIME_INTERVAL_MILLIS + roundStartTime, round.getEndTime());

        System.out.println(round.toString());
```

```
    }

    private List<MeetingMember> getMemberList() {
        List<MeetingMember> meetingMemberList = new ArrayList<>();

        for (int i = 0; i < 10; i++) {
            MeetingMember member = new MeetingMember();
            member.setRoundStartTime(roundStartTime);
            member.setPackingAddress(AddressTool.getAddress(new ECKey().getPubKey()));
            member.setOwnDeposit(Na.NA);
            member.setCreditVal(0.1 * (i + 1));
            member.setCommissionRate(0.2d);
            member.setTotalDeposit(Na.NA.multiply(100));
            member.setRoundIndex(1l);
            meetingMemberList.add(member);
        }

        return meetingMemberList;
    }
}

76:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\module\PocConsensusModuleBootstrapTest.java
 *
 */

package io.nuls.consensus.poc.module;

import io.nuls.consensus.poc.BaseTest;
import io.nuls.consensus.poc.module.impl.PocConsensusModuleBootstrap;
import io.nuls.kernel.MicroKernelBootstrap;
import org.junit.Test;

/**
 * Created by ln on 2018/5/7.
 */
public class PocConsensusModuleBootstrapTest extends BaseTest {

    private PocConsensusModuleBootstrap bootstrap = new PocConsensusModuleBootstrap();

    @Test
```

```
    public void testStartModule() throws Exception {

        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();

        bootstrap.init();
        bootstrap.start();
    }
}
```

77:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\process\RegisterAgentProcessTest.java

```
 *
 */

package io.nuls.consensus.poc.process;

import io.nuls.consensus.poc.BaseTest;
import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.tx.processor.CreateAgentTxProcessor;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;
import org.junit.Before;
import org.junit.Test;

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

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/10.
 */
public class RegisterAgentProcessTest extends BaseTest {
```

```
private CreateAgentTxProcessor registerAgentProcess;

private CreateAgentTransaction tx = newTx();

@Before
public void init() {
    registerAgentProcess = NulsContext.getServiceBean(CreateAgentTxProcessor.class);
}

@Test
public void testOnRollback() {
    assertNotNull(tx);
    assertNotNull(registerAgentProcess);

    testOnCommit();
    Result result = registerAgentProcess.onRollback(tx, null);
    assert (result.isSuccess());
}

@Test
public void testOnCommit() {
    assertNotNull(tx);
    assertNotNull(registerAgentProcess);

    Result result = registerAgentProcess.onCommit(tx, null);
    assert (result.isSuccess());
}

@Test
public void testConflictDetect() {
    assertNotNull(tx);
    assertNotNull(registerAgentProcess);

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

    ValidateResult result = registerAgentProcess.conflictDetect(list);
    assertTrue(result.isSuccess());

    list.add(tx);

    result = registerAgentProcess.conflictDetect(list);
    assertTrue(result.isSuccess());
```

```
        list.add(newTx());
        result = registerAgentProcess.conflictDetect(list);
        assertTrue(result.isSuccess());

        list.add(tx);

        result = registerAgentProcess.conflictDetect(list);
        assertFalse(result.isSuccess());
    }

    private CreateAgentTransaction newTx() {
        CreateAgentTransaction tx = new CreateAgentTransaction();

        Agent agent = new Agent();

        tx.setTxData(agent);

        byte[] address = AddressTool.getAddress(ecKey.getPubKey());
        byte[] address1 = AddressTool.getAddress(new ECKey().getPubKey());
        byte[] address2 = AddressTool.getAddress(new ECKey().getPubKey());
        agent.setRewardAddress(address);
        agent.setPackingAddress(address1);
        agent.setAgentAddress(address2);
        agent.setDeposit(PocConsensusProtocolConstant.AGENT_DEPOSIT_LOWER_LIMIT);

        tx.setTime(System.currentTimeMillis());

        return tx;
    }
}

78:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\provider\BlockQueueProviderTest.java
 *
 */

package io.nuls.consensus.poc.provider;

import io.nuls.consensus.poc.customer.ConsensusDownloadServiceImpl;
import io.nuls.consensus.poc.BaseTest;
import io.nuls.consensus.poc.constant.BlockContainerStatus;
```

```
import io.nuls.consensus.poc.container.BlockContainer;
import io.nuls.kernel.lite.core.SpringLiteContext;
import io.nuls.kernel.model.Block;
import io.nuls.protocol.service.DownloadService;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/5.
 */
public class BlockQueueProviderTest extends BaseTest {

    private BlockQueueProvider blockQueueProvider = BlockQueueProvider.getInstance();
    private ConsensusDownloadServiceImpl downloadService;

    @Before
    public void init() {
        downloadService = (ConsensusDownloadServiceImpl)
SpringLiteContext.getBean(DownloadService.class);
        blockQueueProvider.clear();
    }

    @After
    public void destroy() {
        blockQueueProvider.clear();
    }

    @Test
    public void testPut() {
        assertNotNull(blockQueueProvider);

        assertEquals(0, blockQueueProvider.size());

        Block block = new Block();
        boolean result = blockQueueProvider.put(new BlockContainer(block,
BlockContainerStatus.RECEIVED));
        assertTrue(result);

        assertEquals(1, blockQueueProvider.size());
```

```
    }

    @Test
    public void testGet() {
        assertNotNull(blockQueueProvider);

        assertEquals(0, blockQueueProvider.size());

        if(downloadService.isDownloadSuccess().isSuccess()) {
            downloadService.setDownloadSuccess(false);
        }

        Block block = new Block();
        boolean result = blockQueueProvider.put(new BlockContainer(block,
BlockContainerStatus.RECEIVED));
        assertTrue(result);

        assertEquals(1, blockQueueProvider.size());

        BlockContainer blockContainer = blockQueueProvider.get();

        assertNull(blockContainer);

        downloadService.setDownloadSuccess(true);

        blockContainer = blockQueueProvider.get();

        assertNotNull(blockContainer);

        assertEquals(blockContainer.getBlock(), block);
        assertEquals(blockContainer.getStatus(), BlockContainerStatus.DOWNLOADING);
        assertEquals(0, blockQueueProvider.size());

        block = new Block();
        result = blockQueueProvider.put(new BlockContainer(block,
BlockContainerStatus.RECEIVED));
        assertTrue(result);

        blockContainer = blockQueueProvider.get();
        assertNotNull(blockContainer);
        assertEquals(blockContainer.getBlock(), block);
```

```
        assertEquals(blockContainer.getStatus(), BlockContainerStatus.RECEIVED);

    }

    @Test
    public void testSizeAndClear() {
        assertNotNull(blockQueueProvider);

        assertEquals(0, blockQueueProvider.size());

        Block block = new Block();
        boolean result = blockQueueProvider.put(new BlockContainer(block,
BlockContainerStatus.RECEIVED));
        assertTrue(result);

        assertEquals(1, blockQueueProvider.size());

        blockQueueProvider.clear();

        assertEquals(0, blockQueueProvider.size());

    }
}

79:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\storage\service\ConsensusPocServiceTest.java
 *
 */

package io.nuls.consensus.poc.storage.service;

import io.nuls.consensus.poc.TestTransaction;
import io.nuls.consensus.poc.BaseTest;
import io.nuls.consensus.poc.cache.TxMemoryPool;
import io.nuls.consensus.poc.service.impl.ConsensusPocServiceImpl;
import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.*;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
```

```
import io.nuls.kernel.validate.ValidatorManager;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

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

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/5.
 */
public class ConsensusPocServiceTest extends BaseTest {

    private ConsensusService service = new ConsensusPocServiceImpl();

    @Before
    public void setUp() throws Exception {
        TxMemoryPool.getInstance().clear();
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void newTx() throws Exception {

        assertNotNull(service);

        // new a tx
        Transaction tx = new TestTransaction();
        CoinData coinData = new CoinData();
        List<Coin> fromList = new ArrayList<>();
        fromList.add(new Coin(new byte[20], Na.NA, 0L));
        coinData.setFrom(fromList);
        tx.setCoinData(coinData);
        tx.setTime(1l);

        assertNotNull(tx);
```

```
        assertNotNull(tx.getHash());
        assertEquals(tx.getHash().getDigestHex(),
"00204a54f8b12b75c3c1fe5f261416adaf1a1b906ccf5673bb7a133ede5a0a4c56f8");

        Result result = service.newTx(tx);
        assertNotNull(result);
        assertTrue(result.isSuccess());
        assertFalse(result.isFailed());

        //test orphan
        NulsDataValidator<TestTransaction> testValidator = new
NulsDataValidator<TestTransaction>() {
            @Override
            public ValidateResult validate(TestTransaction data) {
                if
(data.getHash().getDigestHex().equals("0020e27ee243921bf482d7b62b6ee63c7ab1938953c8343
18b79fa3204c5c869e26b")) {
                    return ValidateResult.getFailedResult("test.transaction",
TransactionErrorCode.ORPHAN_TX);
                } else {
                    return ValidateResult.getSuccessResult();
                }
            }
        };
        ValidatorManager.addValidator(TestTransaction.class, testValidator);

        tx = new TestTransaction();
        tx.setTime(2l);
        assertEquals(tx.getHash().getDigestHex(),
"0020e27ee243921bf482d7b62b6ee63c7ab1938953c834318b79fa3204c5c869e26b");

        result = service.newTx(tx);
        assertNotNull(result);
        assertTrue(result.isSuccess());
        assertFalse(result.isFailed());

        List<Transaction> list = TxMemoryPool.getInstance().getAll();
        assertNotNull(list);
        assertEquals(list.size(), 1);

        List<Transaction> orphanList = TxMemoryPool.getInstance().getAllOrphan();
        assertNotNull(orphanList);
```

```
        assertEquals(orphanList.size(), 1);

    }

    @Test
    public void newBlock() throws Exception {
    }

    @Test
    public void addBlock() throws Exception {
    }

    @Test
    public void rollbackBlock() throws Exception {
    }

    @Test
    public void getMemoryTxs() throws Exception {

        assertNotNull(service);

        Transaction tx = new TestTransaction();
        tx.setTime(0l);

        assertEquals(tx.getHash().getDigestHex(),
"0020c7f397ae78f2c1d12b3edc916e8112bcac576a98444c4c26034c207c9a7ad281");

        Result result = service.newTx(tx);
        assertNotNull(result);
        assertTrue(result.isSuccess());
        assertFalse(result.isFailed());

        List<Transaction> memoryTxs = service.getMemoryTxs();
        assertNotNull(memoryTxs);

        assertEquals(1, memoryTxs.size());

        tx = memoryTxs.get(0);
        assertNotNull(tx);
        assertEquals(tx.getHash().getDigestHex(),
"0020c7f397ae78f2c1d12b3edc916e8112bcac576a98444c4c26034c207c9a7ad281");
    }
```

```
    @Test
    public void reset() throws Exception {
//        service.reset();
    }

}

80:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\task\BlockProcessTaskTest.java
 *
 */

package io.nuls.consensus.poc.task;

import io.nuls.consensus.poc.BaseTest;
import io.nuls.consensus.poc.customer.ConsensusDownloadServiceImpl;
import io.nuls.consensus.poc.manager.ChainManager;
import io.nuls.consensus.poc.process.BlockProcess;
import io.nuls.consensus.poc.constant.ConsensusStatus;
import io.nuls.consensus.poc.context.ConsensusStatusContext;
import io.nuls.consensus.poc.provider.OrphanBlockProvider;
import io.nuls.kernel.lite.core.SpringLiteContext;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/8.
 */
public class BlockProcessTaskTest extends BaseTest {

    private BlockProcessTask blockProcessTask;
    private ConsensusDownloadServiceImpl downloadService;

    @Before
    public void init() {
        ChainManager chainManager = new ChainManager();
        OrphanBlockProvider orphanBlockProvider = new OrphanBlockProvider();

        BlockProcess blockProcess = new BlockProcess(chainManager, orphanBlockProvider);
```

```
        blockProcessTask = new BlockProcessTask(blockProcess);

        downloadService = SpringLiteContext.getBean(ConsensusDownloadServiceImpl.class);
    }

    @Test
    public void testRun() {
        assertNotNull(blockProcessTask);

        if(downloadService.isDownloadSuccess().isSuccess()) {
            downloadService.setDownloadSuccess(false);
        }

        ConsensusStatusContext.setConsensusStatus(ConsensusStatus.WAIT_RUNNING);

        blockProcessTask.run();

        assert(!ConsensusStatusContext.isRunning());

        downloadService.setDownloadSuccess(true);

        blockProcessTask.run();

        assert(ConsensusStatusContext.isRunning());
    }
}

81:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
base\src\test\java\io\nuls\consensus\poc\TestTransaction.java
 *
 */

package io.nuls.consensus.poc;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.NulsByteBuffer;

/**
 * Created by ln on 2018/5/5.
```

```
 */
public class TestTransaction extends Transaction {

    public TestTransaction() {
        super(0);
    }

    @Override
    protected TransactionLogicData parseTxData(NulsByteBuffer byteBuffer) throws NulsException
{
        return null;
    }

    @Override
    public String getInfo(byte[] address) {
        return null;
    }
}
```

82:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\util\ConsensusToolTest.java

```
 *
 */

package io.nuls.consensus.poc.util;

import io.nuls.consensus.poc.BaseTest;
import io.nuls.kernel.model.Block;
import io.nuls.protocol.model.SmallBlock;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/8.
 */
public class ConsensusToolTest extends BaseTest {

    @Test
    public void testGetSmallBlock() {

        Block block = createBlock();
```

```
        SmallBlock smallBlock = ConsensusTool.getSmallBlock(block);
        assertNotNull(smallBlock);

        assertEquals(smallBlock.getHeader(), block.getHeader());
        assertEquals(smallBlock.getSubTxList().size() , 0);
        assertEquals(smallBlock.getTxHashList().get(0), block.getTxs().get(0).getHash());
    }
}
```

83:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-base\src\test\java\io\nuls\consensus\poc\validator\CreateAgentTxValidatorTest.java

```
 *
 */

package io.nuls.consensus.poc.validator;

import io.nuls.consensus.poc.BaseTest;
import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.tx.validator.CreateAgentTxValidator;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;
import org.junit.Test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

/**
 * Created by ln on 2018/5/10.
 */
public class CreateAgentTxValidatorTest extends BaseTest {

    @Test
    public void test() {
        CreateAgentTransaction tx = new CreateAgentTransaction();

        CreateAgentTxValidator validator = new CreateAgentTxValidator();

        ValidateResult result = validator.validate(tx);
```

```
        assertFalse(result.isSuccess());

        Agent agent = new Agent();

        tx.setTxData(agent);

        result = validator.validate(tx);
        assertFalse(result.isSuccess());

        byte[] address = AddressTool.getAddress(ecKey.getPubKey());
        byte[] address1 = AddressTool.getAddress(new ECKey().getPubKey());
        byte[] address2 = AddressTool.getAddress(new ECKey().getPubKey());
        agent.setRewardAddress(address);
        agent.setPackingAddress(address1);
        agent.setAgentAddress(address2);

        result = validator.validate(tx);
        assertFalse(result.isSuccess());

        agent.setDeposit(PocConsensusProtocolConstant.AGENT_DEPOSIT_LOWER_LIMIT);

        tx.setTime(System.currentTimeMillis());

        result = validator.validate(tx);
        assertTrue(result.isSuccess());
    }
}

84:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\constant\PocConsensusErrorCode.java
 */

package io.nuls.consensus.poc.protocol.constant;

import io.nuls.kernel.constant.ErrorCode;
import io.nuls.kernel.constant.KernelErrorCode;

/**
 * @author: Niels Wang
 */
public interface PocConsensusErrorCode extends KernelErrorCode {
```

```
    ErrorCode TIME_OUT = ErrorCode.init("70001");
    ErrorCode DEPOSIT_ERROR = ErrorCode.init("70002");
    ErrorCode DEPOSIT_NOT_ENOUGH = ErrorCode.init("70003");
    ErrorCode CONSENSUS_EXCEPTION = ErrorCode.init("70004");
    ErrorCode COMMISSION_RATE_OUT_OF_RANGE = ErrorCode.init("70005");
    ErrorCode LACK_OF_CREDIT = ErrorCode.init("70006");
    ErrorCode DEPOSIT_OVER_COUNT = ErrorCode.init("70007");
    ErrorCode DEPOSIT_TOO_MUCH = ErrorCode.init("70008");
    ErrorCode AGENT_STOPPED = ErrorCode.init("70009");

    ErrorCode DEPOSIT_WAS_CANCELED = ErrorCode.init("70010");
    ErrorCode DEPOSIT_NEVER_CANCELED = ErrorCode.init("70011");
    ErrorCode UPDATE_DEPOSIT_FAILED = ErrorCode.init("70012");

    ErrorCode UPDATE_AGENT_FAILED = ErrorCode.init("70014");
    ErrorCode LOCK_TIME_NOT_REACHED = ErrorCode.init("70015");

    ErrorCode AGENT_NOT_EXIST = ErrorCode.init("70016");
    ErrorCode AGENT_EXIST = ErrorCode.init("70017");
    ErrorCode AGENT_PUNISHED = ErrorCode.init("70018");
    ErrorCode BIFURCATION = ErrorCode.init("70019");
    ErrorCode YELLOW_PUNISH_TX_WRONG = ErrorCode.init("70020");
    ErrorCode ADDRESS_IS_CONSENSUS_SEED = ErrorCode.init("70021");
    ErrorCode TRANSACTIONS_NEVER_DOUBLE_SPEND = ErrorCode.init("70022");
    ErrorCode RED_CARD_VERIFICATION_FAILED = ErrorCode.init("70023");
    ErrorCode AGENT_PACKING_EXIST = ErrorCode.init("70024");
    ErrorCode AGENTADDR_AND_PACKING_SAME = ErrorCode.init("70025");
    ErrorCode REWARDADDR_PACKING_SAME = ErrorCode.init("70026");


}
```

85:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\constant\PocConsensusProtocolConstant.java

```
 *
 */
package io.nuls.consensus.poc.protocol.constant;

import io.nuls.kernel.model.Na;

/**
```

```
 * @author Niels
 */
public interface PocConsensusProtocolConstant {

    int ALIVE_MIN_NODE_COUNT = 2;

    String CFG_CONSENSUS_SECTION = "consensus";
    String PROPERTY_PARTAKE_PACKING = "partake.packing";
    String PROPERTY_SEED_NODES = "seed.nodes";
    String SEED_NODES_DELIMITER = ",";

    String GENESIS_BLOCK_FILE = "block/genesis-block.json";

    short NOTICE_PACKED_BLOCK = 22;
    short NOTICE_REGISTER_AGENT = 23;
    short NOTICE_ASSEMBLED_BLOCK = 24;
    short NOTICE_JOIN_CONSENSUS = 25;
    short NOTICE_EXIT_CONSENSUS = 26;
    short NOTICE_CANCEL_CONSENSUS = 27;
    int COINBASE_UNLOCK_HEIGHT = 1000;


    /**
     * Set temporarily as a fixed value,unit:nuls
     */
    int BLOCK_COUNT_OF_YEAR = 3153600;
    int BLOCK_COUNT_OF_DAY = 8640;
    /**
     * value = 5000000/3154600
     */
    Na BLOCK_REWARD = Na.valueOf(158548960);

    /**
     * default:2M
     */
    long MAX_BLOCK_SIZE = 2 << 21;

    Na AGENT_DEPOSIT_LOWER_LIMIT = Na.parseNuls(20000);
    Na AGENT_DEPOSIT_UPPER_LIMIT = Na.parseNuls(200000);
    Na ENTRUSTER_DEPOSIT_LOWER_LIMIT = Na.parseNuls(2000);
    /**
     * Maximum acceptable number of delegate
```

```
 */
int MAX_ACCEPT_NUM_OF_DEPOSIT = 250;
int MAX_AGENT_COUNT_OF_ADRRESS = 1;

Na SUM_OF_DEPOSIT_OF_AGENT_LOWER_LIMIT = Na.parseNuls(200000);
Na SUM_OF_DEPOSIT_OF_AGENT_UPPER_LIMIT = Na.parseNuls(500000);
/**
 * Annual inflation
 */
Na ANNUAL_INFLATION = Na.parseNuls(5000000);
/**
 * unit: %
 */
double AGENT_FORCED_EXITED_RATE = 70;
/**
 * commission rate,UNIT:%
 */
double MAX_COMMISSION_RATE = 100;
double MIN_COMMISSION_RATE = 10;
/**
 * unit:day
 */
long RED_PUNISH_DEPOSIT_LOCKED_TIME = 90;
long YELLOW_PUNISH_DEPOSIT_LOCKED_TIME = 3;
long STOP_AGENT_DEPOSIT_LOCKED_TIME = 3;

/**
 * credit parameters
 */
/**
 * unit:round of consensus
 */
int RANGE_OF_CAPACITY_COEFFICIENT = 100;
/**
 * Penalty coefficient,greater than 4.
 */
int CREDIT_MAGIC_NUM = 100;

/**
 * lock of lockTime,(max of int48)(281474976710655L)
 */
long LOCK_OF_LOCK_TIME = -1L ;
```

```
}

86:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\constant\PunishReasonEnum.java
 */
package io.nuls.consensus.poc.protocol.constant;


import io.nuls.kernel.i18n.I18nUtils;

/**
 * @author Niels
 */
public enum PunishReasonEnum {
    /**
     * Bifurcate block chain
     */
    BIFURCATION((byte) 1, "69980"),

    /**
     * double spend
     */
    DOUBLE_SPEND((byte) 2, "69981"),

    /**
     * x
     * Continuous x round yellow card.
     */
    TOO_MUCH_YELLOW_PUNISH((byte) 3, "69982"),;
    private final byte code;
    private final String msgCode;

    private PunishReasonEnum(byte code, String msgCode) {
        this.code = code;
        this.msgCode = msgCode;
    }

    public String getMessage() {
        return I18nUtils.get(this.msgCode);
    }

    public byte getCode() {
```

```
        return code;
    }

    public static PunishReasonEnum getEnum(int code) {
        switch (code) {
            case 1:
                return PunishReasonEnum.BIFURCATION;
            case 2:
                return PunishReasonEnum.DOUBLE_SPEND;
            case 3:
                return PunishReasonEnum.TOO_MUCH_YELLOW_PUNISH;
            default:
                return PunishReasonEnum.TOO_MUCH_YELLOW_PUNISH;
        }

    }
}
```

87:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\constant\PunishType.java

```
 *
 */

package io.nuls.consensus.poc.protocol.constant;

/**
 * @author Niels
 */
public enum PunishType {

    YELLOW(0), RED(1);

    private final byte code;

    PunishType(int code) {
        this.code = (byte) code;
    }

    public byte getCode() {
        return code;
    }
}
```

88:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\entity\Agent.java
 *
 */
```
package io.nuls.consensus.poc.protocol.entity;

import io.nuls.consensus.poc.protocol.constant.PocConsensusProtocolConstant;
import io.nuls.consensus.poc.protocol.util.PoConvertUtil;
import io.nuls.core.tools.calc.LongUtils;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.protostuff.Tag;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

/**
 * @author Niels
 */
public class Agent extends TransactionLogicData {

    private byte[] agentAddress;

    private byte[] packingAddress;

    private byte[] rewardAddress;

    private Na deposit;

    private double commissionRate;

    private transient long time;
    private transient long blockHeight = -1L;
```

```
private transient long delHeight = -1L;
/**
 * 0: unconsensus, 1: consensus
 */
private transient int status;
private transient double creditVal;
private transient long totalDeposit;
private transient NulsDigestData txHash;
private transient int memberCount;

@Override
public int size() {
    int size = 0;
    size += SerializeUtils.sizeOfInt64();  // deposit.getValue()
    size += this.agentAddress.length;
    size += this.rewardAddress.length;
    size += this.packingAddress.length;
    size += SerializeUtils.sizeOfDouble(this.commissionRate);
    return size;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeInt64(deposit.getValue());
    stream.write(agentAddress);
    stream.write(packingAddress);
    stream.write(rewardAddress);
    stream.writeDouble(this.commissionRate);
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.deposit = Na.valueOf(byteBuffer.readInt64());
    this.agentAddress = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.packingAddress = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.rewardAddress = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.commissionRate = byteBuffer.readDouble();
}

public Na getDeposit() {
    return deposit;
}
```

```
public void setDeposit(Na deposit) {
    this.deposit = deposit;
}

public byte[] getPackingAddress() {
    return packingAddress;
}

public void setPackingAddress(byte[] packingAddress) {
    this.packingAddress = packingAddress;
}

public int getStatus() {
    return status;
}

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

public double getCommissionRate() {
    return commissionRate;
}

public void setCommissionRate(double commissionRate) {
    this.commissionRate = commissionRate;
}

public long getBlockHeight() {
    return blockHeight;
}

public void setBlockHeight(long blockHeight) {
    this.blockHeight = blockHeight;
}

public void setCreditVal(double creditVal) {
    this.creditVal = creditVal;
}

public double getCreditVal() {
```

```
    return creditVal;
}

public long getTotalDeposit() {
    return totalDeposit;
}

public void setTotalDeposit(long totalDeposit) {
    this.totalDeposit = totalDeposit;
}

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

public NulsDigestData getTxHash() {
    return txHash;
}

public long getTime() {
    return time;
}

public void setTime(long time) {
    this.time = time;
}

public long getDelHeight() {
    return delHeight;
}

public void setDelHeight(long delHeight) {
    this.delHeight = delHeight;
}

public byte[] getAgentAddress() {
    return agentAddress;
}

public void setAgentAddress(byte[] agentAddress) {
    this.agentAddress = agentAddress;
}
```

```
    public byte[] getRewardAddress() {
        return rewardAddress;
    }

    public void setRewardAddress(byte[] rewardAddress) {
        this.rewardAddress = rewardAddress;
    }

    public int getMemberCount() {
        return memberCount;
    }

    public void setMemberCount(int memberCount) {
        this.memberCount = memberCount;
    }

    public long getAvailableDepositAmount() {
        return
LongUtils.sub(PocConsensusProtocolConstant.SUM_OF_DEPOSIT_OF_AGENT_UPPER_LIMIT.
getValue(), this.getTotalDeposit());
    }

    public boolean canDeposit() {
        return getAvailableDepositAmount() >=
PocConsensusProtocolConstant.ENTRUSTER_DEPOSIT_LOWER_LIMIT.getValue();
    }

    @Override
    public Agent clone() throws CloneNotSupportedException {
//        Agent agent = new Agent();
//
//        agent.setAgentAddress(getAgentAddress());
//        agent.setAgentName(getAgentName());
//        agent.setBlockHeight(getBlockHeight());
//        agent.setCommissionRate(getCommissionRate());
//        agent.setCreditVal(getCreditVal());
//        agent.setDelHeight(getDelHeight());
//        agent.setDeposit(getDeposit());
//        agent.setIntroduction(getIntroduction());
//        agent.setStatus(getStatus());
//        agent.setTime(getTime());
```

```
//        agent.setPackingAddress(getPackingAddress());
//        agent.setTotalDeposit(getTotalDeposit());
//        agent.setTxHash(getTxHash());
//
//        return agent;

        return (Agent) super.clone();
    }

    @Override
    public Set<byte[]> getAddresses() {
        Set<byte[]> addressSet = new HashSet<>();
        addressSet.add(this.agentAddress);
        return addressSet;
    }
}
```

89:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\entity\CancelDeposit.java

```
 */

package io.nuls.consensus.poc.protocol.entity;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

/**
 * @author: Niels Wang
 */
public class CancelDeposit extends TransactionLogicData {

    private byte[] address;

    private NulsDigestData joinTxHash;
```

```
@Override
public Set<byte[]> getAddresses() {
    Set<byte[]> addressSet = new HashSet<>();
    if (null != address) {
        addressSet.add(this.address);
    }
    return addressSet;
}

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

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

public NulsDigestData getJoinTxHash() {
    return joinTxHash;
}

public void setJoinTxHash(NulsDigestData joinTxHash) {
    this.joinTxHash = joinTxHash;
}

/**
 * serialize important field
 */
@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeNulsData(this.joinTxHash);

}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.joinTxHash = byteBuffer.readHash();
}

@Override
public int size() {
    return this.joinTxHash.size();
```

```
    }
}
```

90:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\entity\Deposit.java

```
 *
 */
package io.nuls.consensus.poc.protocol.entity;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.*;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

/**
 * @author Niels
 */
public class Deposit extends TransactionLogicData {


    private Na deposit;

    private NulsDigestData agentHash;

    private byte[] address;

    private transient long time;
    private transient int status;
    private transient NulsDigestData txHash;
    private transient long blockHeight = -1L;
    private transient long delHeight = -1L;

    /**
     * serialize important field
     */
    @Override
```

```
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeInt64(deposit.getValue());
    stream.write(address);
    stream.writeNulsData(agentHash);

}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.deposit = Na.valueOf(byteBuffer.readInt64());
    this.address = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.agentHash = byteBuffer.readHash();
}

@Override
public int size() {
    int size = 0;
    size += SerializeUtils.sizeOfInt64(); // deposit.getValue()
    size += Address.ADDRESS_LENGTH;
    size += this.agentHash.size();
    return size;
}

public Na getDeposit() {
    return deposit;
}

public void setDeposit(Na deposit) {
    this.deposit = deposit;
}

public NulsDigestData getAgentHash() {
    return agentHash;
}

public void setAgentHash(NulsDigestData agentHash) {
    this.agentHash = agentHash;
}

public long getTime() {
    return time;
}
```

```
public void setTime(long time) {
    this.time = time;
}

public int getStatus() {
    return status;
}

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

public NulsDigestData getTxHash() {
    return txHash;
}

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

public long getBlockHeight() {
    return blockHeight;
}

public void setBlockHeight(long blockHeight) {
    this.blockHeight = blockHeight;
}

public long getDelHeight() {
    return delHeight;
}

public void setDelHeight(long delHeight) {
    this.delHeight = delHeight;
}

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

public void setAddress(byte[] address) {
```

```
        this.address = address;
    }

    @Override
    public Set<byte[]> getAddresses() {
        Set<byte[]> addressSet = new HashSet<>();
        addressSet.add(this.address);
        return addressSet;
    }

    @Override
    public Deposit clone() throws CloneNotSupportedException {
        return (Deposit) super.clone();
    }

}

91:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
protocol\src\main\java\io\nuls\consensus\poc\protocol\entity\RedPunishData.java
 */
package io.nuls.consensus.poc.protocol.entity;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.*;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

/**
 * @author Niels
 */
public class RedPunishData extends TransactionLogicData {
    private byte[] address;
    private byte reasonCode;
    private byte[] evidence;

    public RedPunishData() {
    }
```

```
@Override
public int size() {
    int size = 0;
    size += address.length;
    size += 1;
    size += SerializeUtils.sizeOfBytes(evidence);
    return size;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.write(address);
    stream.write(reasonCode);
    stream.writeBytesWithLength(evidence);

}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.address = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.reasonCode = byteBuffer.readByte();
    this.evidence = byteBuffer.readByLengthByte();
}

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

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

public byte getReasonCode() {
    return reasonCode;
}

public void setReasonCode(byte reasonCode) {
    this.reasonCode = reasonCode;
}

public byte[] getEvidence() {
    return evidence;
```

```
    }

    public void setEvidence(byte[] evidence) {
        this.evidence = evidence;
    }

    @Override
    public Set<byte[]> getAddresses() {
        Set<byte[]> set = new HashSet<>();
        set.add(address);
        return set;
    }
}

92:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
protocol\src\main\java\io\nuls\consensus\poc\protocol\entity\StopAgent.java
 */

package io.nuls.consensus.poc.protocol.entity;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

/**
 * @author: Niels Wang
 */
public class StopAgent extends TransactionLogicData {

    private byte[] address;

    private NulsDigestData createTxHash;
    /**
     * serialize important field
     */
    @Override
```

```
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeNulsData(this.createTxHash);

    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.createTxHash = byteBuffer.readHash();
    }

    @Override
    public int size() {
        return this.createTxHash.size();
    }
    @Override
    public Set<byte[]> getAddresses() {
        Set<byte[]> addressSet = new HashSet<>();
        if(null!=address){
            addressSet.add(this.address);
        }
        return addressSet;
    }

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

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

    public NulsDigestData getCreateTxHash() {
        return createTxHash;
    }

    public void setCreateTxHash(NulsDigestData createTxHash) {
        this.createTxHash = createTxHash;
    }
}
```

93:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\entity\YellowPunishData.java

```
 */
package io.nuls.consensus.poc.protocol.entity;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import javax.lang.model.element.VariableElement;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * @author Niels
 */
public class YellowPunishData extends TransactionLogicData {
    private List<byte[]> addressList = new ArrayList<>();

    public YellowPunishData() {
    }

    public List<byte[]> getAddressList() {
        return addressList;
    }

    public void setAddressList(List<byte[]> addressList) {
        this.addressList = addressList;
    }

    @Override
    public Set<byte[]> getAddresses() {
        return new HashSet<>(addressList);
    }

    /**
     * serialize important field
```

```
     */
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeVarInt(addressList.size());
        for (byte[] address : addressList) {
            stream.write(address);
        }

    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        int count = (int) byteBuffer.readVarInt();
        addressList.clear();
        for (int i = 0; i < count; i++) {
            addressList.add(byteBuffer.readBytes(Address.ADDRESS_LENGTH));
        }

    }

    @Override
    public int size() {
        int size = SerializeUtils.sizeOfVarInt(addressList.size());
        for (byte[] address : addressList) {
            size += Address.ADDRESS_LENGTH;
        }
        return size;
    }
}
```

94:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\tx\CancelDepositTransaction.java

```
 *
 */
package io.nuls.consensus.poc.protocol.tx;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.entity.CancelDeposit;
import io.nuls.core.tools.calc.DoubleUtils;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.CoinData;
```

```
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.ledger.service.LedgerService;

/**
 * @author Niels
 */
public class CancelDepositTransaction extends Transaction<CancelDeposit> {

    public CancelDepositTransaction() {
        super(ConsensusConstant.TX_TYPE_CANCEL_DEPOSIT);
    }

    public CancelDepositTransaction(CoinData coinData) {
        super(ConsensusConstant.TX_TYPE_CANCEL_DEPOSIT);
    }

    @Override
    protected CancelDeposit parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new CancelDeposit());
    }

    @Override
    public String getInfo(byte[] address) {
        if (null != this.txData) {
            LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);
            DepositTransaction tx = (DepositTransaction)
ledgerService.getTx(this.txData.getJoinTxHash());
            if (null != tx) {
                return "unlock " + tx.getTxData().getDeposit().toCoinString();
            }
        }
        return "--";
    }

    @Override
    public boolean isUnlockTx() {
        return true;
    }
}

95:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
```

protocol\src\main\java\io\nuls\consensus\poc\protocol\tx\CreateAgentTransaction.java

```
 *
 */
package io.nuls.consensus.poc.protocol.tx;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;

/**
 * @author Niels
 */
public class CreateAgentTransaction extends Transaction<Agent> {

    public CreateAgentTransaction() {
        super(ConsensusConstant.TX_TYPE_REGISTER_AGENT);
    }

    @Override
    public CreateAgentTransaction clone() {
        CreateAgentTransaction tx = new CreateAgentTransaction();
        try {
            tx.parse(serialize(), 0);
        } catch (Exception e) {
            throw new NulsRuntimeException(e);
        }
        tx.setBlockHeight(blockHeight);
        tx.setStatus(status);
        tx.setHash(hash);
        tx.setSize(size);

        Agent agent = tx.getTxData();
        agent.setBlockHeight(txData.getBlockHeight());
        agent.setDelHeight(txData.getDelHeight());
        agent.setTime(txData.getTime());
        agent.setTxHash(txData.getTxHash());
        agent.setStatus(txData.getStatus());
        agent.setTotalDeposit(txData.getTotalDeposit());
        agent.setCreditVal(txData.getCreditVal());
```

```
        return tx;
    }

    @Override
    protected Agent parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new Agent());
    }

    @Override
    public String getInfo(byte[] address) {
        return "lock "+getTxData().getDeposit().toText();
    }
}
```

96:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\tx\DepositTransaction.java

```
 *
 */
package io.nuls.consensus.poc.protocol.tx;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;

/**
 * @author Niels
 */
public class DepositTransaction extends Transaction<Deposit> {

    public DepositTransaction() {
        super(ConsensusConstant.TX_TYPE_JOIN_CONSENSUS);
    }

    @Override
    protected Deposit parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new Deposit());
    }
```

```
    @Override
    public String getInfo(byte[] address) {
        return "lock "+ getTxData().getDeposit().toText();
    }

    @Override
    public DepositTransaction clone() {
        DepositTransaction tx = new DepositTransaction();
        try {
            tx.parse(serialize(), 0);
        } catch (Exception e) {
            throw new NulsRuntimeException(e);
        }
        tx.setBlockHeight(blockHeight);
        tx.setStatus(status);
        tx.setHash(hash);
        tx.setSize(size);

        Deposit deposit = tx.getTxData();
        deposit.setBlockHeight(txData.getBlockHeight());
        deposit.setDelHeight(txData.getDelHeight());
        deposit.setTime(txData.getTime());
        deposit.setTxHash(txData.getTxHash());
        deposit.setStatus(txData.getStatus());

        return tx;
    }
}

97:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
protocol\src\main\java\io\nuls\consensus\poc\protocol\tx\RedPunishTransaction.java
 */
package io.nuls.consensus.poc.protocol.tx;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.entity.RedPunishData;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;

/**
 * @author Niels
```

```
 */
public class RedPunishTransaction extends Transaction<RedPunishData> {
    public RedPunishTransaction() {
        super(ConsensusConstant.TX_TYPE_RED_PUNISH);
    }

    @Override
    protected RedPunishData parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new RedPunishData());
    }

    @Override
    public String getInfo(byte[] address) {
        return "--";
    }

    @Override
    public boolean isSystemTx() {
        return true;
    }

    @Override
    public boolean needVerifySignature() {
        return false;
    }
}
```

98:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\tx\StopAgentTransaction.java

```
 *
 */
package io.nuls.consensus.poc.protocol.tx;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.entity.StopAgent;
import io.nuls.core.tools.calc.DoubleUtils;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.CoinData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.ledger.service.LedgerService;
```

```
/**
 * @author Niels
 */
public class StopAgentTransaction extends Transaction<StopAgent> {

    public StopAgentTransaction() {
        super(ConsensusConstant.TX_TYPE_STOP_AGENT);
    }

    public StopAgentTransaction(CoinData coinData) throws NulsException {
        super(ConsensusConstant.TX_TYPE_STOP_AGENT);
    }

    @Override
    protected StopAgent parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new StopAgent());
    }

    @Override
    public String getInfo(byte[] address) {
        if (null != this.txData) {
            LedgerService ledgerService = NulsContext.getServiceBean(LedgerService.class);
            CreateAgentTransaction tx = (CreateAgentTransaction)
ledgerService.getTx(this.txData.getCreateTxHash());
            if (null != tx) {
                return "unlock " + tx.getTxData().getDeposit().toCoinString();
            }
        }
        return "--";
    }

    @Override
    public boolean isUnlockTx() {
        return true;
    }

    @Override
    public boolean needVerifySignature() {
        return false;
    }
}
```

99:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\tx\YellowPunishTransaction.java

```
 */
package io.nuls.consensus.poc.protocol.tx;

import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.consensus.poc.protocol.entity.YellowPunishData;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;

/**
 * @author Niels
 */
public class YellowPunishTransaction extends Transaction<YellowPunishData> {
    public YellowPunishTransaction() {
        super(ConsensusConstant.TX_TYPE_YELLOW_PUNISH);
    }

    @Override
    protected YellowPunishData parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new YellowPunishData());
    }

    @Override
    public String getInfo(byte[] address) {
        return "--";
    }

    @Override
    public boolean isSystemTx() {
        return true;
    }

    @Override
    public boolean needVerifySignature() {
        return false;
    }
}
```

100:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-

protocol\src\main\java\io\nuls\consensus\poc\protocol\util\AgentComparator.java
 */

```
package io.nuls.consensus.poc.protocol.util;

import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;

import java.util.Comparator;

/**
 * @author: Niels Wang
 */
public class AgentComparator implements Comparator<Agent> {

    @Override
    public int compare(Agent o1, Agent o2) {
        if (o1.getBlockHeight() == o2.getBlockHeight()) {
            return (int) (o1.getTime() - o2.getTime());
        }
        return (int) (o1.getBlockHeight() - o2.getBlockHeight());
    }
}
```

101:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\util\DepositComparator.java
 */

```
package io.nuls.consensus.poc.protocol.util;

import io.nuls.consensus.poc.protocol.entity.Deposit;

import java.util.Comparator;

/**
 * @author: Niels Wang
 */
public class DepositComparator implements Comparator<Deposit> {

    @Override
    public int compare(Deposit o1, Deposit o2) {
        if (o1.getBlockHeight() == o2.getBlockHeight()) {
```

```
            return (int) (o1.getTime() - o2.getTime());
        }
        return (int) (o1.getBlockHeight() - o2.getBlockHeight());
    }
}
```

102:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\main\java\io\nuls\consensus\poc\protocol\util\PoConvertUtil.java

```
 *
 */

package io.nuls.consensus.poc.protocol.util;

import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.kernel.model.NulsDigestData;

/**
 * @author ln
 */
public final class PoConvertUtil {

    public static Agent poToAgent(AgentPo agentPo) {
        if (agentPo == null) {
            return null;
        }
        Agent agent = new Agent();
        agent.setAgentAddress(agentPo.getAgentAddress());
        agent.setBlockHeight(agentPo.getBlockHeight());
        agent.setCommissionRate(agentPo.getCommissionRate());
        agent.setDeposit(agentPo.getDeposit());
        agent.setPackingAddress(agentPo.getPackingAddress());
        agent.setRewardAddress(agentPo.getRewardAddress());
        agent.setTxHash(agentPo.getHash());
        agent.setTime(agentPo.getTime());
        agent.setDelHeight(agentPo.getDelHeight());
        return agent;
    }
```

```
public static AgentPo agentToPo(Agent agent) {
    if (agent == null) {
        return null;
    }
    AgentPo agentPo = new AgentPo();
    agentPo.setAgentAddress(agent.getAgentAddress());
    agentPo.setBlockHeight(agent.getBlockHeight());
    agentPo.setCommissionRate(agent.getCommissionRate());
    agentPo.setDeposit(agent.getDeposit());
    agentPo.setPackingAddress(agent.getPackingAddress());
    agentPo.setRewardAddress(agent.getRewardAddress());
    agentPo.setHash(agent.getTxHash());
    agentPo.setTime(agent.getTime());
    return agentPo;
}


public static Deposit poToDeposit(DepositPo po) {
    Deposit deposit = new Deposit();
    deposit.setDeposit(po.getDeposit());
    deposit.setAgentHash(po.getAgentHash());
    deposit.setTime(po.getTime());
    deposit.setDelHeight(po.getDelHeight());
    deposit.setBlockHeight(po.getBlockHeight());
    deposit.setAddress(po.getAddress());
    deposit.setTxHash(po.getTxHash());
    return deposit;
}

public static DepositPo depositToPo(Deposit deposit) {
    DepositPo po = new DepositPo();
    po.setTxHash(deposit.getTxHash());
    po.setAddress(deposit.getAddress());
    po.setAgentHash(deposit.getAgentHash());
    po.setBlockHeight(deposit.getBlockHeight());
    po.setDelHeight(deposit.getDelHeight());
    po.setDeposit(deposit.getDeposit());
    po.setTime(deposit.getTime());
    return po;
}

public static String getAgentId(NulsDigestData hash) {
```

```
        String hashHex = hash.getDigestHex();
        return hashHex.substring(hashHex.length() - 8).toUpperCase();
    }
}
```

103:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\test\java\io\nuls\consensus\poc\protocol\BaseTest.java

```
 *
 */

package io.nuls.consensus.poc.protocol;

import io.nuls.core.tools.crypto.ECKey;
import io.nuls.kernel.MicroKernelBootstrap;
import org.junit.BeforeClass;

/**
 * Created by ln on 2018/5/10.
 */
public class BaseTest {

    protected ECKey ecKey = new ECKey();

    @BeforeClass
    public static void initMicroKernel() {
        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();
    }
}
```

104:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\test\java\io\nuls\consensus\poc\protocol\tx\CancelDepositTransactionTest.java

```
 */

package io.nuls.consensus.poc.protocol.tx;

import io.nuls.consensus.poc.protocol.entity.CancelDeposit;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.AddressTool;
import org.junit.Test;
```

```
import static org.junit.Assert.assertTrue;

/**
 * @author: Niels Wang
 */
public class CancelDepositTransactionTest extends TxSerializeTest {


    @Test
    public void testSerialize() {
        CancelDepositTransaction tx = new CancelDepositTransaction();
        this.setCommonFields(tx);
        CancelDeposit txData = new CancelDeposit();
        txData.setAddress(AddressTool.getAddress(ecKey1.getPubKey()));
        txData.setJoinTxHash(NulsDigestData.calcDigestData("1234567890".getBytes()));
        this.signTransaction(tx, ecKey1);
        try {
            this.testTxSerialize(tx, new CancelDepositTransaction());
        } catch (Exception e) {
            assertTrue(false);
        }
    }

}

105:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
protocol\src\test\java\io\nuls\consensus\poc\protocol\tx\TxSerializeTest.java
 */

package io.nuls.consensus.poc.protocol.tx;

import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.NulsSignData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.script.SignatureUtil;


import java.io.IOException;
import java.security.SignatureException;
```

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

import static junit.framework.TestCase.assertEquals;
import static junit.framework.TestCase.assertTrue;

/**
 * @author: Niels Wang
 */
public class TxSerializeTest {

    protected ECKey ecKey1 = new ECKey();
    protected ECKey ecKey2 = new ECKey();
    protected ECKey ecKey3 = new ECKey();

    protected void setCommonFields(Transaction tx) {
        tx.setTime(System.currentTimeMillis());
        tx.setBlockHeight(1);
        tx.setRemark("for test".getBytes());
    }

    protected void signTransaction(Transaction tx, ECKey ecKey) {
        NulsDigestData hash = null;
        try {
            hash = NulsDigestData.calcDigestData(tx.serializeForHash());
        } catch (IOException e) {
            Log.error(e);
        }
        tx.setHash(hash);

        List<ECKey> keys = new ArrayList<>();
        keys.add(ecKey);

        try {
            SignatureUtil.createTransactionSignture(tx, null, keys);
        } catch (Exception e) {
            Log.error(e);
        }
    }

    public void testTxSerialize(Transaction tx, Transaction nullTx) throws NulsException,
```

```
IOException {
        byte[] bytes = tx.serialize();

        assertEquals(bytes.length, tx.size());

        nullTx.parse(bytes, 0);

        assertEquals(bytes.length, nullTx.size());

        assertTrue(Arrays.equals(bytes, nullTx.serialize()));

    }
}
```

106:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-protocol\src\test\java\io\nuls\consensus\poc\protocol\util\PoConvertUtilTest.java

```
 */

package io.nuls.consensus.poc.protocol.util;

import io.nuls.kernel.model.NulsDigestData;
import org.junit.Test;

import static org.junit.Assert.*;

/**
 * @author: Niels Wang
 */
public class PoConvertUtilTest {

    @Test
    public void getAgentId() {
        NulsDigestData hash = NulsDigestData.calcDigestData("123123".getBytes());
        System.out.println(PoConvertUtil.getAgentId(hash));
        System.out.println(hash);
        assertTrue(true);
    }
}
```

107:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\CreateAgentProcessor.java

```
 */
```

```
package io.nuls.consensus.poc.rpc.cmd;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.RestFulUtils;

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

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<agentAddress>   agent owner address   -required")
                .newLine("\t<packingAddress>    packing address    -required")
                .newLine("\t<commissionRate>    commission rate (10~100), you can have up to 2 valid
digits after the decimal point  -required")
                .newLine("\t<deposit>   amount you want to deposit, you can have up to 8 valid digits
after the decimal point -required")
                .newLine("\t[rewardAddress]  Billing address    -not required");
        return bulider.toString();
    }
```

```
    @Override
    public String getCommandDescription() {
        return "createagent <agentAddress> <packingAddress> <commissionRate> <deposit>
[rewardAddress] --create a agent";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length < 5 || length > 6){
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.validAddressSimple(args[1]) || !StringUtils.validAddressSimple(args[2]) ||
StringUtils.isBlank(args[3])
                || StringUtils.isBlank(args[4])){
            return false;
        }
        if(!StringUtils.isNumberGtZeroLimitTwo(args[3])){
            return false;
        }
        if(!StringUtils.isNuls(args[4])){
            return false;
        }
        if(length == 6 && !StringUtils.validAddressSimple(args[5])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        RpcClientResult res = CommandHelper.getPassword(address, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("agentAddress", address);
```

```
        parameters.put("packingAddress", args[2]);
        parameters.put("commissionRate", Double.valueOf(args[3]));
        Long deposit = null;
        try {
            Na na = Na.parseNuls(args[4]);
            deposit = na.getValue();
        } catch (Exception e) {
            return CommandResult.getFailed("Parameter deposit error");
        }
        parameters.put("deposit", deposit);
        parameters.put("password", password);
        if(args.length == 6){
            parameters.put("rewardAddress", args[5]);
        }
        RpcClientResult result = restFul.post("/consensus/agent",parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataTransformValue(result));
    }
}
```

108:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\CreateMultiAgentProcessor.java

```
 */
package io.nuls.consensus.poc.rpc.cmd;

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

public class CreateMultiAgentProcessor implements CommandProcessor {
    private RestFulUtils restFul = RestFulUtils.getInstance();
    @Override
```

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<agentAddress>   agent owner address   -required")
                .newLine("\t<packingAddress>    packing address    -required")
                .newLine("\t<signAddress> \tsign address address - Required")
                .newLine("\t<commissionRate>    commission rate (10~100), you can have up to 2 valid digits after the decimal point  -required")
                .newLine("\t<deposit>   amount you want to deposit, you can have up to 8 valid digits after the decimal point -required")
                .newLine("\t[rewardAddress]  Billing address    -not required");
        return bulider.toString();

    }

    @Override
    public String getCommandDescription() {
        return "createMultiAgent <agentAddress> <packingAddress> <signAddress> <commissionRate> <deposit> [rewardAddress] --create a multi agent";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length != 6 && length != 7){
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.validAddressSimple(args[1]) || !StringUtils.validAddressSimple(args[2])|| !StringUtils.validAddressSimple(args[3]) || StringUtils.isBlank(args[4])
                || StringUtils.isBlank(args[5])){
            return false;
        }
        if(!StringUtils.isNumberGtZeroLimitTwo(args[4])){
            return false;
```

```
        }
        if(!StringUtils.isNuls(args[5])){
            return false;
        }
        if(length == 7 && !StringUtils.validAddressSimple(args[5])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String signAddress = args[3];
        RpcClientResult res = CommandHelper.getPassword(signAddress, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("agentAddress", args[1]);
        parameters.put("packingAddress", args[2]);
        parameters.put("signAddress", args[3]);
        parameters.put("commissionRate", Double.valueOf(args[4]));
        Long deposit = null;
        try {
            Na na = Na.parseNuls(args[5]);
            deposit = na.getValue();
        } catch (Exception e) {
            return CommandResult.getFailed("Parameter deposit error");
        }
        parameters.put("deposit", deposit);
        parameters.put("password", password);
        if(args.length == 7){
            parameters.put("rewardAddress", args[6]);
        }
        RpcClientResult result =
restFul.post("/consensus/multiAccount/createMultiAgent",parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataMultiTransformValue(result));
    }
```

```
}

109:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\CreateMultiDepositProcessor.java
 */
package io.nuls.consensus.poc.rpc.cmd;

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

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

/**
 * @author: tag
 */
public class CreateMultiDepositProcessor implements CommandProcessor {


    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<address>   Your own account address -required")
                .newLine("\t<signAddress> \tsign address - Required")
                .newLine("\t<agentHash>   The agent hash you want to deposit  -required")
                .newLine("\t<deposit>   the amount you want to deposit, you can have up to 8 valid digits after the decimal point -required");
```

```
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "createMultiDeposit <address> <signAddress> <agentHash> <deposit> --apply for
multi deposit";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length != 5){
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.validAddressSimple(args[1]) || !StringUtils.validAddressSimple(args[2])  ||
!NulsDigestData.validHash(args[3])
                || StringUtils.isBlank(args[4]) || !StringUtils.isNuls(args[4])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String signAddress = args[2];
        RpcClientResult res = CommandHelper.getPassword(signAddress, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Long amount = Na.parseNuls(args[4]).getValue();
        Map<String, Object> parameters = new HashMap<>(4);
        parameters.put("address", args[1]);
        parameters.put("signAddress", args[2]);
        parameters.put("agentHash", args[3]);
        parameters.put("deposit", amount);
        parameters.put("password", password);
        RpcClientResult result = restFul.post("/consensus/multiAccount/createMultiDeposit",
```

```
parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataMultiTransformValue(result));
    }
}

110:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\CreateMultiStopAgentProcessor.java
 */
package io.nuls.consensus.poc.rpc.cmd;

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

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

/**
 * @author: tag
 */
public class CreateMultiStopAgentProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<address> account address of the agent -required")
                .newLine("\t<signAddress> \tsign address - Required");
```

```
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "stopMultiagent <address> <signAddress>-- stop the agent";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length != 3){
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.validAddressSimple(args[1]) || !StringUtils.validAddressSimple(args[2])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String signAddress = args[2];
        RpcClientResult res = CommandHelper.getPassword(signAddress, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Map<String, Object> parameters = new HashMap<>(2);
        parameters.put("address", args[1]);
        parameters.put("signAddress", args[2]);
        parameters.put("password", password);
        RpcClientResult result = restFul.post("/consensus/multiAccount/agent/stopMultiAgent",
parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataMultiTransformValue(result));
    }
```

```
}

111:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\CreateMultiWithdrawProcessor.java
 */
package io.nuls.consensus.poc.rpc.cmd;

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

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

/**
 * @author: tag
 */
public class CreateMultiWithdrawProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<address>   address -required")
                .newLine("\t<signAddress> \tsign address - Required")
                .newLine("\t<txHash>    your deposit transaction hash  -required");
        return bulider.toString();
    }

    @Override
```

```
    public String getCommandDescription() {
        return "createMultiWithdraw <address> <signAddress> <txHash> -- withdraw the  agent";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length != 4){
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.validAddressSimple(args[1]) || !StringUtils.validAddressSimple(args[2]) ||
!NulsDigestData.validHash(args[3])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String signAddress = args[2];
        RpcClientResult res = CommandHelper.getPassword(signAddress, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Map<String, Object> parameters = new HashMap<>(3);
        parameters.put("address", args[1]);
        parameters.put("address", args[1]);
        parameters.put("signAddress", args[2]);
        parameters.put("txHash", args[3]);
        parameters.put("password", password);
        RpcClientResult result = restFul.post("/consensus/multiAccount/mutilWithdraw", parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataMultiTransformValue(result));
    }
}
```

```
112:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\DepositProcessor.java
 */

package io.nuls.consensus.poc.rpc.cmd;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.RestFulUtils;

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

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<address>   Your own account address -required")
                .newLine("\t<agentHash>   The agent hash you want to deposit  -required")
                .newLine("\t<deposit>   the amount you want to deposit, you can have up to 8 valid digits after the decimal point -required");
        return bulider.toString();
    }
```

```
@Override
public String getCommandDescription() {
    return "deposit <address> <agentHash> <deposit> --apply for deposit";
}

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if(length != 4){
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    if(!StringUtils.validAddressSimple(args[1]) || !NulsDigestData.validHash(args[2])
            || StringUtils.isBlank(args[3]) || !StringUtils.isNuls(args[3])){
        return false;
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    String address = args[1];
    RpcClientResult res = CommandHelper.getPassword(address, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();
    Long amount = Na.parseNuls(args[3]).getValue();
    Map<String, Object> parameters = new HashMap<>(4);
    parameters.put("address", address);
    parameters.put("agentHash", args[2]);
    parameters.put("deposit", amount);
    parameters.put("password", password);
    RpcClientResult result = restFul.post("/consensus/deposit", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
    return CommandResult.getResult(CommandResult.dataTransformValue(result));
}
```

```
}

113:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\GetAgentProcessor.java
 */

package io.nuls.consensus.poc.rpc.cmd;

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

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

/**
 * hash
 * Get a consensus node information According to agent hash
 *
 * @author: Charlie
 */
public class GetAgentProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<agentHash>  the hash of an agent -required");
```

```
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getagent <agentHash>  -- get an agent node information According to agent hash";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if (length != 2) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if (!NulsDigestData.validHash(args[1])) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String agentHash = args[1];
        RpcClientResult result = restFul.get("/consensus/agent/" + agentHash, null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> map = (Map) result.getData();
        map.put("deposit", CommandHelper.naToNuls(map.get("deposit")));
        map.put("totalDeposit", CommandHelper.naToNuls(map.get("totalDeposit")));
        map.put("time", DateUtil.convertDate(new Date((Long) map.get("time"))));
        map.put("status", CommandHelper.consensusExplain((Integer) map.get("status")));
        result.setData(map);
        return CommandResult.getResult(result);
    }
}
```

114:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\GetAgentsProcessor.java

```
 */

package io.nuls.consensus.poc.rpc.cmd;

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

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

/**
 *
 * Get all the agent nodes
 *
 * @author: Charlie
 */
public class GetAgentsProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<pageNumber> pageNumber - Required")
                .newLine("\t<pageSize> pageSize(1~100) - Required");
        return bulider.toString();
    }
```

```
    @Override
    public String getCommandDescription() {
        return "getagents <pageNumber> <pageSize> --get agent list";
    }

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

    @Override
    public CommandResult execute(String[] args) {
        int pageNumber = Integer.parseInt(args[1]);
        int pageSize = Integer.parseInt(args[2]);
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("pageNumber", pageNumber);
        parameters.put("pageSize", pageSize);
        RpcClientResult result = restFul.get("/consensus/agent/list", parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        List<Map<String, Object>> list = (List<Map<String, Object>>) ((Map)
result.getData()).get("list");
        for (Map<String, Object> map : list) {
            map.put("deposit", CommandHelper.naToNuls(map.get("deposit")));
            map.put("totalDeposit", CommandHelper.naToNuls(map.get("totalDeposit")));
            map.put("time", DateUtil.convertDate(new Date((Long) map.get("time"))));
            map.put("status", CommandHelper.consensusExplain((Integer) map.get("status")));
        }
        result.setData(list);
        return CommandResult.getResult(result);
    }
```

```
}

115:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\GetConsensusProcessor.java
 */

package io.nuls.consensus.poc.rpc.cmd;

import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.RestFulUtils;

import java.util.Map;

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription());
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getconsensus --get the whole network consensus infomation";
    }
```

```
    @Override
    public boolean argsValidate(String[] args) {
        if(args.length != 1){
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        RpcClientResult result = restFul.get("/consensus",null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> map = (Map)result.getData();
        map.put("rewardOfDay", CommandHelper.naToNuls(map.get("rewardOfDay")));
        map.put("totalDeposit", CommandHelper.naToNuls(map.get("totalDeposit")));
        result.setData(map);
        return CommandResult.getResult(result);
    }
}

116:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\GetDepositedAgentsProcessor.java
 */

package io.nuls.consensus.poc.rpc.cmd;

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

import java.util.Date;
import java.util.HashMap;
```

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

/**
 * ()
 * Get a list of deposited agent info based on your account
 * @author: Charlie
 */
public class GetDepositedAgentsProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<address> address - Required")
                .newLine("\t<pageNumber> pageNumber - Required")
                .newLine("\t<pageSize> pageSize(1~100) - Required");
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getdepositedagents <address> <pageNumber> <pageSize> --get a list of deposited
agent info based on your account";
    }

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

```
        if (!StringUtils.validAddressSimple(args[1]) || !StringUtils.isNumeric(args[2]) ||
!StringUtils.isNumeric(args[3])) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        int pageNumber = Integer.parseInt(args[2]);
        int pageSize = Integer.parseInt(args[3]);
        Map<String, Object> parameters = new HashMap<>(4);
        parameters.put("pageNumber", pageNumber);
        parameters.put("pageSize", pageSize);
        RpcClientResult result = restFul.get("/consensus/agent/address/" + address, parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        List<Map<String, Object>> list = (List<Map<String,
Object>>)((Map)result.getData()).get("list");
        for(Map<String, Object> map : list){
            map.put("deposit",  CommandHelper.naToNuls(map.get("deposit")));
            map.put("totalDeposit", CommandHelper.naToNuls(map.get("totalDeposit")));
            map.put("status", CommandHelper.consensusExplain((Integer) map.get("status")));
            map.put("time",  DateUtil.convertDate(new Date((Long)map.get("time"))));
        }
        result.setData(list);
        return CommandResult.getResult(result);
    }
}

117:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\GetDepositedInfoProcessor.java
 */

package io.nuls.consensus.poc.rpc.cmd;

import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.core.tools.str.StringUtils;
```

```
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.RestFulUtils;
import java.util.Map;

/**
 * ()
 * According to the account address to obtain all information on the deposit of the account
 * @author: Charlie
 */
public class GetDepositedInfoProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "getdepositedinfo <address>  --according to the account address to obtain overview on
the deposited of the account";
    }

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

```
        if(!StringUtils.validAddressSimple(args[1])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        RpcClientResult result = restFul.get("/consensus/address/" + address, null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> map = (Map)result.getData();
        map.put("usableBalance",  CommandHelper.naToNuls(map.get("usableBalance")));
        map.put("totalDeposit", CommandHelper.naToNuls(map.get("totalDeposit")));
        map.put("reward", CommandHelper.naToNuls(map.get("reward")));
        map.put("rewardOfDay", CommandHelper.naToNuls(map.get("rewardOfDay")));
        result.setData(map);
        return CommandResult.getResult(result);
    }
}

118:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\GetDepositedsProcessor.java
 */

package io.nuls.consensus.poc.rpc.cmd;

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

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

```
import java.util.Map;

/**
 * ()
 * Get a list of deposited info based on your account
 *
 * @author: Charlie
 */
public class GetDepositedsProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription())
                .newLine("\t<address> address - Required")
                .newLine("\t<pageNumber> pageNumber - Required")
                .newLine("\t<pageSize> pageSize(1~100) - Required")
                .newLine("\t[agentHash] the agent node hash (default query all)");
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getdepositeds <address> <pageNumber> <pageSize> [agentHash] --get a list of
deposited info based on your account";
    }

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

```
        }

        if (!StringUtils.validAddressSimple(args[1]) || !StringUtils.isNumeric(args[2]) ||
!StringUtils.isNumeric(args[3])) {
            return false;
        }
        if(length == 5 && !NulsDigestData.validHash(args[4])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        int pageNumber = Integer.parseInt(args[2]);
        int pageSize = Integer.parseInt(args[3]);
        Map<String, Object> parameters = new HashMap<>(4);
        parameters.put("pageNumber", pageNumber);
        parameters.put("pageSize", pageSize);
        if(args.length == 5){
            parameters.put("agentHash", args[4]);
        }
        RpcClientResult result = restFul.get("/consensus/deposit/address/" + address, parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        List<Map<String, Object>> list = (List<Map<String,
Object>>)((Map)result.getData()).get("list");
        for(Map<String, Object> map : list){
            map.put("deposit",  CommandHelper.naToNuls(map.get("deposit")));
            map.put("status", CommandHelper.consensusExplain((Integer) map.get("status")));
            map.put("time",  DateUtil.convertDate(new Date((Long)map.get("time"))));
        }
        result.setData(list);
        return CommandResult.getResult(result);
    }
}

119:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\StopAgentProcessor.java
 */
```

```
package io.nuls.consensus.poc.rpc.cmd;

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

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

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "stopagent <address> -- stop the agent";
    }

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

```
        if(length != 2){
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.validAddressSimple(args[1])){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        RpcClientResult res = CommandHelper.getPassword(address, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Map<String, Object> parameters = new HashMap<>(2);
        parameters.put("address", address);
        parameters.put("password", password);
        RpcClientResult result = restFul.post("/consensus/agent/stop", parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataTransformValue(result));
    }
}

120:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\WithdrawMultiProcessor.java
 */
package io.nuls.consensus.poc.rpc.cmd;

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.NulsDigestData;
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.Arrays;
import java.util.HashMap;
import java.util.Map;

/**
 * @author: tag
 */
public class WithdrawMultiProcessor implements CommandProcessor {
    private RestFulUtils restFul = RestFulUtils.getInstance();
    @Override
    public String getCommand() {
        return "withdrawMulti";
    }

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<address> \ttransfer address - Required")
                .newLine("\t<signAddress> \tsign address - Required")
                .newLine("\t<pubkey>,...<pubkey> \tPublic key that needs to be signed,If multiple
commas are used to separate.")
                .newLine("\t<m> \tAt least how many signatures are required to get the money.")
                .newLine("\t<txhash> \tCurrent consensus transaction hash")
                .newLine("\t<txdata> \tExit consensus transaction data currently created");
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "withdrawMulti --- If it's a trading promoter <address> <signAddress>
<pubkey>,...<pubkey> <m> <txhash>" +
                "\t          --- Else <address> <signAddress> <txdata>";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length != 4 && length != 6){
```

```
            return  false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        if(!StringUtils.validAddressSimple(args[1]) || !StringUtils.validAddressSimple(args[2])){
            return false;
        }
        if(length == 6){
            if(!StringUtils.validPubkeys(args[3],args[4])) {
                return false;
            }
            if(!NulsDigestData.validHash(args[5])) {
                return false;
            }
        }else{
            if(args[3] == null || args[3].length() == 0) {
                return false;
            }
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String signAddress = args[2];
        RpcClientResult res = CommandHelper.getPassword(signAddress, restFul);
        if(!res.isSuccess()){
            return CommandResult.getFailed(res);
        }
        String password = (String)res.getData();
        Map<String, Object> parameters = new HashMap<>();
        parameters.put("address", args[1]);
        parameters.put("signAddress", args[2]);
        if(args.length == 4){
            parameters.put("txdata",args[3]);
        }else{
            String[] pubkeys = args[3].split(",");
            parameters.put("pubkeys", Arrays.asList(pubkeys));
            parameters.put("m",Integer.parseInt(args[4]));
            parameters.put("txHash",Integer.parseInt(args[5]));
        }
```

```
        RpcClientResult result = restFul.post("/consensus/withdrawMutil", parameters);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataTransformValue(result));
    }
}
```

121:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\cmd\WithdrawProcessor.java
*/

```
package io.nuls.consensus.poc.rpc.cmd;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.RestFulUtils;

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

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

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

```
    bulider.newLine(getCommandDescription())
            .newLine("\t<address>   address -required")
            .newLine("\t<txHash>    your deposit transaction hash  -required");
    return bulider.toString();
}

@Override
public String getCommandDescription() {
    return "withdraw <address> <txHash> -- withdraw the agent";
}

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if(length != 3){
        return false;
    }
    if (!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    if(!StringUtils.validAddressSimple(args[1]) || !NulsDigestData.validHash(args[2])){
        return false;
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    String address = args[1];
    RpcClientResult res = CommandHelper.getPassword(address, restFul);
    if(!res.isSuccess()){
        return CommandResult.getFailed(res);
    }
    String password = (String)res.getData();
    Map<String, Object> parameters = new HashMap<>(3);
    parameters.put("address", address);
    parameters.put("txHash", args[2]);
    parameters.put("password", password);
    RpcClientResult result = restFul.post("/consensus/withdraw", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
    }
```

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

122:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\AccountConsensusInfoDTO.java
 */

package io.nuls.consensus.poc.rpc.model;

/**
 * @author Niels
 */
public class AccountConsensusInfoDTO {
    private int agentCount;
    private long totalDeposit;
    private int joinAgentCount;
    private long usableBalance;
    private long reward;
    private long rewardOfDay;
    private String agentHash;

    public int getAgentCount() {
        return agentCount;
    }

    public void setAgentCount(int agentCount) {
        this.agentCount = agentCount;
    }

    public long getTotalDeposit() {
        return totalDeposit;
    }

    public void setTotalDeposit(long totalDeposit) {
        this.totalDeposit = totalDeposit;
    }

    public int getJoinAgentCount() {
        return joinAgentCount;
    }
```

```
    public void setJoinAgentCount(int joinAgentCount) {
        this.joinAgentCount = joinAgentCount;
    }

    public long getUsableBalance() {
        return usableBalance;
    }

    public void setUsableBalance(long usableBalance) {
        this.usableBalance = usableBalance;
    }

    public long getReward() {
        return reward;
    }

    public void setReward(long reward) {
        this.reward = reward;
    }

    public long getRewardOfDay() {
        return rewardOfDay;
    }

    public void setRewardOfDay(long rewardOfDay) {
        this.rewardOfDay = rewardOfDay;
    }

    public void setAgentHash(String agentHash) {
        this.agentHash = agentHash;
    }

    public String getAgentHash() {
        return agentHash;
    }
}

123:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\AgentDTO.java
 */

package io.nuls.consensus.poc.rpc.model;
```

```
import com.fasterxml.jackson.annotation.JsonIgnore;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.util.PoConvertUtil;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.utils.AddressTool;

import java.io.UnsupportedEncodingException;

/**
 * @author Niels
 */
public class AgentDTO {

    public AgentDTO(Agent agent,String alias) {
        this.agentHash = agent.getTxHash().getDigestHex();
        this.agentAddress = AddressTool.getStringAddressByBytes(agent.getAgentAddress());
        this.packingAddress = AddressTool.getStringAddressByBytes(agent.getPackingAddress());
        this.rewardAddress = AddressTool.getStringAddressByBytes(agent.getRewardAddress());
        this.deposit = agent.getDeposit().getValue();
        this.commissionRate = agent.getCommissionRate();
        this.agentName = alias;
        this.agentId = PoConvertUtil.getAgentId(agent.getTxHash());
        this.time = agent.getTime();
        this.blockHeight = agent.getBlockHeight();
        this.delHeight = agent.getDelHeight();
        this.status = agent.getStatus();
        this.creditVal = agent.getCreditVal();
        this.totalDeposit = agent.getTotalDeposit();
        this.txHash = agent.getTxHash().getDigestHex();
        this.memberCount = agent.getMemberCount();
    }

    private String agentHash;

    private String agentAddress;

    private String packingAddress;
```

```
private String rewardAddress;

private long deposit;

private double commissionRate;

private String agentName;

private String agentId;

@JsonIgnore
private String introduction;

private long time;
private long blockHeight = -1L;
private long delHeight = -1L;
private int status;
private double creditVal;
private long totalDeposit;
private String txHash;
private final int memberCount;
private String version;

public String getAgentAddress() {
    return agentAddress;
}

public void setAgentAddress(String agentAddress) {
    this.agentAddress = agentAddress;
}

public String getPackingAddress() {
    return packingAddress;
}

public void setPackingAddress(String packingAddress) {
    this.packingAddress = packingAddress;
}

public String getRewardAddress() {
    return rewardAddress;
}
```

```
public void setRewardAddress(String rewardAddress) {
    this.rewardAddress = rewardAddress;
}

public long getDeposit() {
    return deposit;
}

public void setDeposit(long deposit) {
    this.deposit = deposit;
}

public double getCommissionRate() {
    return commissionRate;
}

public void setCommissionRate(double commissionRate) {
    this.commissionRate = commissionRate;
}

public String getAgentName() {
    return agentName;
}

public void setAgentName(String agentName) {
    this.agentName = agentName;
}

public String getIntroduction() {
    return introduction;
}

public void setIntroduction(String introduction) {
    this.introduction = introduction;
}

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 getDelHeight() {
    return delHeight;
}

public void setDelHeight(long delHeight) {
    this.delHeight = delHeight;
}

public int getStatus() {
    return status;
}

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

public double getCreditVal() {
    return creditVal;
}

public void setCreditVal(double creditVal) {
    this.creditVal = creditVal;
}

public long getTotalDeposit() {
    return totalDeposit;
}

public void setTotalDeposit(long totalDeposit) {
    this.totalDeposit = totalDeposit;
}
```

```
    public String getTxHash() {
        return txHash;
    }

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

    public int getMemberCount() {
        return memberCount;
    }

    public String getAgentHash() {
        return agentHash;
    }

    public void setAgentHash(String agentHash) {
        this.agentHash = agentHash;
    }

    public String getAgentId() {
        return agentId;
    }

    public void setAgentId(String agentId) {
        this.agentId = agentId;
    }

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }
}

124:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\ConsensusInfoDTO.java
 */
```

```
package io.nuls.consensus.poc.rpc.model;

/**
 * @author: Niels Wang
 */
public class ConsensusInfoDTO {

    private int agentCount;
    private long totalDeposit;
    private long reward;
    private int joinAccountCount;
    private long usableBalance;
    private long rewardOfDay;

    public long getRewardOfDay() {
        return rewardOfDay;
    }

    public void setRewardOfDay(long rewardOfDay) {
        this.rewardOfDay = rewardOfDay;
    }

    public int getAgentCount() {
        return agentCount;
    }

    public void setAgentCount(int agentCount) {
        this.agentCount = agentCount;
    }

    public long getTotalDeposit() {
        return totalDeposit;
    }

    public void setTotalDeposit(long totalDeposit) {
        this.totalDeposit = totalDeposit;
    }

    public long getReward() {
        return reward;
    }
```

```
    public void setReward(long reward) {
        this.reward = reward;
    }

    public int getJoinAccountCount() {
        return joinAccountCount;
    }

    public void setJoinAccountCount(int joinAccountCount) {
        this.joinAccountCount = joinAccountCount;
    }

    public long getUsableBalance() {
        return usableBalance;
    }

    public void setUsableBalance(long usableBalance) {
        this.usableBalance = usableBalance;
    }
}

125:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\CreateAgentForm.java
 *
 */

package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author Niels
 */
@ApiModel(value = "()")
public class CreateAgentForm {

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

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

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

@ApiModelProperty(name = "commissionRate", value = "", required = true)
private double commissionRate;

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

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

public double getCommissionRate() {
    return commissionRate;
}

public void setCommissionRate(double commissionRate) {
    this.commissionRate = commissionRate;
}

public String getPassword() {
    return password;
}

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

public String getAgentAddress() {
    return agentAddress;
}

public void setAgentAddress(String agentAddress) {
    this.agentAddress = StringUtils.formatStringPara(agentAddress);
}

public String getPackingAddress() {
    return packingAddress;
}

public void setPackingAddress(String packingAddress) {
```

```
        this.packingAddress = StringUtils.formatStringPara(packingAddress);
    }

    public String getRewardAddress() {
        return rewardAddress;
    }

    public void setRewardAddress(String rewardAddress) {
        this.rewardAddress = rewardAddress;
    }

    public long getDeposit() {
        return deposit;
    }

    public void setDeposit(long deposit) {
        this.deposit = deposit;
    }

}

126:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\CreateMultiAgentForm.java
 */
package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.util.List;

/**
 * @author tag
 */
@ApiModel(value = "()")
public class CreateMultiAgentForm {
    @ApiModelProperty(name = "agentAddress", value = "", required = true)
    private String agentAddress;

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

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

@ApiModelProperty(name = "commissionRate", value = "", required = true)
private double commissionRate;

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

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

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

public double getCommissionRate() {
    return commissionRate;
}

public void setCommissionRate(double commissionRate) {
    this.commissionRate = commissionRate;
}

public String getPassword() {
    return password;
}

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

public String getAgentAddress() {
    return agentAddress;
}

public void setAgentAddress(String agentAddress) {
    this.agentAddress = StringUtils.formatStringPara(agentAddress);
}

public String getPackingAddress() {
    return packingAddress;
```

```
    }

    public void setPackingAddress(String packingAddress) {
        this.packingAddress = StringUtils.formatStringPara(packingAddress);
    }

    public String getRewardAddress() {
        return rewardAddress;
    }

    public void setRewardAddress(String rewardAddress) {
        this.rewardAddress = rewardAddress;
    }

    public long getDeposit() {
        return deposit;
    }

    public void setDeposit(long deposit) {
        this.deposit = deposit;
    }

    public String getSignAddress() {
        return signAddress;
    }

    public void setSignAddress(String signAddress) {
        this.signAddress = signAddress;
    }

}

127:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\CreateMultiDepositForm.java
 */
package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import java.util.List;
```

```
/**
 * @author tag
 */
@ApiModel(value = "")
public class CreateMultiDepositForm {


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

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

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

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

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

    public String getAddress() {
        return address;
    }

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

    public String getAgentHash() {
        return agentHash;
    }

    public void setAgentHash(String agentHash) {
        this.agentHash = StringUtils.formatStringPara(agentHash);
    }

    public long getDeposit() {
        return deposit;
    }
```

```
    public void setDeposit(long deposit) {
        this.deposit = deposit;
    }

    public String getPassword() {
        return password;
    }

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

    public String getSignAddress() {
        return signAddress;
    }

    public void setSignAddress(String signAddress) {
        this.signAddress = signAddress;
    }

}

128:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\CreateMultiWithdrawForm.java
 */
package io.nuls.consensus.poc.rpc.model;

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

/**
 * @author tag
 */
@ApiModel(value = "")
public class CreateMultiWithdrawForm {
    @ApiModelProperty(name = "address", value = "", required = true)
    private String address;

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

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

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

    public String getAddress() {
        return address;
    }

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

    public String getTxHash() {
        return txHash;
    }

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

    public String getPassword() {
        return password;
    }

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

    public String getSignAddress() {
        return signAddress;
    }

    public void setSignAddress(String signAddress) {
        this.signAddress = signAddress;
    }
}

129:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\CreateStopMultiAgentForm.java
 */
```

```
package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModelProperty;

import java.util.List;

public class CreateStopMultiAgentForm {

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

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

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

    public String getPassword() {
        return password;
    }

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

    public String getSignAddress() {
        return signAddress;
    }

    public void setSignAddress(String signAddress) {
        this.signAddress = signAddress;
    }

    public String getAddress() {
        return address;
    }

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

```
}

130:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\DepositDTO.java
 */

package io.nuls.consensus.poc.rpc.model;

import io.nuls.kernel.model.Address;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.protocol.util.PoConvertUtil;
import io.nuls.core.tools.crypto.Base58;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.utils.AddressTool;

import java.io.UnsupportedEncodingException;

/**
 * @author Niels
 */
public class DepositDTO {

    private Long deposit;

    private String agentHash;

    private String address;

    private Long time;

    private String txHash;

    private Long blockHeight;

    private Long delHeight;

    /**
     * 0:, 1:
     */
```

```
private int status;

private String agentName;

private String agentAddress;

public DepositDTO(Deposit deposit) {
    this.deposit = deposit.getDeposit().getValue();
    this.agentHash = deposit.getAgentHash().getDigestHex();
    this.address = AddressTool.getStringAddressByBytes(deposit.getAddress());
    this.time = deposit.getTime();
    this.txHash = deposit.getTxHash().getDigestHex();
    this.blockHeight = deposit.getBlockHeight();
    this.delHeight = deposit.getDelHeight();
    this.status = deposit.getStatus();
}

public DepositDTO(Deposit deposit, Agent agent) {
    this(deposit);
    if (agent != null) {
        this.agentAddress = AddressTool.getStringAddressByBytes(agent.getAgentAddress());
        this.agentName = PoConvertUtil.getAgentId(agent.getTxHash());
    }
}

public Long getDeposit() {
    return deposit;
}

public void setDeposit(Long deposit) {
    this.deposit = deposit;
}

public String getAgentHash() {
    return agentHash;
}

public void setAgentHash(String agentHash) {
    this.agentHash = agentHash;
}

public String getAddress() {
```

```
    return address;
}

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

public Long getTime() {
    return time;
}

public void setTime(Long time) {
    this.time = time;
}

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 getDelHeight() {
    return delHeight;
}

public void setDelHeight(Long delHeight) {
    this.delHeight = delHeight;
}

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

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

    public String getAgentName() {
        return agentName;
    }

    public void setAgentName(String agentName) {
        this.agentName = agentName;
    }

    public String getAgentAddress() {
        return agentAddress;
    }

    public void setAgentAddress(String agentAddress) {
        this.agentAddress = agentAddress;
    }
}

131:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\DepositForm.java
 *
 */

package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author Niels
 */
@ApiModel(value = "")
public class DepositForm {

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

```
    @ApiModelProperty(name = "agentHash", value = "id", required = true)
    private String agentHash;

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

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

    public String getAddress() {
        return address;
    }

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

    public String getAgentHash() {
        return agentHash;
    }

    public void setAgentHash(String agentHash) {
        this.agentHash = StringUtils.formatStringPara(agentHash);
    }

    public long getDeposit() {
        return deposit;
    }

    public void setDeposit(long deposit) {
        this.deposit = deposit;
    }

    public String getPassword() {
        return password;
    }

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

}
```

```
132:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\GetCreateAgentFeeForm.java
 */

package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import javax.ws.rs.QueryParam;
import java.util.List;

/**
 * @author Niels
 */
@ApiModel(value = "()")
public class GetCreateAgentFeeForm {

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

    @ApiModelProperty(name = "packingAddress", value = "", required = true)
    @QueryParam("packingAddress")
    private String packingAddress;
    @ApiModelProperty(name = "rewardAddress", value = "", required = false)
    @QueryParam("rewardAddress")
    private String rewardAddress;

    @ApiModelProperty(name = "commissionRate", value = "", required = true)
    @QueryParam("commissionRate")
    private double commissionRate;

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

    public double getCommissionRate() {
        return commissionRate;
    }
```

```
    public void setCommissionRate(double commissionRate) {
        this.commissionRate = commissionRate;
    }

    public String getAgentAddress() {
        return agentAddress;
    }

    public void setAgentAddress(String agentAddress) {
        this.agentAddress = StringUtils.formatStringPara(agentAddress);
    }

    public String getPackingAddress() {
        return packingAddress;
    }

    public void setPackingAddress(String packingAddress) {
        this.packingAddress = StringUtils.formatStringPara(packingAddress);
    }

    public String getRewardAddress() {
        return rewardAddress;
    }

    public void setRewardAddress(String rewardAddress) {
        this.rewardAddress = rewardAddress;
    }

    public long getDeposit() {
        return deposit;
    }

    public void setDeposit(long deposit) {
        this.deposit = deposit;
    }

}

133:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\GetDepositFeeForm.java
 */
```

```
package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

import javax.ws.rs.QueryParam;
import java.util.List;

/**
 * @author Niels
 */
@ApiModel(value = "")
public class GetDepositFeeForm {

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

    @ApiModelProperty(name = "agentHash", value = "id", required = true)
    @QueryParam("agentHash")
    private String agentHash;

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

    public String getAddress() {
        return address;
    }

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

    public String getAgentHash() {
        return agentHash;
    }

    public void setAgentHash(String agentHash) {
        this.agentHash = StringUtils.formatStringPara(agentHash);
```

```
    }

    public long getDeposit() {
        return deposit;
    }

    public void setDeposit(long deposit) {
        this.deposit = deposit;
    }
}
```

134:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\model\PunishLogDTO.java

```
        this.type = po.getType();
        this.address = AddressTool.getStringAddressByBytes(po.getAddress());
        this.time = DateUtil.convertDate(new Date(po.getTime()));
        this.height = po.getHeight();
        this.roundIndex = po.getRoundIndex();
        this.reasonCode = PunishReasonEnum.getEnum(po.getReasonCode()).getMessage();
    }

    public byte getType() {
        return type;
    }

    public String getAddress() {
        return address;
    }

    public String getTime() {
        return time;
    }

    public long getHeight() {
        return height;
    }

    public long getRoundIndex() {
        return roundIndex;
    }

    public String getReasonCode() {
```

```
        return reasonCode;
    }
}

135:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\StopAgentForm.java
 *
 */

package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author Niels
 */
@ApiModel(value = "")
public class StopAgentForm {

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

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

    public String getAddress() {
        return address;
    }

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

    public String getPassword() {
        return password;
    }

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

```
}

136:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\WholeNetConsensusInfoDTO.java
 */

package io.nuls.consensus.poc.rpc.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author: Niels Wang
 */
@ApiModel(value = "wholeNetConsensusInfoJSON")
public class WholeNetConsensusInfoDTO {

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

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

    @ApiModelProperty(name = "rewardOfDay", value = "24")
    @JsonIgnore
    private long rewardOfDay;

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

    public int getConsensusAccountNumber() {
        return consensusAccountNumber;
    }

    public void setConsensusAccountNumber(int consensusAccountNumber) {
        this.consensusAccountNumber = consensusAccountNumber;
    }

    public int getAgentCount() {
        return agentCount;
```

```
    }

    public void setAgentCount(int agentCount) {
        this.agentCount = agentCount;
    }

    public long getTotalDeposit() {
        return totalDeposit;
    }

    public void setTotalDeposit(long totalDeposit) {
        this.totalDeposit = totalDeposit;
    }

    public long getRewardOfDay() {
        return rewardOfDay;
    }

    public void setRewardOfDay(long rewardOfDay) {
        this.rewardOfDay = rewardOfDay;
    }

    public void setPackingAgentCount(int packingAgentCount) {
        this.packingAgentCount = packingAgentCount;
    }

    public int getPackingAgentCount() {
        return packingAgentCount;
    }
}

137:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\model\WithdrawForm.java
 *
 */

package io.nuls.consensus.poc.rpc.model;

import io.nuls.core.tools.str.StringUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
/**
 * @author Niels
 */
@ApiModel(value = "")
public class WithdrawForm {

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

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

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = (address == null) ? null : address.trim();
    }

    public String getPassword() {
        return password;
    }

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

    public String getTxHash() {
        return txHash;
    }

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

138:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\main\java\io\nuls\consensus\poc\rpc\resource\PocConsensusResource.java

```
 */

package io.nuls.consensus.poc.rpc.resource;

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.account.ledger.model.CoinDataResult;
import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.account.model.Account;
import io.nuls.account.model.MultiSigAccount;
import io.nuls.account.service.AccountService;
import io.nuls.consensus.poc.constant.PocConsensusConstant;
import io.nuls.consensus.poc.context.PocConsensusContext;
import io.nuls.consensus.poc.model.MeetingMember;
import io.nuls.consensus.poc.model.MeetingRound;
import io.nuls.consensus.poc.protocol.constant.PocConsensusErrorCode;
import io.nuls.consensus.poc.protocol.entity.Agent;
import io.nuls.consensus.poc.protocol.entity.CancelDeposit;
import io.nuls.consensus.poc.protocol.entity.Deposit;
import io.nuls.consensus.poc.protocol.entity.StopAgent;
import io.nuls.consensus.poc.protocol.tx.CancelDepositTransaction;
import io.nuls.consensus.poc.protocol.tx.CreateAgentTransaction;
import io.nuls.consensus.poc.protocol.tx.DepositTransaction;
import io.nuls.consensus.poc.protocol.tx.StopAgentTransaction;
import io.nuls.consensus.poc.protocol.util.PoConvertUtil;
import io.nuls.consensus.poc.rpc.model.*;
import io.nuls.consensus.poc.rpc.utils.AgentComparator;
import io.nuls.consensus.poc.service.impl.PocRewardCacheService;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.util.ConsensusTool;
import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.json.JSONUtils;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.page.Page;
import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
```

```
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.*;
import io.nuls.kernel.script.*;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.TransactionFeeCalculator;
import io.nuls.kernel.utils.VarInt;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.base.version.ProtocolContainer;
import io.nuls.protocol.model.validator.TxMaxSizeValidator;
import io.nuls.protocol.service.TransactionService;
import io.swagger.annotations.*;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.*;

/**
 * @author Niels
 */
@Path("/consensus")
@Api(value = "/consensus", description = "poc-consensus")
@Component
public class PocConsensusResource {

    @Autowired
    private ConsensusService consensusService;

    @Autowired
    private AccountLedgerService accountLedgerService;

    @Autowired
    private AccountService accountService;

    @Autowired
    private TransactionService transactionService;

    @Autowired
```

```
    private LedgerService ledgerService;

    @Autowired
    private PocRewardCacheService rewardCacheService;

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("Get the whole network consensus infomation!  [3.6.1]")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response =
WholeNetConsensusInfoDTO.class)
    })
    public RpcClientResult getWholeInfo() {
        Result result = Result.getSuccess();
        WholeNetConsensusInfoDTO dto = new WholeNetConsensusInfoDTO();
        if (null == PocConsensusContext.getChainManager() || null ==
PocConsensusContext.getChainManager().getMasterChain()) {
            return Result.getFailed(KernelErrorCode.DATA_NOT_FOUND).toRpcClientResult();
        }

        List<Agent> allAgentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        long startBlockHeight = NulsContext.getInstance().getBestHeight();
        List<Agent> agentList = new ArrayList<>();

        for (int i = allAgentList.size() - 1; i >= 0; i--) {
            Agent agent = allAgentList.get(i);
            if (agent.getDelHeight() != -1L && agent.getDelHeight() <= startBlockHeight) {
                continue;
            } else if (agent.getBlockHeight() > startBlockHeight || agent.getBlockHeight() < 0L) {
                continue;
            }
            agentList.add(agent);
        }
        MeetingRound round =
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound();
        long totalDeposit = 0;
        int packingAgentCount = 0;
        if (null != round) {
            for (MeetingMember member : round.getMemberList()) {
                totalDeposit += (member.getTotalDeposit().getValue() +
member.getOwnDeposit().getValue());
```

```
                if (member.getAgent() != null) {
                    packingAgentCount++;
                }
            }
        }

        dto.setAgentCount(agentList.size());
        dto.setTotalDeposit(totalDeposit);
        dto.setConsensusAccountNumber(agentList.size());
        dto.setPackingAgentCount(packingAgentCount);
        result.setData(dto);
        return result.toRpcClientResult();
    }

    @GET
    @Path("/address/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = Map.class)
    })
    public RpcClientResult getInfo(@ApiParam(name = "address", value = "", required = true)
                                   @PathParam("address") String address) {

        if (!AddressTool.validAddress(StringUtils.formatStringPara(address))) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        Result accountResult = accountService.getAccount(address);
        if (accountResult.isFailed()) {
            return accountResult.toRpcClientResult();
        }
        //Account account = (Account) accountResult.getData();
        Result result = Result.getSuccess();
        AccountConsensusInfoDTO dto = new AccountConsensusInfoDTO();
        List<Agent> allAgentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        long startBlockHeight = NulsContext.getInstance().getBestHeight();
        int agentCount = 0;
        String agentHash = null;
        byte[] addressBytes = AddressTool.getAddress(address);
        for (int i = allAgentList.size() - 1; i >= 0; i--) {
            Agent agent = allAgentList.get(i);
```

```
            if (agent.getDelHeight() != -1L && agent.getDelHeight() <= startBlockHeight) {
                continue;
            } else if (agent.getBlockHeight() > startBlockHeight || agent.getBlockHeight() < 0L) {
                continue;
            }
            if (Arrays.equals(agent.getAgentAddress(), addressBytes)) {
                agentCount = 1;
                agentHash = agent.getTxHash().getDigestHex();
                break;
            }
        }
        List<Deposit> depositList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getDepositList();
        Set<NulsDigestData> agentSet = new HashSet<>();
        long totalDeposit = 0;
        for (Deposit deposit : depositList) {
            if (deposit.getDelHeight() != -1L && deposit.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (deposit.getBlockHeight() > startBlockHeight || deposit.getBlockHeight() < 0L) {
                continue;
            }
            if (!Arrays.equals(deposit.getAddress(), addressBytes)) {
                continue;
            }
            agentSet.add(deposit.getAgentHash());
            totalDeposit += deposit.getDeposit().getValue();
        }


        dto.setAgentCount(agentCount);
        dto.setAgentHash(agentHash);
        dto.setJoinAgentCount(agentSet.size());
        dto.setReward(this.rewardCacheService.getReward(address).getValue());
        dto.setRewardOfDay(rewardCacheService.getRewardToday(address).getValue());
        dto.setTotalDeposit(totalDeposit);
        try {
dto.setUsableBalance(accountLedgerService.getBalance(addressBytes).getData().getUsable().get
Value());
        } catch (Exception e) {
            Log.error(e);
            dto.setUsableBalance(0L);
```

```
    }
    result.setData(dto);
    return result.toRpcClientResult();
}


@GET
@Path("/agent/fee")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "get the fee of create agent! ()", notes = "")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = String.class)
})
public RpcClientResult getCreateAgentFee(
        @BeanParam() GetCreateAgentFeeForm form) throws NulsException {
    AssertUtil.canNotEmpty(form);
    AssertUtil.canNotEmpty(form.getAgentAddress(), "agent address can not be null");
    AssertUtil.canNotEmpty(form.getCommissionRate(), "commission rate can not be null");
    AssertUtil.canNotEmpty(form.getDeposit(), "deposit can not be null");
    AssertUtil.canNotEmpty(form.getPackingAddress(), "packing address can not be null");
    if (StringUtils.isBlank(form.getRewardAddress())) {
        form.setRewardAddress(form.getAgentAddress());
    }
    CreateAgentTransaction tx = new CreateAgentTransaction();
    tx.setTime(TimeService.currentTimeMillis());
    Agent agent = new Agent();
    agent.setAgentAddress(AddressTool.getAddress(form.getAgentAddress()));
    agent.setPackingAddress(AddressTool.getAddress(form.getPackingAddress()));
    if (StringUtils.isBlank(form.getRewardAddress())) {
        agent.setRewardAddress(agent.getAgentAddress());
    } else {
        agent.setRewardAddress(AddressTool.getAddress(form.getRewardAddress()));
    }

    agent.setDeposit(Na.valueOf(form.getDeposit()));
    agent.setCommissionRate(form.getCommissionRate());
    tx.setTxData(agent);
    CoinData coinData = new CoinData();
    List<Coin> toList = new ArrayList<>();
    if (agent.getAgentAddress()[2] == 3) {
        Script scriptPubkey = SignatureUtil.createOutputScript(agent.getAgentAddress());
        toList.add(new Coin(scriptPubkey.getProgram(), agent.getDeposit(), -1));
```

```
        } else {
            toList.add(new Coin(agent.getAgentAddress(), agent.getDeposit(), -1));
        }
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        CoinDataResult result = accountLedgerService.getCoinData(agent.getAgentAddress(),
agent.getDeposit(), tx.size() - P2PHKSignature.SERIALIZE_LENGTH,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        tx.getCoinData().setFrom(result.getCoinList());
        if (null != result.getChange()) {
            tx.getCoinData().getTo().add(result.getChange());
        }
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        Map<String, Long> map = new HashMap<>();
        map.put("fee", fee.getValue());
        map.put("maxAmount", getMaxAmount(fee, form.getAgentAddress(), tx));
        return Result.getSuccess().setData(map).toRpcClientResult();
    }

    @GET
    @Path("/deposit/fee")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "get the fee of create agent! ", notes = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult getDepositFee(@BeanParam() GetDepositFeeForm form) throws
NulsException {
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getAddress(), "address can not be null");
        AssertUtil.canNotEmpty(form.getAgentHash(), "agent hash can not be null");
        AssertUtil.canNotEmpty(form.getDeposit(), "deposit can not be null");
        DepositTransaction tx = new DepositTransaction();
        Deposit deposit = new Deposit();
        deposit.setAddress(AddressTool.getAddress(form.getAddress()));
        deposit.setAgentHash(NulsDigestData.fromDigestHex(form.getAgentHash()));
        deposit.setDeposit(Na.valueOf(form.getDeposit()));
        tx.setTxData(deposit);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        toList.add(new Coin(deposit.getAddress(), deposit.getDeposit(), -1));
        coinData.setTo(toList);
```

```
        tx.setCoinData(coinData);
        CoinDataResult result = accountLedgerService.getCoinData(deposit.getAddress(),
deposit.getDeposit(), tx.size(), TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        tx.getCoinData().setFrom(result.getCoinList());
        if (null != result.getChange()) {
            tx.getCoinData().getTo().add(result.getChange());
        }
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        Map<String, Long> map = new HashMap<>();
        map.put("fee", fee.getValue());
        map.put("maxAmount", getMaxAmount(fee, form.getAddress(), tx));
        return Result.getSuccess().setData(map).toRpcClientResult();
    }

    //null
    private Long getMaxAmount(Na fee, String address, Transaction tx) {
        long feeMax =
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES.multiply(TxMaxSizeValidator.MA
X_TX_BYTES).getValue();
        Long maxAmount = null;
        if (fee.getValue() > feeMax) {
            Result rs =
accountLedgerService.getMaxAmountOfOnce(AddressTool.getAddress(address), tx,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
            if (rs.isSuccess()) {
                maxAmount = ((Na) rs.getData()).getValue();
            }
        }
        return maxAmount;
    }

    @GET
    @Path("/agent/stop/fee")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "get the fee of stop agent! ", notes = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult getStopAgentFee(@ApiParam(name = "address", value = "", required =
true)
                                           @QueryParam("address") String address) throws NulsException,
IOException {
```

```
        AssertUtil.canNotEmpty(address, "address can not be null");
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

        StopAgentTransaction tx = new StopAgentTransaction();
        StopAgent stopAgent = new StopAgent();
        stopAgent.setAddress(AddressTool.getAddress(address));
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        Agent agent = null;
        for (Agent a : agentList) {
            if (a.getDelHeight() > 0) {
                continue;
            }
            if (Arrays.equals(a.getAgentAddress(), stopAgent.getAddress())) {
                agent = a;
                break;
            }
        }
        if (agent == null || agent.getDelHeight() > 0) {
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }
        NulsDigestData createTxHash = agent.getTxHash();
        stopAgent.setCreateTxHash(createTxHash);
        tx.setTxData(stopAgent);
        CoinData coinData = ConsensusTool.getStopAgentCoinData(agent,
TimeService.currentTimeMillis() + PocConsensusConstant.STOP_AGENT_LOCK_TIME);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        coinData.getTo().get(0).setNa(coinData.getTo().get(0).getNa().subtract(fee));
        Na resultFee = TransactionFeeCalculator.getMaxFee(tx.size());
        Map<String, Long> map = new HashMap<>();
        map.put("fee", fee.getValue());
        map.put("maxAmount", getMaxAmount(resultFee, address, tx));
        return Result.getSuccess().setData(map).toRpcClientResult();
    }

    @GET
    @Path("/withdraw/fee")
    @Produces(MediaType.APPLICATION_JSON)
```

```
    @ApiOperation(value = "get the fee of cancel deposit! ", notes = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult getWithdrawFee(@ApiParam(name = "address", value = "", required =
true)
                                          @QueryParam("address") String address,
                                          @ApiParam(name = "depositTxHash", value = "", required = true)
                                          @QueryParam("depositTxHash") String depositTxHash) throws
NulsException, IOException {
        AssertUtil.canNotEmpty(depositTxHash);
        if (!NulsDigestData.validHash(depositTxHash)) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        AssertUtil.canNotEmpty(address);
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        Account account = accountService.getAccount(address).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        CancelDepositTransaction tx = new CancelDepositTransaction();
        CancelDeposit cancelDeposit = new CancelDeposit();
        NulsDigestData hash = NulsDigestData.fromDigestHex(depositTxHash);
        DepositTransaction depositTransaction = (DepositTransaction) ledgerService.getTx(hash);
        if (null == depositTransaction) {
            return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST).toRpcClientResult();
        }
        cancelDeposit.setAddress(account.getAddress().getAddressBytes());
        cancelDeposit.setJoinTxHash(hash);
        tx.setTxData(cancelDeposit);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        toList.add(new Coin(cancelDeposit.getAddress(),
depositTransaction.getTxData().getDeposit(), 0));
        coinData.setTo(toList);
        List<Coin> fromList = new ArrayList<>();
        for (int index = 0; index < depositTransaction.getCoinData().getTo().size(); index++) {
            Coin coin = depositTransaction.getCoinData().getTo().get(index);
            if (coin.getLockTime() == -1L &&
coin.getNa().equals(depositTransaction.getTxData().getDeposit())) {
```

```
                coin.setOwner(ArraysTool.concatenate(hash.serialize(), new VarInt(index).encode()));
                fromList.add(coin);
                break;
            }
        }
        if (fromList.isEmpty()) {
            return Result.getFailed(KernelErrorCode.DATA_ERROR).toRpcClientResult();
        }
        coinData.setFrom(fromList);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        coinData.getTo().get(0).setNa(coinData.getTo().get(0).getNa().subtract(fee));
        Na resultFee = TransactionFeeCalculator.getMaxFee(tx.size());

        Map<String, Long> map = new HashMap<>();
        map.put("fee", fee.getValue());
        map.put("maxAmount", getMaxAmount(resultFee, account.getAddress().getBase58(), tx));
        return Result.getSuccess().setData(map).toRpcClientResult();
    }


    @POST
    @Path("/agent")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "Create an agent for consensus! () [3.6.3]", notes = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult createAgent(@ApiParam(name = "form", value = "", required = true)
                                       CreateAgentForm form) throws NulsException {
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getAgentAddress(), "agent address can not be null");
        AssertUtil.canNotEmpty(form.getCommissionRate(), "commission rate can not be null");
        AssertUtil.canNotEmpty(form.getDeposit(), "deposit can not be null");
        AssertUtil.canNotEmpty(form.getPackingAddress(), "packing address can not be null");

        if (!AddressTool.isPackingAddress(form.getPackingAddress()) ||
!AddressTool.validAddress(form.getAgentAddress())) {
            throw new NulsRuntimeException(AccountErrorCode.ADDRESS_ERROR);
        }
        Account account = accountService.getAccount(form.getAgentAddress()).getData();
        if (null == account) {
```

```
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }
        CreateAgentTransaction tx = new CreateAgentTransaction();
        tx.setTime(TimeService.currentTimeMillis());
        Agent agent = new Agent();
        agent.setAgentAddress(AddressTool.getAddress(form.getAgentAddress()));
        agent.setPackingAddress(AddressTool.getAddress(form.getPackingAddress()));
        if (StringUtils.isBlank(form.getRewardAddress())) {
            agent.setRewardAddress(agent.getAgentAddress());
        } else {
            agent.setRewardAddress(AddressTool.getAddress(form.getRewardAddress()));
        }

        agent.setDeposit(Na.valueOf(form.getDeposit()));
        agent.setCommissionRate(form.getCommissionRate());
        tx.setTxData(agent);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        toList.add(new Coin(agent.getAgentAddress(), agent.getDeposit(),
PocConsensusConstant.CONSENSUS_LOCK_TIME));
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        CoinDataResult result = accountLedgerService.getCoinData(agent.getAgentAddress(),
agent.getDeposit(), tx.size(), TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        RpcClientResult result1 = this.txProcessing(tx, result, account, form.getPassword());
        if (!result1.isSuccess()) {
            return result1;
        }
        Map<String, String> valueMap = new HashMap<>();
        valueMap.put("value", tx.getHash().getDigestHex());
        return Result.getSuccess().setData(valueMap).toRpcClientResult();
    }


    @POST
```

```
    @Path("/deposit")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "deposit nuls to a bank!  ", notes = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult depositToAgent(@ApiParam(name = "form", value = "", required = true)
                                          DepositForm form) throws NulsException {
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getAddress());
        AssertUtil.canNotEmpty(form.getAgentHash());
        if (!NulsDigestData.validHash(form.getAgentHash())) {
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }
        AssertUtil.canNotEmpty(form.getDeposit());
        if (!AddressTool.validAddress(form.getAddress())) {
            throw new NulsRuntimeException(KernelErrorCode.PARAMETER_ERROR);
        }
        Account account = accountService.getAccount(form.getAddress()).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }

        DepositTransaction tx = new DepositTransaction();
        Deposit deposit = new Deposit();
        deposit.setAddress(AddressTool.getAddress(form.getAddress()));
        deposit.setAgentHash(NulsDigestData.fromDigestHex(form.getAgentHash()));
        deposit.setDeposit(Na.valueOf(form.getDeposit()));
        tx.setTxData(deposit);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        toList.add(new Coin(deposit.getAddress(), deposit.getDeposit(),
PocConsensusConstant.CONSENSUS_LOCK_TIME));
        coinData.setTo(toList);
```

```
        tx.setCoinData(coinData);
        CoinDataResult result = accountLedgerService.getCoinData(deposit.getAddress(),
deposit.getDeposit(), tx.size(), TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);

        RpcClientResult result1 = this.txProcessing(tx, result, account, form.getPassword());
        if (!result1.isSuccess()) {
            return result1;
        }
        Map<String, String> valueMap = new HashMap<>();
        valueMap.put("value", tx.getHash().getDigestHex());
        return Result.getSuccess().setData(valueMap).toRpcClientResult();
    }

    public RpcClientResult txProcessing(Transaction tx, CoinDataResult result, Account account,
String password) {
        if (null != result) {
            if (result.isEnough()) {
                tx.getCoinData().setFrom(result.getCoinList());
                if (null != result.getChange()) {
                    tx.getCoinData().getTo().add(result.getChange());
                }
            } else {
                return
Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE).toRpcClientResult();
            }
        }
        try {
            tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
            //
            List<ECKey> signEckeys = new ArrayList<>();
            ECKey eckey = account.getEcKey(password);
            signEckeys.add(eckey);
            SignatureUtil.createTransactionSignture(tx, null, signEckeys);

        } catch (Exception e) {
            Log.error(e);
            return
Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
        }
        Result saveResult = accountLedgerService.verifyAndSaveUnconfirmedTransaction(tx);
        if (saveResult.isFailed()) {
            if
```

```
(KernelErrorCode.DATA_SIZE_ERROR.getCode().equals(saveResult.getErrorCode().getCode()))
{
            //()
            Result rs =
accountLedgerService.getMaxAmountOfOnce(account.getAddress().getAddressBytes(), tx,
                    TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
            if (rs.isSuccess()) {
                Na maxAmount = (Na) rs.getData();
                rs = Result.getFailed(KernelErrorCode.DATA_SIZE_ERROR_EXTEND);
                rs.setMsg(rs.getMsg() + maxAmount.toDouble());
            }
            return rs.toRpcClientResult();

        }
        return saveResult.toRpcClientResult();
    }
    transactionService.newTx(tx);
    Result sendResult = this.transactionService.broadcastTx(tx);
    if (sendResult.isFailed()) {
        accountLedgerService.deleteTransaction(tx);
        return sendResult.toRpcClientResult();
    }
    return Result.getSuccess().toRpcClientResult();
}


@POST
@Path("/agent/stop")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = " [3.6.5]", notes = "hash")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = String.class)
})
public RpcClientResult stopAgent(@ApiParam(name = "form", value = "", required = true)
                                 StopAgentForm form) throws NulsException, IOException {
    AssertUtil.canNotEmpty(form);
    AssertUtil.canNotEmpty(form.getAddress());
    if (!AddressTool.validAddress(form.getAddress())) {
        throw new NulsRuntimeException(KernelErrorCode.PARAMETER_ERROR);
    }
    Account account = accountService.getAccount(form.getAddress()).getData();
    if (null == account) {
```

```
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }
        StopAgentTransaction tx = new StopAgentTransaction();
        StopAgent stopAgent = new StopAgent();
        stopAgent.setAddress(AddressTool.getAddress(form.getAddress()));
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        Agent agent = null;
        for (Agent a : agentList) {
            if (a.getDelHeight() > 0) {
                continue;
            }
            if (Arrays.equals(a.getAgentAddress(), account.getAddress().getAddressBytes())) {
                agent = a;
                break;
            }
        }
        if (agent == null || agent.getDelHeight() > 0) {
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }

        stopAgent.setCreateTxHash(agent.getTxHash());
        tx.setTxData(stopAgent);

        CoinData coinData = ConsensusTool.getStopAgentCoinData(agent,
TimeService.currentTimeMillis() + PocConsensusConstant.STOP_AGENT_LOCK_TIME);

        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        coinData.getTo().get(0).setNa(coinData.getTo().get(0).getNa().subtract(fee));
        RpcClientResult result1 = this.txProcessing(tx, null, account, form.getPassword());
        if (!result1.isSuccess()) {
            return result1;
        }
```

```
        Map<String, String> valueMap = new HashMap<>();
        valueMap.put("value", tx.getHash().getDigestHex());
        return Result.getSuccess().setData(valueMap).toRpcClientResult();
    }


    @GET
    @Path("/agent/list")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = " [3.6.6]", notes = "result.data.page.list: List<Map<String, Object>>")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = Page.class)
    })
    public RpcClientResult getAgentList(@ApiParam(name = "pageNumber", value = "")
                                        @QueryParam("pageNumber") Integer pageNumber,
                                        @ApiParam(name = "pageSize", value = "")
                                        @QueryParam("pageSize") Integer pageSize,
                                        @ApiParam(name = "keyword", value = "")
                                        @QueryParam("keyword") String keyword,
                                        @ApiParam(name = "sortType", value = "")
                                        @QueryParam("sortType") String sortType) throws
UnsupportedEncodingException {
        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();
        }
        Result result = Result.getSuccess();
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        agentList = new ArrayList<>(agentList);
        long startBlockHeight = NulsContext.getInstance().getBestHeight();


        for (int i = agentList.size() - 1; i >= 0; i--) {
            Agent agent = agentList.get(i);
            if (agent.getDelHeight() != -1L && agent.getDelHeight() <= startBlockHeight) {
                agentList.remove(i);
```

```
            } else if (agent.getBlockHeight() > startBlockHeight || agent.getBlockHeight() < 0L) {
                agentList.remove(i);
            } else if (StringUtils.isNotBlank(keyword)) {
                keyword = keyword.toUpperCase();
                String agentAddress =
AddressTool.getStringAddressByBytes(agent.getAgentAddress()).toUpperCase();
                String packingAddress =
AddressTool.getStringAddressByBytes(agent.getPackingAddress()).toUpperCase();
                String agentId = PoConvertUtil.getAgentId(agent.getTxHash()).toUpperCase();
                String alias = accountService.getAlias(agent.getAgentAddress()).getData();
                boolean b = agentId.indexOf(keyword) >= 0;
                b = b || agentAddress.equals(keyword) || packingAddress.equals(keyword);
                if (StringUtils.isNotBlank(alias)) {
                    b = b || alias.toUpperCase().indexOf(keyword) >= 0;
                }
                if (!b) {
                    agentList.remove(i);
                }
            }
        }
        int start = pageNumber * pageSize - pageSize;
        Page<AgentDTO> page = new Page<>(pageNumber, pageSize, agentList.size());
        if (start >= page.getTotal()) {
            result.setData(page);
            return result.toRpcClientResult();
        }
        fillAgentList(agentList, null);
        int type = AgentComparator.COMPREHENSIVE;
        if ("deposit".equals(sortType)) {
            type = AgentComparator.DEPOSIT;
        } else if ("commissionRate".equals(sortType)) {
            type = AgentComparator.COMMISSION_RATE;
        } else if ("creditVal".equals(sortType)) {
            type = AgentComparator.CREDIT_VALUE;
        } else if ("totalDeposit".equals(sortType)) {
            type = AgentComparator.DEPOSITABLE;
        } else if ("comprehensive".equals(sortType)) {
            type = AgentComparator.COMPREHENSIVE;
        }
        Collections.sort(agentList, AgentComparator.getInstance(type));

        List<AgentDTO> resultList = new ArrayList<>();
```

```
        for (int i = start; i < agentList.size() && i < (start + pageSize); i++) {
            Agent agent = agentList.get(i);
            AgentDTO agentDTO = new AgentDTO(agent,
accountService.getAlias(agent.getAgentAddress()).getData());
            Integer version =
NulsVersionManager.getConsensusVersionMap().get(agentDTO.getPackingAddress());
            if (version == null) {
                agentDTO.setVersion("--");
            } else {
                agentDTO.setVersion(version + "");
            }
            resultList.add(agentDTO);
        }
        page.setList(resultList);
        result.setData(page);
        return result.toRpcClientResult();
    }

    private void fillAgentList(List<Agent> agentList, List<Deposit> depositList) {
        MeetingRound round =
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound();
        for (Agent agent : agentList) {
            fillAgent(agent, round, depositList);
        }
    }

    private void fillAgent(Agent agent, MeetingRound round, List<Deposit> depositList) {
        if (null == depositList || depositList.isEmpty()) {
            depositList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getDepositList();
        }
        Set<String> memberSet = new HashSet<>();
        Na total = Na.ZERO;
        for (int i = 0; i < depositList.size(); i++) {
            Deposit deposit = depositList.get(i);
            if (!agent.getTxHash().equals(deposit.getAgentHash())) {
                continue;
            }
            if (deposit.getDelHeight() >= 0) {
                continue;
            }
            total = total.add(deposit.getDeposit());
```

```
            memberSet.add(AddressTool.getStringAddressByBytes(deposit.getAddress()));
        }
        agent.setMemberCount(memberSet.size());
        agent.setTotalDeposit(total.getValue());

        if (round == null) {
            return;
        }
        MeetingMember member = round.getMember(agent.getPackingAddress());
        if (null == member) {
            agent.setStatus(0);
            return;
        }
        agent.setStatus(1);
        agent.setCreditVal(member.getCreditVal());

    }

    @GET
    @Path("/agent/{agentHash}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = " [3.6.7]", notes = "result.data: Map<String, Object>")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = Map.class)
    })
    public RpcClientResult getAgent(@ApiParam(name = "agentHash", value = "", required = true)
                                    @PathParam("agentHash") String agentHash) throws NulsException {

        if (!NulsDigestData.validHash(agentHash)) {
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }
        Result result = Result.getSuccess();
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        NulsDigestData agentHashData = NulsDigestData.fromDigestHex(agentHash);
        for (Agent agent : agentList) {
            if (agent.getTxHash().equals(agentHashData)) {
                MeetingRound round =
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound();
                this.fillAgent(agent, round, null);
                String alias = accountService.getAlias(agent.getAgentAddress()).getData();
```

```
            AgentDTO dto = new AgentDTO(agent, alias);
            result.setData(dto);
            return result.toRpcClientResult();
        }
    }
    return Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
}

@GET
@Path("/agent/address/{address}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = " [3.6.12]",
        notes = "result.data.page.list: List<Map<String, Object>>")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = Page.class)
})
public RpcClientResult getAgentListByDepositAddress(@ApiParam(name = "pageNumber",
value = "")
                                                    @QueryParam("pageNumber") Integer pageNumber,
                                                    @ApiParam(name = "pageSize", value = "")
                                                    @QueryParam("pageSize") Integer pageSize,
                                                    @ApiParam(name = "address", value = "", required = true)
                                                    @PathParam("address") String address) {
    AssertUtil.canNotEmpty(address);
    if (!AddressTool.validAddress(address)) {
        return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
    }
    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();
    }
    Result result = Result.getSuccess();
    List<Deposit> allList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getDepositList();
    long startBlockHeight = NulsContext.getInstance().getBestHeight();
    byte[] addressBytes = AddressTool.getAddress(address);
    Set<NulsDigestData> agentHashSet = new HashSet<>();
```

```
        for (Deposit deposit : allList) {
            if (deposit.getDelHeight() != -1L && deposit.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (deposit.getBlockHeight() > startBlockHeight || deposit.getBlockHeight() < 0L) {
                continue;
            }
            if (!Arrays.equals(deposit.getAddress(), addressBytes)) {
                continue;
            }
            agentHashSet.add(deposit.getAgentHash());
        }
        List<Agent> allAgentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        List<Agent> agentList = new ArrayList<>();
        Agent ownAgent = null;
        for (int i = allAgentList.size() - 1; i >= 0; i--) {
            Agent agent = allAgentList.get(i);
            if (agent.getDelHeight() != -1L && agent.getDelHeight() <= startBlockHeight) {
                continue;
            } else if (agent.getBlockHeight() > startBlockHeight || agent.getBlockHeight() < 0L) {
                continue;
            }
            if (Arrays.equals(agent.getAgentAddress(), addressBytes)) {
                ownAgent = agent;
                continue;
            }
            if (!agentHashSet.contains(agent.getTxHash())) {
                continue;
            }
            agentList.add(agent);
        }
        if (null != ownAgent) {
            agentList.add(0, ownAgent);
        }
        int start = pageNumber * pageSize - pageSize;
        Page<AgentDTO> page = new Page<>(pageNumber, pageSize, agentList.size());
        if (start >= agentList.size()) {
            result.setData(page);
            return result.toRpcClientResult();
        }
        fillAgentList(agentList, allList);
```

```
        List<AgentDTO> resultList = new ArrayList<>();
        for (int i = start; i < agentList.size() && i < (start + pageSize); i++) {
            Agent agent = agentList.get(i);
            resultList.add(new AgentDTO(agent,
accountService.getAlias(agent.getAgentAddress()).getData()));
        }
        page.setList(resultList);
        result.setData(page);
        return result.toRpcClientResult();
    }

    @GET
    @Path("/deposit/address/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = " [3.6.8]",
            notes = "result.data.page.list: List<Map<String, Object>>")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = Page.class)
    })
    public RpcClientResult getDepositListByAddress(@ApiParam(name = "address", value = "",
required = true)
                                                   @PathParam("address") String address,
                                                   @ApiParam(name = "pageNumber", value = "")
                                                   @QueryParam("pageNumber") Integer pageNumber,
                                                   @ApiParam(name = "pageSize", value = "")
                                                   @QueryParam("pageSize") Integer pageSize,
                                                   @ApiParam(name = "agentHash", value = "()")
                                                   @QueryParam("agentHash") String agentHash) {
        AssertUtil.canNotEmpty(address);
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        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 (null != agentHash && !NulsDigestData.validHash(agentHash)) {
```

```
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }
        Result result = Result.getSuccess();
        List<Deposit> allList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getDepositList();
        List<Deposit> depositList = new ArrayList<>();
        long startBlockHeight = NulsContext.getInstance().getBestHeight();
        byte[] addressBytes = AddressTool.getAddress(address);
        for (Deposit deposit : allList) {
            if (deposit.getDelHeight() != -1L && deposit.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (deposit.getBlockHeight() > startBlockHeight || deposit.getBlockHeight() < 0L) {
                continue;
            }
            if (!Arrays.equals(deposit.getAddress(), addressBytes)) {
                continue;
            }
            if (agentHash != null && !deposit.getAgentHash().getDigestHex().equals(agentHash)) {
                continue;
            }
            depositList.add(deposit);
        }
        int start = pageNumber * pageSize - pageSize;
        Page<DepositDTO> page = new Page<>(pageNumber, pageSize, depositList.size());
        if (start >= depositList.size()) {
            result.setData(page);
            return result.toRpcClientResult();
        }
        Map<NulsDigestData, Agent> map = new HashMap<>();
        for (MeetingMember member :
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound().getMemberList())
{
            if (null != member.getAgent()) {
                map.put(member.getAgent().getTxHash(), member.getAgent());
            }
        }
        List<DepositDTO> resultList = new ArrayList<>();

        for (int i = start; i < depositList.size() && i < (start + pageSize); i++) {
            Deposit deposit = depositList.get(i);
```

```
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        Agent agent = null;
        for (Agent a : agentList) {
            if (a.getTxHash().equals(deposit.getAgentHash())) {
                agent = a;
                break;
            }
        }
        deposit.setStatus(agent == null ? 0 : agent.getStatus());
        resultList.add(new DepositDTO(deposit, agent));
    }
    page.setList(resultList);
    result.setData(page);
    return result.toRpcClientResult();
}

@GET
@Path("/deposit/agent/{agentHash}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = " [3.6.9]",
        notes = "result.data.page.list: List<Map<String, Object>>")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = Page.class)
})
public RpcClientResult queryDepositListByAgentAddress(@ApiParam(name = "agentHash",
value = "", required = true)
                                                      @PathParam("agentHash") String agentHash,
                                                      @ApiParam(name = "pageNumber", value = "")
                                                      @QueryParam("pageNumber") Integer pageNumber,
                                                      @ApiParam(name = "pageSize", value = "")
                                                      @QueryParam("pageSize") Integer pageSize) throws
NulsException {
    AssertUtil.canNotEmpty(agentHash);
    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();
```

```
        }
        Result result = Result.getSuccess();
        List<Deposit> allList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getDepositList();
        List<Deposit> depositList = new ArrayList<>();
        long startBlockHeight = NulsContext.getInstance().getBestHeight();
        NulsDigestData agentDigestData = NulsDigestData.fromDigestHex(agentHash);
        for (Deposit deposit : allList) {
            if (deposit.getDelHeight() != -1L && deposit.getDelHeight() <= startBlockHeight) {
                continue;
            }
            if (deposit.getBlockHeight() > startBlockHeight || deposit.getBlockHeight() < 0L) {
                continue;
            }
            if (!deposit.getAgentHash().equals(agentDigestData)) {
                continue;
            }
            depositList.add(deposit);
        }
        int start = pageNumber * pageSize - pageSize;
        Page<DepositDTO> page = new Page<>(pageNumber, pageSize, depositList.size());
        if (start >= depositList.size()) {
            result.setData(page);
            return result.toRpcClientResult();
        }
        Map<NulsDigestData, Integer> map = new HashMap<>();
        for (MeetingMember member :
PocConsensusContext.getChainManager().getMasterChain().getCurrentRound().getMemberList())
{
            if (null != member.getAgent()) {
                map.put(member.getAgent().getTxHash(), 1);
            }
        }

        List<DepositDTO> resultList = new ArrayList<>();
        for (int i = start; i < depositList.size() && i < (start + pageSize); i++) {
            Deposit deposit = depositList.get(i);
            deposit.setStatus(map.get(deposit.getAgentHash()) == null ? 0 : 1);
            resultList.add(new DepositDTO(deposit));
        }
        page.setList(resultList);
        result.setData(page);
```

```
        return result.toRpcClientResult();
    }

    @POST
    @Path("/withdraw")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = " [3.6.11]",
            notes = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult withdraw(@ApiParam(name = "form", value = "", required = true)
                                    WithdrawForm form) throws NulsException, IOException {
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getTxHash());
        if (!NulsDigestData.validHash(form.getTxHash())) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        AssertUtil.canNotEmpty(form.getAddress());
        if (!AddressTool.validAddress(form.getAddress())) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        Account account = accountService.getAccount(form.getAddress()).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }
        CancelDepositTransaction tx = new CancelDepositTransaction();
        CancelDeposit cancelDeposit = new CancelDeposit();
        NulsDigestData hash = NulsDigestData.fromDigestHex(form.getTxHash());
        DepositTransaction depositTransaction = null;
        try {
            depositTransaction = (DepositTransaction) ledgerService.getTx(hash);
        } catch (Exception e) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
```

```
        if (null == depositTransaction) {
            return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST).toRpcClientResult();
        }
        cancelDeposit.setAddress(AddressTool.getAddress(form.getAddress()));
        cancelDeposit.setJoinTxHash(hash);
        tx.setTxData(cancelDeposit);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        toList.add(new Coin(cancelDeposit.getAddress(),
depositTransaction.getTxData().getDeposit(), 0));
        coinData.setTo(toList);
        List<Coin> fromList = new ArrayList<>();
        for (int index = 0; index < depositTransaction.getCoinData().getTo().size(); index++) {
            Coin coin = depositTransaction.getCoinData().getTo().get(index);
            if (coin.getLockTime() == -1L &&
coin.getNa().equals(depositTransaction.getTxData().getDeposit())) {
                coin.setOwner(ArraysTool.concatenate(hash.serialize(), new VarInt(index).encode()));
                fromList.add(coin);
                break;
            }
        }
        if (fromList.isEmpty()) {
            return Result.getFailed(KernelErrorCode.DATA_ERROR).toRpcClientResult();
        }
        coinData.setFrom(fromList);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        coinData.getTo().get(0).setNa(coinData.getTo().get(0).getNa().subtract(fee));
        RpcClientResult result1 = this.txProcessing(tx, null, account, form.getPassword());
        if (!result1.isSuccess()) {
            return result1;
        }
        Map<String, String> valueMap = new HashMap<>();
        valueMap.put("value", tx.getHash().getDigestHex());
        return Result.getSuccess().setData(valueMap).toRpcClientResult();
    }


    @GET
    @Path("/redPunish/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
```

```
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = Boolean.class)
    })
    public RpcClientResult getRedPunishByAddress(@ApiParam(name = "address", value = "",
required = true)
                                         @PathParam("address") String address) {
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        List<PunishLogPo> list =
PocConsensusContext.getChainManager().getMasterChain().getChain().getRedPunishList();
        boolean rs = false;
        for (PunishLogPo po : list) {
            if (Arrays.equals(AddressTool.getAddress(address), po.getAddress())) {
                rs = true;
                break;
            }
        }
        return Result.getSuccess().setData(rs).toRpcClientResult();
    }

    @GET
    @Path("/multiAccount/Agent/fee")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "get the fee of create agent! ()", notes = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult getCreateMultiAgentFee(
            @BeanParam() GetCreateAgentFeeForm form) throws Exception {
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getAgentAddress(), "agent address can not be null");
        AssertUtil.canNotEmpty(form.getCommissionRate(), "commission rate can not be null");
        AssertUtil.canNotEmpty(form.getDeposit(), "deposit can not be null");
        AssertUtil.canNotEmpty(form.getPackingAddress(), "packing address can not be null");
        if (StringUtils.isBlank(form.getRewardAddress())) {
            form.setRewardAddress(form.getAgentAddress());
        }
        CreateAgentTransaction tx = new CreateAgentTransaction();
        tx.setTime(TimeService.currentTimeMillis());
        Agent agent = new Agent();
        agent.setAgentAddress(AddressTool.getAddress(form.getAgentAddress()));
```

```
        agent.setPackingAddress(AddressTool.getAddress(form.getPackingAddress()));
        if (StringUtils.isBlank(form.getRewardAddress())) {
            agent.setRewardAddress(agent.getAgentAddress());
        } else {
            agent.setRewardAddress(AddressTool.getAddress(form.getRewardAddress()));
        }
        agent.setDeposit(Na.valueOf(form.getDeposit()));
        agent.setCommissionRate(form.getCommissionRate());
        tx.setTxData(agent);
        Result<MultiSigAccount> sigAccountResult =
accountService.getMultiSigAccount(form.getAgentAddress());
        MultiSigAccount multiSigAccount = sigAccountResult.getData();
        Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
        if (redeemScript == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        if (agent.getAgentAddress()[2] == 3) {
            Script scriptPubkey = SignatureUtil.createOutputScript(agent.getAgentAddress());
            toList.add(new Coin(scriptPubkey.getProgram(), agent.getDeposit(), -1));
        } else {
            toList.add(new Coin(agent.getAgentAddress(), agent.getDeposit(), -1));
        }
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        CoinDataResult result = accountLedgerService.getMutilCoinData(agent.getAgentAddress(),
agent.getDeposit(), tx.size(), TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        tx.getCoinData().setFrom(result.getCoinList());
        if (null != result.getChange()) {
            tx.getCoinData().getTo().add(result.getChange());
        }
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        //m*+
        int scriptSignLenth = redeemScript.getProgram().length + ((int) multiSigAccount.getM()) * 72;
        Result rs =
accountLedgerService.getMultiMaxAmountOfOnce(AddressTool.getAddress(form.getAgentAddres
s()), tx, TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES, scriptSignLenth);
        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);
        rs.setData(map);
        return Result.getSuccess().setData(rs).toRpcClientResult();
    }

    @GET
    @Path("/multiAccount/deposit/fee")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "get the fee of create agent! ", notes = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult getMultiDepositFee(@BeanParam() GetDepositFeeForm form) throws
Exception {
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getAddress(), "address can not be null");
        AssertUtil.canNotEmpty(form.getAgentHash(), "agent hash can not be null");
        AssertUtil.canNotEmpty(form.getDeposit(), "deposit can not be null");
        DepositTransaction tx = new DepositTransaction();
        Deposit deposit = new Deposit();
        deposit.setAddress(AddressTool.getAddress(form.getAddress()));
        deposit.setAgentHash(NulsDigestData.fromDigestHex(form.getAgentHash()));
        deposit.setDeposit(Na.valueOf(form.getDeposit()));
        tx.setTxData(deposit);
        Result<MultiSigAccount> sigAccountResult =
accountService.getMultiSigAccount(form.getAddress());
        MultiSigAccount multiSigAccount = sigAccountResult.getData();
        Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
        if (redeemScript == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        toList.add(new Coin(deposit.getAddress(), deposit.getDeposit(), -1));
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        CoinDataResult result = accountLedgerService.getCoinData(deposit.getAddress(),
deposit.getDeposit(), tx.size(), TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        tx.getCoinData().setFrom(result.getCoinList());
        if (null != result.getChange()) {
```

```
            tx.getCoinData().getTo().add(result.getChange());
        }
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        //m*+
        int scriptSignLenth = redeemScript.getProgram().length + ((int) multiSigAccount.getM()) * 72;
        Result rs =
accountLedgerService.getMultiMaxAmountOfOnce(AddressTool.getAddress(form.getAddress()),
tx, TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES, scriptSignLenth);
        Map<String, Long> map = new HashMap<>();
        Long maxAmount = null;
        if (rs.isSuccess()) {
            maxAmount = ((Na) rs.getData()).getValue();
        }
        map.put("fee", fee.getValue());
        map.put("maxAmount", maxAmount);
        rs.setData(map);
        return Result.getSuccess().setData(rs).toRpcClientResult();
    }

    @GET
    @Path("/multiAccount/agent/stop/fee")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "get the fee of stop agent! ", notes = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult getStopMultiAgentFee(@ApiParam(name = "address", value = "",
required = true)
                                                @QueryParam("address") String address) throws NulsException,
IOException {
        AssertUtil.canNotEmpty(address, "address can not be null");
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }

        StopAgentTransaction tx = new StopAgentTransaction();
        StopAgent stopAgent = new StopAgent();
        stopAgent.setAddress(AddressTool.getAddress(address));
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        Agent agent = null;
        for (Agent a : agentList) {
```

```
            if (a.getDelHeight() > 0) {
                continue;
            }
            if (Arrays.equals(a.getAgentAddress(), stopAgent.getAddress())) {
                agent = a;
                break;
            }
        }
        if (agent == null || agent.getDelHeight() > 0) {
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }
        NulsDigestData createTxHash = agent.getTxHash();
        stopAgent.setCreateTxHash(createTxHash);
        tx.setTxData(stopAgent);
        CoinData coinData = ConsensusTool.getStopAgentCoinData(agent,
TimeService.currentTimeMillis() + PocConsensusConstant.STOP_AGENT_LOCK_TIME);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        coinData.getTo().get(0).setNa(coinData.getTo().get(0).getNa().subtract(fee));
        Na resultFee = TransactionFeeCalculator.getMaxFee(tx.size());
        Result rs =
accountLedgerService.getMultiMaxAmountOfOnce(AddressTool.getAddress(address), tx,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES, 0);
        Map<String, Long> map = new HashMap<>();
        Long maxAmount = null;
        if (rs.isSuccess()) {
            maxAmount = ((Na) rs.getData()).getValue();
        }
        map.put("fee", resultFee.getValue());
        map.put("maxAmount", maxAmount);
        rs.setData(map);
        return Result.getSuccess().setData(rs).toRpcClientResult();
    }


    @POST
    @Path("/multiAccount/createMultiAgent")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "Create an agent for consensus! () [3.6.3]", notes = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
```

```
    })
    public RpcClientResult createMutilAgent(@ApiParam(name = "form", value = "", required =
true)
                                            CreateMultiAgentForm form) throws Exception {
        if (NulsContext.MAIN_NET_VERSION <= 1) {
            return Result.getFailed(KernelErrorCode.VERSION_TOO_LOW).toRpcClientResult();
        }
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getAgentAddress(), "agent address can not be null");
        AssertUtil.canNotEmpty(form.getSignAddress(), "agent address can not be null");
        AssertUtil.canNotEmpty(form.getCommissionRate(), "commission rate can not be null");
        AssertUtil.canNotEmpty(form.getDeposit(), "deposit can not be null");
        AssertUtil.canNotEmpty(form.getPackingAddress(), "packing address can not be null");

        if (!AddressTool.isPackingAddress(form.getPackingAddress()) ||
!AddressTool.validAddress(form.getAgentAddress()) ||
!AddressTool.validAddress(form.getSignAddress())) {
            throw new NulsRuntimeException(AccountErrorCode.ADDRESS_ERROR);
        }
        Account account = accountService.getAccount(form.getSignAddress()).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }
        Result<MultiSigAccount> sigAccountResult =
accountService.getMultiSigAccount(form.getAgentAddress());
        MultiSigAccount multiSigAccount = sigAccountResult.getData();
        Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
        if (redeemScript == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        CreateAgentTransaction tx = new CreateAgentTransaction();
        tx.setTime(TimeService.currentTimeMillis());
        Agent agent = new Agent();
        agent.setAgentAddress(AddressTool.getAddress(form.getAgentAddress()));
        agent.setPackingAddress(AddressTool.getAddress(form.getPackingAddress()));
```

```
        if (StringUtils.isBlank(form.getRewardAddress())) {
            agent.setRewardAddress(agent.getAgentAddress());
        } else {
            agent.setRewardAddress(AddressTool.getAddress(form.getRewardAddress()));
        }
        TransactionSignature transactionSignature = new TransactionSignature();
        List<Script> scripts = new ArrayList<>();
        agent.setDeposit(Na.valueOf(form.getDeposit()));
        agent.setCommissionRate(form.getCommissionRate());
        tx.setTxData(agent);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        if (agent.getAgentAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
            Script scriptPubkey = SignatureUtil.createOutputScript(agent.getAgentAddress());
            toList.add(new Coin(scriptPubkey.getProgram(), agent.getDeposit(),
PocConsensusConstant.CONSENSUS_LOCK_TIME));
        } else {
            toList.add(new Coin(agent.getAgentAddress(), agent.getDeposit(),
PocConsensusConstant.CONSENSUS_LOCK_TIME));
        }
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        //m*+
        int scriptSignLenth = redeemScript.getProgram().length + ((int) multiSigAccount.getM()) * 72;
        CoinDataResult result = accountLedgerService.getMutilCoinData(agent.getAgentAddress(),
agent.getDeposit(), tx.size() + scriptSignLenth,
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        if (null != result) {
            if (result.isEnough()) {
                tx.getCoinData().setFrom(result.getCoinList());
                if (null != result.getChange()) {
                    tx.getCoinData().getTo().add(result.getChange());
                }
            } else {
                return
Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE).toRpcClientResult();
            }
        }
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        //
        scripts.add(redeemScript);
        transactionSignature.setScripts(scripts);
```

```
        Result finalResult = accountLedgerService.txMultiProcess(tx, transactionSignature, account,
form.getPassword());
        if (finalResult.isSuccess()) {
            Map<String, String> valueMap = new HashMap<>();
            valueMap.put("txData", (String) finalResult.getData());
            return Result.getSuccess().setData(valueMap).toRpcClientResult();
        }
        return finalResult.toRpcClientResult();
    }

    @POST
    @Path("/multiAccount/createMultiDeposit")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "deposit nuls to a bank!  ", notes = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult createMutilDeposit(@ApiParam(name = "form", value = "", required =
true)
                                              CreateMultiDepositForm form) throws Exception {
        if (NulsContext.MAIN_NET_VERSION <= 1) {
            return Result.getFailed(KernelErrorCode.VERSION_TOO_LOW).toRpcClientResult();
        }
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getDeposit());
        AssertUtil.canNotEmpty(form.getAgentHash());
        AssertUtil.canNotEmpty(form.getAddress());
        AssertUtil.canNotEmpty(form.getSignAddress());
        if (form == null) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        if (!NulsDigestData.validHash(form.getAgentHash())) {
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }
        if (!AddressTool.validAddress(form.getSignAddress()) ||
!AddressTool.validAddress(form.getAddress())) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        Account account = accountService.getAccount(form.getSignAddress()).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
```

```
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }

        Result<MultiSigAccount> sigAccountResult =
accountService.getMultiSigAccount(form.getAddress());
        MultiSigAccount multiSigAccount = sigAccountResult.getData();
        Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
        if (redeemScript == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }

        DepositTransaction tx = new DepositTransaction();
        TransactionSignature transactionSignature = new TransactionSignature();
        List<Script> scripts = new ArrayList<>();
        Deposit deposit = new Deposit();
        deposit.setAddress(AddressTool.getAddress(form.getAddress()));
        deposit.setAgentHash(NulsDigestData.fromDigestHex(form.getAgentHash()));
        deposit.setDeposit(Na.valueOf(form.getDeposit()));
        tx.setTxData(deposit);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        //AddressTool.getAddress(addr)
        if (deposit.getAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
            Script scriptPubkey = SignatureUtil.createOutputScript(deposit.getAddress());
            toList.add(new Coin(scriptPubkey.getProgram(), deposit.getDeposit(),
PocConsensusConstant.CONSENSUS_LOCK_TIME));
        } else {
            toList.add(new Coin(deposit.getAddress(), deposit.getDeposit(),
PocConsensusConstant.CONSENSUS_LOCK_TIME));
        }
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        //m*+
        int scriptSignLenth = redeemScript.getProgram().length + ((int) multiSigAccount.getM()) * 72;
        CoinDataResult result = accountLedgerService.getMutilCoinData(deposit.getAddress(),
deposit.getDeposit(), tx.size() + scriptSignLenth,
```

```
TransactionFeeCalculator.OTHER_PRECE_PRE_1024_BYTES);
        if (null != result) {
            if (result.isEnough()) {
                tx.getCoinData().setFrom(result.getCoinList());
                if (null != result.getChange()) {
                    tx.getCoinData().getTo().add(result.getChange());
                }
            } else {
                return
Result.getFailed(TransactionErrorCode.INSUFFICIENT_BALANCE).toRpcClientResult();
            }
        }
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        //
        scripts.add(redeemScript);
        transactionSignature.setScripts(scripts);
        Result resultData = accountLedgerService.txMultiProcess(tx, transactionSignature, account,
form.getPassword());
        if (resultData.isSuccess()) {
            Map<String, String> valueMap = new HashMap<>();
            valueMap.put("txData", (String) resultData.getData());
            return Result.getSuccess().setData(valueMap).toRpcClientResult();
        }
        return resultData.toRpcClientResult();
    }

    @POST
    @Path("/multiAccount/agent/stopMultiAgent")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "", notes = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult createStopMutilAgent(@ApiParam(name = "form", value = "", required =
true)
                                                 CreateStopMultiAgentForm form) throws Exception {

        if (NulsContext.MAIN_NET_VERSION <= 1) {
            return Result.getFailed(KernelErrorCode.VERSION_TOO_LOW).toRpcClientResult();
        }
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getAddress());
```

```
        AssertUtil.canNotEmpty(form.getSignAddress());
        if (!AddressTool.validAddress(form.getAddress()) ||
!AddressTool.validAddress(form.getSignAddress())) {
            throw new NulsRuntimeException(KernelErrorCode.PARAMETER_ERROR);
        }
        Account account = accountService.getAccount(form.getSignAddress()).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }
        List<Agent> agentList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getAgentList();
        Agent agent = null;
        for (Agent p : agentList) {
            if (p.getDelHeight() > 0) {
                continue;
            }
            if (Arrays.equals(p.getAgentAddress(), AddressTool.getAddress(form.getAddress()))) {
                agent = p;
                break;
            }
        }
        if (agent == null || agent.getDelHeight() > 0) {
            return
Result.getFailed(PocConsensusErrorCode.AGENT_NOT_EXIST).toRpcClientResult();
        }

        Result<MultiSigAccount> sigAccountResult =
accountService.getMultiSigAccount(form.getAddress());
        MultiSigAccount multiSigAccount = sigAccountResult.getData();
        Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
        if (redeemScript == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }

        //
```

```
        StopAgentTransaction tx = new StopAgentTransaction();
        TransactionSignature transactionSignature = new TransactionSignature();
        List<Script> scripts = new ArrayList<>();
        StopAgent stopAgent = new StopAgent();
        stopAgent.setAddress(agent.getAgentAddress());
        stopAgent.setCreateTxHash(agent.getTxHash());
        tx.setTime(TimeService.currentTimeMillis());
        tx.setTxData(stopAgent);
        CoinData coinData = ConsensusTool.getStopAgentCoinData(agent,
TimeService.currentTimeMillis() + PocConsensusConstant.STOP_AGENT_LOCK_TIME, null);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        coinData.getTo().get(0).setNa(coinData.getTo().get(0).getNa().subtract(fee));
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        //
        scripts.add(redeemScript);
        transactionSignature.setScripts(scripts);
        //
        Result resultData = accountLedgerService.txMultiProcess(tx, transactionSignature, account,
form.getPassword());

        if (resultData.isSuccess()) {
            Map<String, String> valueMap = new HashMap<>();
            valueMap.put("txData", (String) resultData.getData());
            return Result.getSuccess().setData(valueMap).toRpcClientResult();
        }

        return resultData.toRpcClientResult();
    }

    @POST
    @Path("/multiAccount/mutilWithdraw")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "", notes = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult createWithdrawMutil(@ApiParam(name = "form", value = "", required =
true)
                                               CreateMultiWithdrawForm form) throws Exception {
        AssertUtil.canNotEmpty(form);
        AssertUtil.canNotEmpty(form.getTxHash());
```

```
        AssertUtil.canNotEmpty(form.getAddress());
        AssertUtil.canNotEmpty(form.getSignAddress());
        if (!NulsDigestData.validHash(form.getTxHash())) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        if (!AddressTool.validAddress(form.getAddress()) ||
!AddressTool.validAddress(form.getSignAddress())) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        Account account = accountService.getAccount(form.getSignAddress()).getData();
        if (null == account) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        if (account.isEncrypted() && account.isLocked()) {
            AssertUtil.canNotEmpty(form.getPassword(), "password is wrong");
            if (!account.validatePassword(form.getPassword())) {
                return
Result.getFailed(AccountErrorCode.PASSWORD_IS_WRONG).toRpcClientResult();
            }
        }

        Result<MultiSigAccount> sigAccountResult =
accountService.getMultiSigAccount(form.getAddress());
        MultiSigAccount multiSigAccount = sigAccountResult.getData();
        Script redeemScript = accountLedgerService.getRedeemScript(multiSigAccount);
        if (redeemScript == null) {
            return Result.getFailed(AccountErrorCode.ACCOUNT_NOT_EXIST).toRpcClientResult();
        }
        CancelDepositTransaction tx = new CancelDepositTransaction();
        TransactionSignature transactionSignature = new TransactionSignature();
        List<Script> scripts = new ArrayList<>();
        CancelDeposit cancelDeposit = new CancelDeposit();
        NulsDigestData hash = NulsDigestData.fromDigestHex(form.getTxHash());
        DepositTransaction depositTransaction = null;
        try {
            depositTransaction = (DepositTransaction) ledgerService.getTx(hash);
        } catch (Exception e) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        if (null == depositTransaction) {
            return Result.getFailed(TransactionErrorCode.TX_NOT_EXIST).toRpcClientResult();
        }
```

```
        // Create unlock script
        cancelDeposit.setAddress(AddressTool.getAddress(form.getAddress()));
        cancelDeposit.setJoinTxHash(hash);
        tx.setTxData(cancelDeposit);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        if (cancelDeposit.getAddress()[2] == NulsContext.P2SH_ADDRESS_TYPE) {
            Script scriptPubkey = SignatureUtil.createOutputScript(cancelDeposit.getAddress());
            toList.add(new Coin(scriptPubkey.getProgram(),
depositTransaction.getTxData().getDeposit(),
PocConsensusConstant.CONSENSUS_LOCK_TIME));
        } else {
            toList.add(new Coin(cancelDeposit.getAddress(),
depositTransaction.getTxData().getDeposit(), 0));
        }
        coinData.setTo(toList);
        List<Coin> fromList = new ArrayList<>();
        for (int index = 0; index < depositTransaction.getCoinData().getTo().size(); index++) {
            Coin coin = depositTransaction.getCoinData().getTo().get(index);
            if (coin.getLockTime() == -1L &&
coin.getNa().equals(depositTransaction.getTxData().getDeposit())) {
                coin.setOwner(ArraysTool.concatenate(hash.serialize(), new VarInt(index).encode()));
                fromList.add(coin);
                break;
            }
        }
        if (fromList.isEmpty()) {
            return Result.getFailed(KernelErrorCode.DATA_ERROR).toRpcClientResult();
        }
        coinData.setFrom(fromList);
        tx.setCoinData(coinData);
        Na fee = TransactionFeeCalculator.getMaxFee(tx.size());
        coinData.getTo().get(0).setNa(coinData.getTo().get(0).getNa().subtract(fee));
        tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        //
        scripts.add(redeemScript);
        transactionSignature.setScripts(scripts);
        Result resultData = accountLedgerService.txMultiProcess(tx, transactionSignature, account,
form.getPassword());
        if (resultData.isSuccess()) {
            Map<String, String> valueMap = new HashMap<>();
            valueMap.put("txData", (String) resultData.getData());
```

```
            return Result.getSuccess().setData(valueMap).toRpcClientResult();
        }
        return resultData.toRpcClientResult();
    }


    @GET
    @Path("/punish/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "", notes = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = String.class)
    })
    public RpcClientResult getPunishList(@ApiParam(name = "address", value = "", required =
true)
                                         @PathParam("address") String address,
                                         @ApiParam(name = "type", value = ":yellow:0,red:1", required = true)
                                         @QueryParam("type") int type) {
        byte[] addressByte = AddressTool.getAddress(address);
        if (!AddressTool.validNormalAddress(addressByte)) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        List<PunishLogPo> punishList = null;
        if (type == 0) {
            punishList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getYellowPunishList();
        } else if (1 == type) {
            punishList =
PocConsensusContext.getChainManager().getMasterChain().getChain().getRedPunishList();
        } else {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        List<PunishLogDTO> list = new ArrayList<>();
        for (PunishLogPo po : punishList) {
            if (!ArraysTool.arrayEquals(po.getAddress(), addressByte)) {
                continue;
            }
            list.add(new PunishLogDTO(po));
        }
        Result result = Result.getSuccess();
        result.setData(list);
        return result.toRpcClientResult();
```

```
    }
}

139:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
rpc\src\main\java\io\nuls\consensus\poc\rpc\utils\AgentComparator.java
 *
 */

package io.nuls.consensus.poc.rpc.utils;

import io.nuls.consensus.poc.protocol.entity.Agent;

import java.util.Comparator;

/**
 * @author Niels
 */
public class AgentComparator implements Comparator<Agent> {

    public static final int DEPOSIT = 0;
    public static final int COMMISSION_RATE = 1;
    public static final int CREDIT_VALUE = 2;
    public static final int DEPOSITABLE = 3;
    public static final int COMPREHENSIVE = 4;

    private static final AgentComparator[] INSTANCE_ARRAY = new AgentComparator[]{
            new AgentComparator(DEPOSIT),
            new AgentComparator(COMMISSION_RATE),
            new AgentComparator(CREDIT_VALUE),
            new AgentComparator(DEPOSITABLE),
            new AgentComparator(COMPREHENSIVE)
    };
    private final int sortType;

    public static AgentComparator getInstance(int sortType) {
        switch (sortType) {
            case DEPOSIT:
                return INSTANCE_ARRAY[DEPOSIT];
            case COMMISSION_RATE:
                return INSTANCE_ARRAY[COMMISSION_RATE];
            case CREDIT_VALUE:
                return INSTANCE_ARRAY[CREDIT_VALUE];
```

```
        case DEPOSITABLE:
            return INSTANCE_ARRAY[DEPOSITABLE];
        case COMPREHENSIVE:
            return INSTANCE_ARRAY[COMPREHENSIVE];
        default:
            return INSTANCE_ARRAY[CREDIT_VALUE];
    }
}

private AgentComparator(int sortType) {
    this.sortType = sortType;
}

@Override
public int compare(Agent o1, Agent o2) {
    switch (sortType) {
        case DEPOSIT: {
            if (o1.getDeposit().getValue() < o2.getDeposit().getValue()) {
                return 1;
            } else if (o1.getDeposit().getValue() == o2.getDeposit().getValue()) {
                return 0;
            }
            return -1;
        }
        case COMMISSION_RATE: {
            if (o1.getCommissionRate() < o2.getCommissionRate()) {
                return -1;
            } else if (o1.getCommissionRate() == o2.getCommissionRate()) {
                return 0;
            }
            return 1;
        }
        case CREDIT_VALUE: {
            if (o2.getCreditVal() < o1.getCreditVal()) {
                return -1;
            } else if (o2.getCreditVal() == o1.getCreditVal()) {
                return 0;
            }
            return 1;
        }
        case DEPOSITABLE: {
            if (o2.getTotalDeposit() < o1.getTotalDeposit()) {
```

```
                return -1;
            } else if (o2.getTotalDeposit() == o1.getTotalDeposit()) {
                return 0;
            }
            return 1;
        }
        case COMPREHENSIVE: {
            //
            //First judge whether the trust can be entrusted, then judge the credit value, then judge the amount of trust
            if (o1.canDeposit() && !o2.canDeposit()) {
                return -1;
            } else if (o2.canDeposit() == o1.canDeposit()) {
                if (o2.getCreditVal() < o1.getCreditVal()) {
                    return -1;
                } else if (o2.getCreditVal() == o1.getCreditVal()) {
                    if (o2.getAvailableDepositAmount() < o1.getAvailableDepositAmount()) {
                        return -1;
                    } else if (o2.getAvailableDepositAmount() == o1.getAvailableDepositAmount()) {
                        return 0;
                    }
                    return 1;
                }
                return 1;
            }
            return 1;
        }
        default: {
            if (o2.getCreditVal() < o1.getCreditVal()) {
                return -1;
            } else if (o2.getCreditVal() == o1.getCreditVal()) {
                return 0;
            }
            return 1;
        }
    }
  }
}

140:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-rpc\src\test\java\CreateAgentTest.java
 */
```

```
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;
import org.junit.Test;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

/**
 * @author: Niels Wang
 */
public class CreateAgentTest {

    @Test
    public void test() {
        List<String> agentAddressList = getAgentAddressList();
        List<String> packingAddressList = getPackingAddressList();
        for (int x = 0; x < agentAddressList.size(); x++) {
            String address = agentAddressList.get(x);
            String toAddress = packingAddressList.get(x);

            String param = "{\"agentAddress\": \"" + address + "\",\"packingAddress\": \"" + toAddress +
"\", \"rewardAddress\": \"" + address + "\", \"commissionRate\": 10,\"deposit\": 2000000000000,
\"password\": \"\"}";

            String url = "http://127.0.0.1:8001/api/consensus/agent";

            String res = post(url, param, "utf-8");

            System.out.println(res);
        }
    }

    public List<String> getAgentAddressList() {
        List<String> list = new ArrayList<>();

        list.add("2Cc2gqxcsJyM4tbJxqaHTj22HESAeXD");
        list.add("2CYfymtFuVCuYrQLzJ3gBvdZptvz4zj");
        list.add("2CZXYx5qszeCcqYXCwBbvkXZXiK1gk4");
```

```
list.add("2CiHFDCksjv5aCJkz3fSLqtBLucjmMH");
list.add("2CXRy7Utqpd1RNuhYEVkLYTcRv5JDLS");
list.add("2CWdSPCo4t1PYHsRCMqs9Vfdy1hEYn5");
list.add("2CXwBnL1TTqJ7KdjrEMvTsJoFQw6ui6");
list.add("2CY6uMkQXDyzS8qYBBCwCRFNw2QMjLE");
list.add("2CcLYrTCre75a2gzcToNYnYA8VgXmjX");
list.add("2CZXSrhjHp9vKigCFv2SveS4bo1DHfc");
list.add("2CdxbVs9j8DWKjEztvLafWo8myqqm1k");
list.add("2CjQ8Mq6Es5gsDhkDbry4pwXXc5DcCW");
list.add("2Cc8Q5rSFHcDSoQfUEqYcG5HSHRhfzm");
list.add("2CfwtEqXuhLFWg2EtV6gxqdpmLpfoMJ");
list.add("2CbJFyiDxkdmTVAtJFU5cqrSNcuw355");
list.add("2CdKrRBh7bkTcj7TyLJpNS83zi1HKzr");
list.add("2CirE7j1m2LxeweeH3EiZZZjHcGHCT4");
list.add("2Cef5TNpVdzsAveDUXUZCT96KEvET5z");
list.add("2CeH8wjuLk2fVdQD3Q4GyXXEEBxT3Sg");
list.add("2CgA1Y795Yv8PSo7QhdSHUiXkGiofNs");
list.add("2CaMJfVt7ouBfGkrAT3ud8pprUymUJb");
list.add("2CXkbp52hTfx4YKFp2uuXUb2p6uSTc1");
list.add("2CdD7nHtapywL8sdu43c72YpHaKHrKr");
list.add("2CfvjcESYnkq8fZZFLtJXSjNa2ehUox");
list.add("2Cb4oMyPSybkDyPXWsqnpGcoPoAZLmY");
list.add("2CZ22WuJpByGXKE9Xhw3sXYFK45zg1X");
list.add("2CbHbC2QEK87hddZAFcLXegwCg5Pq1J");
list.add("2CWx9PZb4uVLAJX1xMjS4iAndYe3mcG");
list.add("2Cbp16QYb7tzpHL9MvA1cQbPCkfSbP4");
list.add("2CcD8oStg6cYCj9SWE9y39j2KNJoGr8");
list.add("2CivRco72vWqHgPTn2meNuzScP2aQXL");
list.add("2CiTdr35bjC35ERDFDM2tL6qBxKMAtz");
list.add("2CV5QF5r5XDzj9WWdmHPJrc8xm5JYAB");
list.add("2CW4fFepEVpp7DXy9G4jPFPvNMugL7Q");
list.add("2CkBNLW8Y2X7vdX3hxFBHR3DqsFuWGU");
list.add("2CfiBnL55wsphYjjPVJHqFiQH7M3oSQ");
list.add("2CZuUGpZgvRzxzZ4TZNbrp9PrXs51Tn");
list.add("2CiCmGsSsnfb8Kc8zHafmkG6588yx5u");
list.add("2CYHLVWGLBxGkijXq2N8JnHRrTLysjQ");
list.add("2CWWtH7uEA836Mvbd9VzMVZdRiRrdGy");
list.add("2CeuVcXeibNjDjungWABCRzDa9qtcru");
list.add("2Cg5j3A3ejCxYsmfUwXfVQePVhFhwHg");
list.add("2CegCbx9jWCncSHK7M3Jeu4fWu5ZRs5");
list.add("2CeasNhdfxvVkyCL7rKNK7rCFprUQmG");
list.add("2Cay675epEHA8YFwYmPEgrQRAhMSj4h");
```

```
list.add("2CiUZjo9yzgEDa81KSZaiGHqfD5UmLt");
list.add("2CheTVzJ2L37h1w8MRvzafostSzJGsT");
list.add("2CfkLhjEXkPGQtm3x6oU1wEnjpHYkPu");
list.add("2Ch5ai5evu94E6Ew2SBiQWaYXAhuMUz");
list.add("2CaCfVbPGhd2VAiiXTpLxUfJPajrxci");
list.add("2ChgP29U9SKQFezCen4LELXery6bz1Y");
list.add("2CkJPfPa2gogLTK5acgrRzEkPHjrDXG");
list.add("2CZokSNjXUmTKY4yzvtxTQvSBM7iJFh");
list.add("2CjgMy8zTJUzCWFPDgQR61xfDWBuZLF");
list.add("2CcTfQgAbMJ6piTnYogNQCC9K1C6mgs");
list.add("2CjEStoSAngTSdeR7VYbYy9Gj2eTQ9i");
list.add("2CiB66gXTunGfpvApQavJBKM8oNWo48");
list.add("2CfjoHuqH9SoUZRW7m19HLFjppaeuX4");
list.add("2CWDPkrGKvqDY9GtW4GrSH5j8TAg5bB");
list.add("2CcFe6QLTHLzQvYK1mKi3DqGmyjPXro");
list.add("2CZ9ekWTuVY41ycJJdn2LDqdTpzNG6r");
list.add("2Cdieo8MQJL7KMQ51hoSynDJfSR2tk5");
list.add("2CWVZ9m1kbFKvcB2UwgQGJTGpWTGjVp");
list.add("2CjLaiMjPmstrAZiV4WnxnPCt7UYAUy");
list.add("2CdA5WvEmragTZKzmH8QW7WBNvKFis3");
list.add("2Cg9wn7nuUH9WbKoKfTDsXvWsPHfhBu");
list.add("2CXRbbbRumpyLbrC8SX96ugqK1WygFr");
list.add("2CbM3M1KeA3nJHwrNJxvDjf1N3V6Mpm");
list.add("2CYKSuDFiNrA387LAF526eUeUKw67vC");
list.add("2CVAxGcxCZv6rwq2v7vUPpe931V3muf");
list.add("2Cj61FNPY9B8wex1kW95RgApMS4MmTg");
list.add("2CdPSMVbEnSHErVAxfstxmQctBHDaaQ");
list.add("2Cc5u8feD9ifQErApAvj7aLaUeg4EQm");
list.add("2CeEMobjcDf3irnapm2xqdSzPQg724z");
list.add("2CcFx5iv17Tgymp33HP5fdZeMgL5nDD");
list.add("2CfzQntRknL5brgKDnpiz1Ayp8zsLYC");
list.add("2Cbr9JrJvCKZPFo2XPvoDa1aTm6pK81");
list.add("2ChczYCL6LW3cRj8PtCgmJVoXCrzRJP");
list.add("2Ce1EyCijdkSKHPSMRu7WXiw8E3Yuhp");
list.add("2CeXu6QPpQYe9RTZSkkucVFCtMwNRss");
list.add("2Cba9QaakwYX6eHtQVZgmobBQ7fuzpi");
list.add("2CZJNsbzzBHsmHvy6MEgxJM4MLoATve");
list.add("2Ca4f5TN2ZTZY1gWgnmcgwMqXcdgMb3");
list.add("2Ci6FJsZVzjqN8hyiWpVLh3MbpMfi57");
list.add("2Ce29AuYXgR1c7UALYxRAp8ZXLJuHU5");
list.add("2CXzL7yuWFrGD3aJae9tGEE5Hgzy66C");
list.add("2CfMavgXm2zumM7Jf9oNVdZT1erzvHY");
```

```
    list.add("2Cczq6oKSVcaFXkLzds2RM7cBNb5nW9");
    list.add("2CWi6SL7QZMuPEEsRpXrWLirsAbR4aC");
    list.add("2CkSBB7ns5Q3yFUUPfkoWcnAUwcDVwv");
    list.add("2CZPgN5VU81WbhZJrCBXtgT3PXMkUAi");
    list.add("2Cdt9yhbXpCDLuU2gQaQH5M5b3umM3a");
    list.add("2CjuDKiWy9QpaT4VLGY8rGAAGSuwpfv");
    list.add("2CYcaZYkgcpMKHX7TtLjpkyRotbs55R");
    list.add("2Cext6pmhnHoN41Hf3XuZQxmt1cPW8B");
    list.add("2CVyo48t6bqtktuTXYLG41vbn2ZdhbR");
    list.add("2CbzPf8PkZkTTrViAqXp598kfBt24id");
    list.add("2CaCVQjNQFuJCofmcqyPK6GnWJwjimK");
    list.add("2CgTcje5EePMoHsLkTsfVG3uy2hZ61v");
    list.add("2Cc4fBADknrbKLLK7Qkg35W6GH3ur7E");
    return list;
}

public List<String> getPackingAddressList() {
    List<String> list = new ArrayList<>();
    list.add("2Ck4ZpGavtaXtgdBx8NTTZyHpNGoWer");
    list.add("2CZYnwNfMba7inw22p74QSaWQ4QRfD5");
    list.add("2CcHvoKqmUUJB6Fi7ykWg1buzT7fKez");
    list.add("2CVtTnX17jw4gPGkCbkmdqom97JvRFW");
    list.add("2Cc22xksbES2PQko37tPyoxho8RUEnW");
    list.add("2Cj7JT1FW1HHnt8VFLutoWAmG9wWcZ7");
    list.add("2CZde4pbCQwLZ4jU5DseptXoJSPG3zc");
    list.add("2CffA9fq7yVFaovFH8jJh5toV5d3X5K");
    list.add("2CYF21KHdXBXVwQg8PJTbSNLjZpht7g");
    list.add("2CgyVBoRmgNquHkxJTi3vzFWfLto5c9");
    list.add("2CdAeu8Xn4v1NEra1CZsU3xwccDQXcY");
    list.add("2CezK5q6or6z6qoSZCKnARckSLwvWW5");
    list.add("2CbZupjHbjpWxAr6yu6Ruoqu3r4gYNa");
    list.add("2CgT4swe1XHEy3VaqqTCKZUNLM1FTuN");
    list.add("2CisF4qfWZVbZqvsb5xdwCb3z6ELfDB");
    list.add("2CkKdr8yaN1fwDiEti4ezScFbo1RhqV");
    list.add("2CiDg37ubkPWX2R8DRBpNnWpRgFSF4E");
    list.add("2CW4VJyzpw8ZwE8QdFDiH47NVSUj5UG");
    list.add("2CbEQ1eB3n16sBCp9tKddW1cAyWxvfH");
    list.add("2CaafwmH2MuTxjUaPbE92W3Uh3bxH6v");
    list.add("2CWoJCkqw6yGAYvAjsgWTEhS2GK9F5Z");
    list.add("2CbNvcKS3JUeWv6bdiWm9MhMp3be11d");
    list.add("2CXMFnaAHKVA2THQiqBxvj2MaYr9uaM");
    list.add("2CjJTpBs16jpfV9sR2NAgkT8XXzNPug");
```

```
list.add("2Ca2Ke8FGWQHrnVP8H5BYi2XMgkJWD5");
list.add("2CiHwohP7Fb9rgKPvJ546nEyBwh72Pw");
list.add("2ChhfPDBAuXBQ5f2G5dRqZmwykvTgPL");
list.add("2CbiWEkFcGyjRFW4VDNmAVt1J5TY315");
list.add("2CiEHcrAygtoq7txuCZQ6c93NARz4oD");
list.add("2CdoJ5TDQj2JTveNHwefiRSAvmqYfXt");
list.add("2CY5YZpQkfBCpY3qjoAwL5tPmTs7id3");
list.add("2ChwoeVtnrzJ7MhKeUVFoGb7kviizcN");
list.add("2CaJPyNAwzW6PfabztHtjwTtYPLdF7A");
list.add("2CbB7LbZL9F6ZBCfRZfZM9Xe92iDaEf");
list.add("2Cg6yNqc4QUxorNRtnt2DW7zdfRL6Ef");
list.add("2CeGcWE2iMhGy6MAMKvmoDAkbxiWsDb");
list.add("2Ceh7WJmBwQEJmn86vr21fYPpBJe4Vk");
list.add("2CfccNBRkWbFMHcWEBo6nkJjJ3o8L1h");
list.add("2CbEPiz7xigbDQwgFD89t2yUTbPWxLR");
list.add("2CfCUVXNBWzUvRFGSUmhMyr5ebopkue");
list.add("2Cdcj2trBX7j7fLLBmHSMPX4qXonwPz");
list.add("2CWeYLSWW5C3gawfYqc7Gd5KwJVyQbQ");
list.add("2CaqGzmQVR6crzWjXRLJASRLE6GswzC");
list.add("2CgJvCSoekj2CPADL1Um4Fdfdf5PJw2");
list.add("2CeW8toTYsvpF8RWJASZXCiqUu5jyCD");
list.add("2CVeBwK6s21ABdsstRcfB7NLBaZ1ipA");
list.add("2Cdvo8LJxWYE7DEgB3R62D5Wf5hDkHH");
list.add("2CWfKD2Lx56cUtAZWTv1XUosJwCTPeG");
list.add("2CfsNtmZ7h5bgCKhManD8EVgLzATctT");
list.add("2CVU8kut2DVtLX8UhCx8Bd9cdLTbs84");
list.add("2CcPoF8VsSwkYHY45PKrXuviKyaXL3E");
list.add("2CV71AkLjfTmGGbfRAdRks354mtagAt");
list.add("2CYDL8DyTS6NSTRonXtgeMkcEkdtYLP");
list.add("2CZKbvmEGjqhEKJHvtQUwpZqURNiVsy");
list.add("2CdvmDyH8BZERtqFxyzBMYci6Lb1o2f");
list.add("2CiXuVmydsN91XRFoXWb96MbQxN1r95");
list.add("2CVYi8FG19yevLGMA4aWip66FXa1yPM");
list.add("2CYtphwb563mkWRSqHasn6PdDA7quMF");
list.add("2CXoVMETRuTr9AiHKNVPNTidw9zaKxq");
list.add("2CWr2XPBMCjR2cn7fTfBTHMWCBDrLpX");
list.add("2CiX199ak2wJ7KyAW7heBmShZdNTRci");
list.add("2CXRjvKPUtHAHGUb4k87f7Q4MCmbGSk");
list.add("2CeagHhjeQ3yvoj4hr9j8WLPi666PgQ");
list.add("2ChFonnG83efYYtLUsfJDgF7nq6FX5b");
list.add("2CgiJ4VPVzJpcdzaLU7cX9Tqg2X3WYS");
list.add("2ChGvFBJB2QwHwViD64cxtE4p73ivBc");
```

```
    list.add("2CkZweBpnWYnoseyScLdiWc77RYTkCk");
    list.add("2CXnobsnFebExBZN63JmzMF8UNCp9HE");
    list.add("2CjziRzR8hWesf5z5TBahaJXvFWgbCH");
    list.add("2Ci6ZBtnXHNJrGAbWwHwgSWJWxrB98R");
    list.add("2CfvNnddxUz4kV7VtjYiwzfcJ3PCodo");
    list.add("2CaSYqYXADQqpMU5SkqFxpzFJeMSZcJ");
    list.add("2Cj1iAztSFGRTY4eX7pjQ6uk3gJhkzo");
    list.add("2CYrbFKfmuk5sihkGWg75GtYfrq8AtX");
    list.add("2ChpyEHB2eTU3Jhdq6jAWm1rKG5RNr2");
    list.add("2CVpkMQ7ewLbHLHqAqKcxLqt7Ba35rm");
    list.add("2CkJxvgxcaPPnPVgBFcewHdHdfVFpKn");
    list.add("2CVrEftP4XoyUWKFdZgMoA5JQhbm2yF");
    list.add("2ChgHHrW81zxiv4v96sNgYRMnr4rjXD");
    list.add("2Cbm57iGZUDCfh4uKc1Vpb9JPJVBNuy");
    list.add("2Cgp3SWYWoFefyPKniRLMwjCA5WMeHx");
    list.add("2CcJn7Bg9hknLzRp62BSzWzrcL95LWo");
    list.add("2CWnMRAKYUjd7r4vMMaDbVvFEcF7pz3");
    list.add("2ChmWVLCgzp5CB6mTRmy3vJV2KXwCQe");
    list.add("2CaiUxt4g8NZW6JTj7yzsZiGHFSjNYu");
    list.add("2Cf2HtK8ATiFQ5XZN8PvHvj1UXZBYPY");
    list.add("2CauS5Zc7M5hQuCeJXyhmMiTNnq77fB");
    list.add("2CgTPo8nYobx4T2sqQer9X4eeewwhxf");
    list.add("2CcjKN7bGr2mXzWTLhHXZEUtMU96ZA8");
    list.add("2CW4AiPggkRipHw3PHWig112LDEUGfC");
    list.add("2CVLibmSAx8cK3iwB3Uk8X7tJJJKhJF");
    list.add("2Ce94xXZ7sTKCrUNgQL3NN8F7NhdLsB");
    list.add("2CY8ZghBG7Q18rgmFxuPmw7Yf2mYyB6");
    list.add("2CenDwftmMXfFi2xf3ccydD7SVoG1tp");
    list.add("2CiYUrAGmd5S3PNSBt8ERJKPqZizZZd");
    list.add("2CeYeDfvfwBXL9PnLykdAW3K9ox7Nam");
    list.add("2CkTtW9n9ZQXVE5CLUyweVpkj4XuDTy");
    list.add("2CfwW5B2crpXx7GL6v35vybFN4XWMyH");
    list.add("2ChJhikdEQ1YPevk77Wxcfa79qUyU2y");
    return list;
}

public String post(String url, final String param, String encoding) {
    StringBuffer sb = new StringBuffer();
    OutputStream os = null;
    InputStream is = null;
    InputStreamReader isr = null;
    BufferedReader br = null;
```

```
// UTF-8
if (StringUtils.isNull(encoding)) {
    encoding = "UTF-8";
}
try {
    URL u = new URL(url);
    HttpURLConnection connection = (HttpURLConnection) u.openConnection();
    connection.setRequestProperty("Content-Type", "application/json");
    connection.setDoOutput(true);
    connection.setDoInput(true);
    connection.setRequestMethod("POST");

    connection.connect();

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

```
            if (isr != null) {
                try {
                    isr.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
        }
        return sb.toString();
    }
}
```

141:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\constant\ConsensusStorageConstant.java

```
 */

package io.nuls.consensus.poc.storage.constant;

public interface ConsensusStorageConstant {

    String DB_NAME_CONSENSUS_AGENT = "consensus_agent";
    String DB_NAME_CONSENSUS_DEPOSIT = "consensus_deposit";
    String DB_NAME_CONSENSUS_PUNISH_LOG = "consensus_punish_log";
    String DB_NAME_CONSENSUS_BIFURCATION_EVIDENCE =
"consensus_bifurcation_evidence";
    String DB_BIFURCATION_EVIDENCE_KEY = "bifurcation_evidence_key";

}
```

142:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\po\AgentPo.java

```
 *
 */

package io.nuls.consensus.poc.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.Na;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;

/**
 * @author ln
 */
public class AgentPo extends BaseNulsData {

    private transient NulsDigestData hash;

    private byte[] agentAddress;

    private byte[] packingAddress;

    private byte[] rewardAddress;

    private Na deposit;

    private double commissionRate;

    private long time;

    private long blockHeight = -1L;

    private long delHeight = -1L;

    /**
     * serialize important field
     */
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeNulsData(hash);
```

```
    stream.write(agentAddress);
    stream.write(packingAddress);
    stream.write(rewardAddress);
    stream.writeInt64(deposit.getValue());
    stream.writeDouble(commissionRate);
    stream.writeUint48(time);
    stream.writeVarInt(blockHeight);
    stream.writeVarInt(delHeight);
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.hash = byteBuffer.readHash();
    this.agentAddress = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.packingAddress = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.rewardAddress = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
    this.deposit = Na.valueOf(byteBuffer.readInt64());
    this.commissionRate = byteBuffer.readDouble();
    this.time = byteBuffer.readUint48();
    this.blockHeight = byteBuffer.readVarInt();
    this.delHeight = byteBuffer.readVarInt();
}

@Override
public int size() {
    int size = SerializeUtils.sizeOfNulsData(hash);
    size += Address.ADDRESS_LENGTH * 3;
    size += SerializeUtils.sizeOfInt64();
    size += SerializeUtils.sizeOfDouble(commissionRate);
    size += SerializeUtils.sizeOfUint48();
    size += SerializeUtils.sizeOfVarInt(blockHeight);
    size += SerializeUtils.sizeOfVarInt(delHeight);
    return size;
}

public NulsDigestData getHash() {
    return hash;
}

public void setHash(NulsDigestData hash) {
    this.hash = hash;
}
```

```
public byte[] getAgentAddress() {
    return agentAddress;
}

public void setAgentAddress(byte[] agentAddress) {
    this.agentAddress = agentAddress;
}

public byte[] getPackingAddress() {
    return packingAddress;
}

public void setPackingAddress(byte[] packingAddress) {
    this.packingAddress = packingAddress;
}

public byte[] getRewardAddress() {
    return rewardAddress;
}

public void setRewardAddress(byte[] rewardAddress) {
    this.rewardAddress = rewardAddress;
}

public Na getDeposit() {
    return deposit;
}

public void setDeposit(Na deposit) {
    this.deposit = deposit;
}

public double getCommissionRate() {
    return commissionRate;
}

public void setCommissionRate(double commissionRate) {
    this.commissionRate = commissionRate;
}

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 getDelHeight() {
        return delHeight;
    }

    public void setDelHeight(long delHeight) {
        this.delHeight = delHeight;
    }
}

143:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
storage\src\main\java\io\nuls\consensus\poc\storage\po\DepositPo.java
 */

package io.nuls.consensus.poc.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.Na;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;
```

```
/**
 * @author: Niels Wang
 */
public class DepositPo extends BaseNulsData {


    private NulsDigestData txHash;
    private Na deposit;
    private NulsDigestData agentHash;
    private byte[] address;
    private long time;
    private long blockHeight = -1L;
    private long delHeight = -1L;

    /**
     * serialize important field
     */
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeInt64(deposit.getValue());
        stream.writeNulsData(agentHash);
        stream.write(address);
        stream.writeUint48(time);
        stream.writeNulsData(txHash);
        stream.writeVarInt(blockHeight);
        stream.writeVarInt(delHeight);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.deposit = Na.valueOf(byteBuffer.readInt64());
        this.agentHash = byteBuffer.readHash();
        this.address = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
        this.time = byteBuffer.readUint48();
        this.txHash = byteBuffer.readHash();
        this.blockHeight = byteBuffer.readVarInt();
        this.delHeight = byteBuffer.readVarInt();
    }

    @Override
    public int size() {
        int size = 0;
```

```
    size += SerializeUtils.sizeOfInt64(); // deposit.getValue()
    size += SerializeUtils.sizeOfNulsData(agentHash);
    size += address.length;
    size += SerializeUtils.sizeOfUint48();
    size += SerializeUtils.sizeOfNulsData(txHash);
    size += SerializeUtils.sizeOfVarInt(blockHeight);  // blockHeight
    size += SerializeUtils.sizeOfVarInt(delHeight);  // delHeight
    return size;
}

public Na getDeposit() {
    return deposit;
}

public void setDeposit(Na deposit) {
    this.deposit = deposit;
}

public NulsDigestData getAgentHash() {
    return agentHash;
}

public void setAgentHash(NulsDigestData agentHash) {
    this.agentHash = agentHash;
}

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

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

public long getTime() {
    return time;
}

public void setTime(long time) {
    this.time = time;
}
```

```
    public NulsDigestData getTxHash() {
        return txHash;
    }

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

    public long getBlockHeight() {
        return blockHeight;
    }

    public void setBlockHeight(long blockHeight) {
        this.blockHeight = blockHeight;
    }

    public long getDelHeight() {
        return delHeight;
    }

    public void setDelHeight(long delHeight) {
        this.delHeight = delHeight;
    }
}

144:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
storage\src\main\java\io\nuls\consensus\poc\storage\po\EvidencePo.java
 */
package io.nuls.consensus.poc.storage.po;

import io.nuls.kernel.model.BlockHeader;

/**
 * @author: Charlie
 * @date: 2018/9/4
 */
public class EvidencePo {

    private long roundIndex;
    private BlockHeader blockHeader1;
    private BlockHeader blockHeader2;
```

```
    public EvidencePo(){

    }

    public EvidencePo(long roundIndex, BlockHeader blockHeader1, BlockHeader blockHeader2){
        this.roundIndex = roundIndex;
        this.blockHeader1 = blockHeader1;
        this.blockHeader2 = blockHeader2;
    }

    public long getRoundIndex() {
        return roundIndex;
    }

    public void setRoundIndex(long roundIndex) {
        this.roundIndex = roundIndex;
    }

    public BlockHeader getBlockHeader1() {
        return blockHeader1;
    }

    public void setBlockHeader1(BlockHeader blockHeader1) {
        this.blockHeader1 = blockHeader1;
    }

    public BlockHeader getBlockHeader2() {
        return blockHeader2;
    }

    public void setBlockHeader2(BlockHeader blockHeader2) {
        this.blockHeader2 = blockHeader2;
    }
}

145:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
storage\src\main\java\io\nuls\consensus\poc\storage\po\PunishLogPo.java
 *
 */

package io.nuls.consensus.poc.storage.po;
```

```
import io.nuls.core.tools.array.ArraysTool;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.utils.*;

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

/**
 * @author Niels
 */
public class PunishLogPo extends BaseNulsData {
    private byte type;
    private byte[] address;
    private long time;
    private long height;
    private long roundIndex;
    private short reasonCode;
    private byte[] evidence;
    private int index;

    /**
     * serialize important field
     */
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.write(type);
        stream.write(address);
        stream.writeUint48(time);
        stream.writeVarInt(height);
        stream.writeVarInt(roundIndex);
        stream.writeShort(reasonCode);
        stream.writeBytesWithLength(evidence);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.type = byteBuffer.readByte();
        this.address = byteBuffer.readBytes(Address.ADDRESS_LENGTH);
        this.time = byteBuffer.readUint48();
        this.height = byteBuffer.readVarInt();
```

```
    this.roundIndex = byteBuffer.readVarInt();
    this.reasonCode = byteBuffer.readShort();
    this.evidence = byteBuffer.readByLengthByte();
}

@Override
public int size() {
    int size = 0;
    size += 1;
    size += Address.ADDRESS_LENGTH;
    size += SerializeUtils.sizeOfUint48();
    size += SerializeUtils.sizeOfVarInt(height);
    size += SerializeUtils.sizeOfVarInt(roundIndex);
    size += 2;
    size += SerializeUtils.sizeOfBytes(this.evidence);
    return size;
}

public long getRoundIndex() {
    return roundIndex;
}

public void setRoundIndex(long roundIndex) {
    this.roundIndex = roundIndex;
}

public byte getType() {
    return type;
}

public void setType(byte type) {
    this.type = type;
}

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

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

```
    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 int getIndex() {
        return index;
    }

    public void setIndex(int index) {
        this.index = index;
    }

    public byte[] getKey() {
        return ArraysTool.concatenate(address, new byte[]{type},
SerializeUtils.uint64ToByteArray(height), new VarInt(index).encode());
    }

    public void setReasonCode(short reasonCode) {
        this.reasonCode = reasonCode;
    }

    public void setEvidence(byte[] evidence) {
        this.evidence = evidence;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof PunishLogPo)) {
            return false;
        }
```

```
        return Arrays.equals(this.getKey(), ((PunishLogPo) obj).getKey());
    }

    public short getReasonCode() {
        return reasonCode;
    }

    public byte[] getEvidence() {
        return evidence;
    }
}
```

146:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\AgentStorageService.java

```
 *
 */

package io.nuls.consensus.poc.storage.service;

import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.kernel.model.NulsDigestData;

import java.util.List;

/**
 * Created by ln on 2018/5/10.
 */
public interface AgentStorageService {

    boolean save(AgentPo agentPo);

    AgentPo get(NulsDigestData hash);

    boolean delete(NulsDigestData hash);

    List<AgentPo> getList();

    int size();
}
```

147:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\BifurcationEvidenceStorageService.j

```
ava
 */
package io.nuls.consensus.poc.storage.service;



import io.nuls.consensus.poc.storage.po.EvidencePo;

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

/**
 * @author: Charlie
 * @date: 2018/9/4
 */
public interface BifurcationEvidenceStorageService {

    Map<String, List<EvidencePo>> getBifurcationEvidence();

    boolean save(Map<String, List<EvidencePo>> map);

}

148:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
storage\src\main\java\io\nuls\consensus\poc\storage\service\DepositStorageService.java
 */

package io.nuls.consensus.poc.storage.service;

import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.kernel.model.NulsDigestData;

import java.util.List;

/**
 * @author: Niels Wang
 */
public interface DepositStorageService {

    boolean save(DepositPo depositPo);

    DepositPo get(NulsDigestData hash);
```

```
    boolean delete(NulsDigestData hash);

    List<DepositPo> getList();

    int size();
}
```

149:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\impl\AgentStorageServiceImpl.java

```
 *
 */

package io.nuls.consensus.poc.storage.service.impl;

import io.nuls.consensus.poc.storage.constant.ConsensusStorageConstant;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.consensus.poc.storage.service.AgentStorageService;
import io.nuls.core.tools.log.Log;
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 java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

/**
 * Created by ln on 2018/5/10.
 */
@Component
public class AgentStorageServiceImpl implements AgentStorageService, InitializingBean {

    @Autowired
    private DBService dbService;
```

```
    @Override
    public boolean save(AgentPo agentPo) {
        if (agentPo == null || agentPo.getHash() == null) {
            return false;
        }
        byte[] hash;
        try {
            hash = agentPo.getHash().serialize();
        } catch (IOException e) {
            Log.error(e);
            throw new NulsRuntimeException(e);
        }
        Result result = null;
        try {
            result = dbService.put(ConsensusStorageConstant.DB_NAME_CONSENSUS_AGENT,
hash, agentPo.serialize());
        } catch (IOException e) {
            Log.error(e);
            return false;
        }
        return result.isSuccess();
    }

    @Override
    public AgentPo get(NulsDigestData hash) {
        if (hash == null) {
            return null;
        }
        byte[] body = null;
        try {
            body = dbService.get(ConsensusStorageConstant.DB_NAME_CONSENSUS_AGENT,
hash.serialize());
        } catch (IOException e) {
            Log.error(e);
        }
        if (body == null) {
            return null;
        }
        AgentPo agentPo = new AgentPo();
        try {
            agentPo.parse(body,0);
```

```
        } catch (NulsException e) {
            Log.error(e);
            throw  new NulsRuntimeException(e);
        }
        agentPo.setHash(hash);
        return agentPo;
    }

    @Override
    public boolean delete(NulsDigestData hash) {
        if (hash == null) {
            return false;
        }
        Result result = null;
        try {
            result = dbService.delete(ConsensusStorageConstant.DB_NAME_CONSENSUS_AGENT,
hash.serialize());
        } catch (IOException e) {
            Log.error(e);
        }
        return result.isSuccess();
    }

    @Override
    public List<AgentPo> getList() {
        List<Entry<byte[], byte[]>> list =
dbService.entryList(ConsensusStorageConstant.DB_NAME_CONSENSUS_AGENT);
        List<AgentPo> resultList = new ArrayList<>();
        if (list == null) {
            return resultList;
        }
        for (Entry<byte[], byte[]> entry : list) {
            AgentPo agentPo = new AgentPo();
            try {
                agentPo.parse(entry.getValue(),0);
            } catch (NulsException e) {
                Log.error(e);
                throw  new NulsRuntimeException(e);
            }
            NulsDigestData hash = new NulsDigestData();
            try {
                hash.parse(entry.getKey(),0);
```

```
            } catch (NulsException e) {
                Log.error(e);
            }
            agentPo.setHash(hash);
            resultList.add(agentPo);
        }
        return resultList;
    }

    @Override
    public int size() {
        Set<byte[]> list =
dbService.keySet(ConsensusStorageConstant.DB_NAME_CONSENSUS_AGENT);
        if (list == null) {
            return 0;
        }
        return list.size();
    }

    @Override
    public void afterPropertiesSet() throws NulsException {
        dbService.createArea(ConsensusStorageConstant.DB_NAME_CONSENSUS_AGENT);
    }
}
```

150:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\impl\BifurcationEvidenceStorageServiceImpl.java

```
 */
package io.nuls.consensus.poc.storage.service.impl;

import io.nuls.consensus.poc.storage.constant.ConsensusStorageConstant;
import io.nuls.consensus.poc.storage.po.EvidencePo;
import io.nuls.consensus.poc.storage.service.BifurcationEvidenceStorageService;
import io.nuls.db.service.DBService;
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.Result;

import java.util.List;
```

```
import java.util.Map;

/**
 * @author: Charlie
 * @date: 2018/9/4
 */
@Component
public class BifurcationEvidenceStorageServiceImpl implements
BifurcationEvidenceStorageService, InitializingBean {
    @Autowired
    private DBService dbService;

    @Override
    public Map<String, List<EvidencePo>> getBifurcationEvidence() {
        Map<String, List<EvidencePo>> map =
dbService.getModel(ConsensusStorageConstant.DB_NAME_CONSENSUS_BIFURCATION_EVI
DENCE,
                ConsensusStorageConstant.DB_BIFURCATION_EVIDENCE_KEY.getBytes(),
Map.class);
        return map;
    }

    @Override
    public boolean save(Map<String, List<EvidencePo>> map) {
       Result result =
dbService.putModel(ConsensusStorageConstant.DB_NAME_CONSENSUS_BIFURCATION_EVI
DENCE,
                ConsensusStorageConstant.DB_BIFURCATION_EVIDENCE_KEY.getBytes(), map);
        return result.isSuccess();
    }

    @Override
    public void afterPropertiesSet() throws NulsException {
dbService.createArea(ConsensusStorageConstant.DB_NAME_CONSENSUS_BIFURCATION_E
VIDENCE);
    }
}

151:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-
storage\src\main\java\io\nuls\consensus\poc\storage\service\impl\DepositStorageServiceImpl.java
 */
```

```
package io.nuls.consensus.poc.storage.service.impl;

import io.nuls.consensus.poc.storage.constant.ConsensusStorageConstant;
import io.nuls.consensus.poc.storage.po.DepositPo;
import io.nuls.consensus.poc.storage.service.DepositStorageService;
import io.nuls.core.tools.log.Log;
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 java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

/**
 * @author: Niels Wang
 */
@Component
public class DepositStorageServiceImpl implements DepositStorageService, InitializingBean {

    @Autowired
    private DBService dbService;

    @Override
    public boolean save(DepositPo depositPo) {
        if (depositPo == null || depositPo.getTxHash() == null) {
            return false;
        }
        byte[] hash;
        try {
            hash = depositPo.getTxHash().serialize();
        } catch (IOException e) {
            Log.error(e);
            throw new NulsRuntimeException(e);
        }
```

```
        Result result = null;
        try {
            result = dbService.put(ConsensusStorageConstant.DB_NAME_CONSENSUS_DEPOSIT,
hash, depositPo.serialize());
        } catch (IOException e) {
            Log.error(e);
            return false;
        }
        return result.isSuccess();
    }

    @Override
    public DepositPo get(NulsDigestData hash) {
        if (hash == null) {
            return null;
        }
        byte[] body = null;
        try {
            body = dbService.get(ConsensusStorageConstant.DB_NAME_CONSENSUS_DEPOSIT,
hash.serialize());
        } catch (IOException e) {
            Log.error(e);
        }
        if (body == null) {
            return null;
        }
        DepositPo depositPo = new DepositPo();
        try {
            depositPo.parse(body, 0);
        } catch (NulsException e) {
            Log.error(e);
            throw new NulsRuntimeException(e);
        }
        depositPo.setTxHash(hash);
        return depositPo;
    }

    @Override
    public boolean delete(NulsDigestData hash) {
        if (hash == null) {
            return false;
        }
```

```
        Result result = null;
        try {
            result =
dbService.delete(ConsensusStorageConstant.DB_NAME_CONSENSUS_DEPOSIT,
hash.serialize());
        } catch (IOException e) {
            Log.error(e);
        }
        return result.isSuccess();
    }

    @Override
    public List<DepositPo> getList() {
        List<Entry<byte[], byte[]>> list =
dbService.entryList(ConsensusStorageConstant.DB_NAME_CONSENSUS_DEPOSIT);
        List<DepositPo> resultList = new ArrayList<>();
        if (list == null) {
            return resultList;
        }
        for (Entry<byte[], byte[]> entry : list) {
            DepositPo depositPo = new DepositPo();
            try {
                depositPo.parse(entry.getValue(), 0);
            } catch (NulsException e) {
                Log.error(e);
                throw new NulsRuntimeException(e);
            }
            NulsDigestData hash = new NulsDigestData();
            try {
                hash.parse(entry.getKey(), 0);
            } catch (NulsException e) {
                Log.error(e);
            }
            depositPo.setTxHash(hash);
            resultList.add(depositPo);
        }
        return resultList;
    }

    @Override
    public int size() {
        Set<byte[]> list =
```

```
dbService.keySet(ConsensusStorageConstant.DB_NAME_CONSENSUS_DEPOSIT);
        if (list == null) {
            return 0;
        }
        return list.size();
    }

    @Override
    public void afterPropertiesSet() throws NulsException {
        dbService.createArea(ConsensusStorageConstant.DB_NAME_CONSENSUS_DEPOSIT);
    }
}
```

152:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\impl\OrphanStorageServiceImpl.java

```
 *
 */

package io.nuls.consensus.poc.storage.service.impl;

import io.nuls.consensus.poc.storage.service.OrphanStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.db.service.DBService;
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.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;

import java.io.IOException;

/**
 * Created by ln on 2018/5/8.
 */
@Component
public class OrphanStorageServiceImpl implements OrphanStorageService, InitializingBean {

    private final String DB_NAME = "orphan";

    @Autowired
```

```
    private DBService dbService;

    @Override
    public boolean save(Block block) {

        assert (block != null);

        Result result = null;
        try {
            result = dbService.put(DB_NAME, block.getHeader().getHash().getDigestBytes(),
block.serialize());
        } catch (IOException e) {
            Log.error(e);
            return false;
        }
        return result.isSuccess();
    }

    @Override
    public Block get(NulsDigestData key) {

        assert (key != null);

        byte[] content = dbService.get(DB_NAME, key.getDigestBytes());
        Block block = new Block();
        try {
            block.parse(content, 0);
        } catch (NulsException e) {
            Log.error(e);
        }

        return block;
    }

    @Override
    public boolean delete(NulsDigestData key) {

        assert (key != null);

        Result result = dbService.delete(DB_NAME, key.getDigestBytes());

        return result.isSuccess();
```

```
    }

    @Override
    public void afterPropertiesSet() throws NulsException {
//        dbService.createArea(DB_NAME);
    }
}
```

153:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\impl\PunishLogStorageServiceImpl.java

```
 */

package io.nuls.consensus.poc.storage.service.impl;

import io.nuls.consensus.poc.storage.constant.ConsensusStorageConstant;
import io.nuls.consensus.poc.storage.po.PunishLogPo;
import io.nuls.consensus.poc.storage.service.PunishLogStorageService;
import io.nuls.core.tools.log.Log;
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.Result;

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

/**
 * @author: Niels Wang
 */
@Component
public class PunishLogStorageServiceImpl implements PunishLogStorageService, InitializingBean
{

    @Autowired
    private DBService dbService;
```

```
    @Override
    public boolean save(PunishLogPo po) {
        if (po == null || po.getKey() == null) {
            return false;
        }
        Result result = null;
        try {
            result =
dbService.put(ConsensusStorageConstant.DB_NAME_CONSENSUS_PUNISH_LOG,
po.getKey(), po.serialize());
        } catch (IOException e) {
            Log.error(e);
            return false;
        }
        return result.isSuccess();
    }

    @Override
    public boolean delete(byte[] key) {
        if (null == key) {
            return false;
        }
        Result result =
dbService.delete(ConsensusStorageConstant.DB_NAME_CONSENSUS_PUNISH_LOG, key);
        return result.isSuccess();
    }

    @Override
    public List<PunishLogPo> getPunishList() {
        List<Entry<byte[], byte[]>> list =
dbService.entryList(ConsensusStorageConstant.DB_NAME_CONSENSUS_PUNISH_LOG);
        List<PunishLogPo> polist = new ArrayList<>();
        for (Entry<byte[], byte[]> entry : list) {
            PunishLogPo po = new PunishLogPo();
            try {
                po.parse(entry.getValue(), 0);
            } catch (NulsException e) {
                throw new NulsRuntimeException(e);
            }
            polist.add(po);
        }
        return polist;
```

```
    }

    @Override
    public void afterPropertiesSet() throws NulsException {
dbService.createArea(ConsensusStorageConstant.DB_NAME_CONSENSUS_PUNISH_LOG);      }
}
```

154:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\impl\TransactionCacheStorageServiceImpl.java

```
 */
package io.nuls.consensus.poc.storage.service.impl;

import io.nuls.consensus.poc.storage.service.TransactionCacheStorageService;
import io.nuls.core.tools.crypto.Util;
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.core.bean.InitializingBean;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.TransactionManager;

import java.io.IOException;
import java.util.concurrent.atomic.AtomicInteger;

@Component
public class TransactionCacheStorageServiceImpl implements TransactionCacheStorageService, InitializingBean {

    private final static String TRANSACTION_CACHE_KEY_NAME = "transactions_cache";
    private final static byte[] LAST_KEY = "last_key".getBytes();
    private final static byte[] START_KEY = "start_key".getBytes();
    private int lastIndex = 0;
    private AtomicInteger startIndex = new AtomicInteger(0);
```

```
    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    @Override
    public void afterPropertiesSet() throws NulsException {
        dbService.destroyArea(TRANSACTION_CACHE_KEY_NAME);

        Result result = this.dbService.createArea(TRANSACTION_CACHE_KEY_NAME);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
        startIndex.set(1);
    }

    @Override
    public boolean putTx(Transaction tx) {
        if (tx == null) {
            return false;
        }
        byte[] txHashBytes = null;
        try {
            txHashBytes = tx.getHash().serialize();
        } catch (IOException e) {
            Log.error(e);
            return false;
        }
        //
        Result result = null;
        try {
            result = dbService.put(TRANSACTION_CACHE_KEY_NAME, txHashBytes, tx.serialize());
        } catch (IOException e) {
            Log.error(e);
            return false;
        }

//        if(!result.isSuccess()) {
//            return result.isSuccess();
//        }
```

```
//        lastIndex++;
//        byte[] lastIndexBytes = Util.intToBytes(lastIndex);
//        result = dbService.put(TRANSACTION_CACHE_KEY_NAME, lastIndexBytes,
txHashBytes);
//        if(!result.isSuccess()) {
//            removeTx(tx.getHash());
//            return result.isSuccess();
//        }
//        result = dbService.put(TRANSACTION_CACHE_KEY_NAME, LAST_KEY, lastIndexBytes);
        return result.isSuccess();
    }

    @Override
    public int getStartIndex() {
        byte[] lastIndexBytes = dbService.get(TRANSACTION_CACHE_KEY_NAME, START_KEY);
        if (lastIndexBytes == null) {
            return 0;
        }
        return Util.byteToInt(lastIndexBytes);
    }

    @Override
    public Transaction pollTx() {

        byte[] startIndexBytes = Util.intToBytes(startIndex.get());

        byte[] hashBytes = dbService.get(TRANSACTION_CACHE_KEY_NAME, startIndexBytes);
        if (hashBytes == null) {
            return null;
        }

        byte[] txBytes = dbService.get(TRANSACTION_CACHE_KEY_NAME, hashBytes);
        Transaction tx = null;
        if (null != txBytes) {
            try {
                tx = TransactionManager.getInstance(new NulsByteBuffer(txBytes, 0));
            } catch (Exception e) {
                Log.error(e);
                return null;
            }
        }
```

```
        startIndex.incrementAndGet();
//        dbService.put(TRANSACTION_CACHE_KEY_NAME, START_KEY,
Util.intToBytes(startIndex));

        return tx;
    }

    @Override
    public Transaction getTx(NulsDigestData hash) {
        if (hash == null) {
            return null;
        }
        byte[] hashBytes = null;
        try {
            hashBytes = hash.serialize();
        } catch (IOException e) {
            Log.error(e);
            throw new NulsRuntimeException(e);
        }
        byte[] txBytes = dbService.get(TRANSACTION_CACHE_KEY_NAME, hashBytes);
        Transaction tx = null;
        if (null != txBytes) {
            try {
                tx = TransactionManager.getInstance(new NulsByteBuffer(txBytes, 0));
            } catch (Exception e) {
                Log.error(e);
                return null;
            }
        }
        return tx;
    }

    @Override
    public boolean removeTx(NulsDigestData hash) {
        if (hash == null) {
            return false;
        }
        try {
            Result result = dbService.delete(TRANSACTION_CACHE_KEY_NAME, hash.serialize());
            return result.isSuccess();
        } catch (IOException e) {
            Log.error(e);
```

```
        }
        return false;
    }
}

155:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\impl\TransactionQueueStorageServiceImpl.java
 */

package io.nuls.consensus.poc.storage.service.impl;

import io.nuls.consensus.poc.storage.service.TransactionQueueStorageService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.TransactionManager;
import io.nuls.kernel.utils.queue.entity.PersistentQueue;

import java.io.IOException;

/**
 * @author: Niels Wang
 * @date: 2018/7/8
 */
@Component
public class TransactionQueueStorageServiceImpl implements TransactionQueueStorageService
{

    private PersistentQueue queue = new PersistentQueue("tx-cache-queue", 10000000L);

    public TransactionQueueStorageServiceImpl() throws Exception {
    }

    @Override
    public boolean putTx(Transaction tx) {
        try {
            queue.offer(tx.serialize());
            return true;
        } catch (IOException e) {
            Log.error(e);
```

```
        }
        return false;
    }

    @Override
    public Transaction pollTx() {
        byte[] bytes = queue.poll();
        if (null == bytes) {
            return null;
        }
        try {
            return TransactionManager.getInstance(new NulsByteBuffer(bytes));
        } catch (Exception e) {
            Log.error(e);
        }
        return null;
    }
}
```

156:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\OrphanStorageService.java

```
 *
 */

package io.nuls.consensus.poc.storage.service;

import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;

/**
 *
 * @author ln
 */
public interface OrphanStorageService {

    boolean save(Block block);

    Block get(NulsDigestData key);

    boolean delete(NulsDigestData key);

}
```

157:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\PunishLogStorageService.java
*/

```
package io.nuls.consensus.poc.storage.service;

import io.nuls.consensus.poc.storage.po.PunishLogPo;

import java.util.List;

/**
 * @author: Niels Wang
 */
public interface PunishLogStorageService {

    boolean save(PunishLogPo po);

    boolean delete(byte[] key);

    List<PunishLogPo> getPunishList();
}
```

158:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\TransactionCacheStorageService.java
*/

```
package io.nuls.consensus.poc.storage.service;

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;

public interface TransactionCacheStorageService {

    boolean putTx(Transaction tx);

    Transaction getTx(NulsDigestData hash);

    boolean removeTx(NulsDigestData hash);

    int getStartIndex();
```

```
    Transaction pollTx();
}
```

159:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\service\TransactionQueueStorageService.java

```
 */
package io.nuls.consensus.poc.storage.service;

import io.nuls.kernel.model.Transaction;

public interface TransactionQueueStorageService {

    boolean putTx(Transaction tx);

    Transaction pollTx();
}
```

160:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\main\java\io\nuls\consensus\poc\storage\utils\PunishLogComparator.java

```
 */

package io.nuls.consensus.poc.storage.utils;

import io.nuls.consensus.poc.storage.po.PunishLogPo;

import java.util.Comparator;

/**
 * @author: Niels Wang
 */
public class PunishLogComparator implements Comparator<PunishLogPo> {

    @Override
    public int compare(PunishLogPo o1, PunishLogPo o2) {
        if (o1.getHeight() == o2.getHeight()) {
            return (int) (o1.getTime() - o2.getTime());
        }
        return (int) (o1.getHeight() - o2.getHeight());
    }
}
```

161:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\test\java\io\nuls\consensus\poc\storage\BaseTest.java

```
 *
 */

package io.nuls.consensus.poc.storage;

import io.nuls.kernel.MicroKernelBootstrap;
import org.junit.Before;
import org.junit.BeforeClass;

/**
 * Created by ln on 2018/5/10.
 */
public class BaseTest {

    @BeforeClass
    public static void initMicroKernel() {
        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();
    }
}
```

162:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\test\java\io\nuls\consensus\poc\storage\service\AgentStorageServiceTest.java

```
 *
 */

package io.nuls.consensus.poc.storage.service;

import io.nuls.consensus.poc.storage.BaseTest;
import io.nuls.consensus.poc.storage.po.AgentPo;
import io.nuls.kernel.lite.core.SpringLiteContext;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsDigestData;
import org.junit.Before;
import org.junit.Test;

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

```
import static org.junit.Assert.*;

/**
 * Created by ln on 2018/5/10.
 */
public class AgentStorageServiceTest extends BaseTest {

    private AgentStorageService agentStorageService;

    @Before
    public void init() {
        agentStorageService = SpringLiteContext.getBean(AgentStorageService.class);
        List<AgentPo> list = agentStorageService.getList();
        if(list != null) {
            for (AgentPo agentPo : list) {
                agentStorageService.delete(agentPo.getHash());
            }
        }
    }

    @Test
    public void testSave() {
        assertNotNull(agentStorageService);

        NulsDigestData hash = NulsDigestData.calcDigestData(new byte[23]);

        AgentPo agentPo = new AgentPo();
        agentPo.setAgentAddress(new byte[23]);
        agentPo.setRewardAddress(new byte[23]);
        agentPo.setPackingAddress(new byte[23]);
        agentPo.setDeposit(Na.ZERO);
        agentPo.setHash(hash);

        boolean success = agentStorageService.save(agentPo);

        assert(success);
    }

    @Test
    public void testGet() {
        assertNotNull(agentStorageService);
```

```
    testSave();

    NulsDigestData hash = NulsDigestData.calcDigestData(new byte[23]);

    AgentPo agentPo = agentStorageService.get(hash);

    assertNotNull(agentPo);

    assert(Arrays.equals(agentPo.getAgentAddress(), new byte[23]));
}

@Test
public void testDelete() {
    assertNotNull(agentStorageService);

    testSave();

    NulsDigestData hash = NulsDigestData.calcDigestData(new byte[23]);

    boolean success = agentStorageService.delete(hash);

    assert(success);

    AgentPo agentPo = agentStorageService.get(hash);

    assertNull(agentPo);
}

@Test
public void testList() {
    assertNotNull(agentStorageService);

    testSave();

    NulsDigestData hash = NulsDigestData.calcDigestData(new byte[20]);

    AgentPo agentPo = new AgentPo();
    agentPo.setAgentAddress(new byte[23]);
    agentPo.setRewardAddress(new byte[23]);
    agentPo.setPackingAddress(new byte[23]);
    agentPo.setDeposit(Na.ZERO);
    agentPo.setHash(hash);
```

```
    boolean success = agentStorageService.save(agentPo);

    assert(success);

    List<AgentPo> list = agentStorageService.getList();

    assertEquals(list.size(), 2);

    NulsDigestData hash1 = NulsDigestData.calcDigestData(new byte[23]);

    assertEquals(hash1, list.get(0).getHash());
    assertEquals(hash, list.get(1).getHash());

    int size = agentStorageService.size();
    assertEquals(size, 2);

    success = agentStorageService.delete(hash);
    assert(success);

    success = agentStorageService.delete(hash1);
    assert(success);
}

@Test
public void testRepeatSave() {

    int i = 0;
    while(true) {
        testSave();
        testSave();
        testSave();
        testSave();
        i++;
        if(i > 100) {
            break;
        }
    }

    List<AgentPo> list = agentStorageService.getList();

    assertEquals(list.size(), 1);
```

```
    }
}
```

163:F:\git\coin\nuls\nuls-1.1.3\nuls\consensus-module\poc\consensus-poc-storage\src\test\java\io\nuls\consensus\poc\storage\service\TransactionCacheStorageServiceTest.java

```
 */
package io.nuls.consensus.poc.storage.service;

import io.nuls.core.tools.log.Log;
import io.nuls.db.module.impl.LevelDbModuleBootstrap;
import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Transaction;
import org.junit.Before;
import org.junit.Test;

public class TransactionCacheStorageServiceTest {

    TransactionCacheStorageService service = null;

    @Before
    public void init() {
        try {
            MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
            mk.init();
            mk.start();

            LevelDbModuleBootstrap bootstrap = new LevelDbModuleBootstrap();
            bootstrap.init();
            bootstrap.start();

            service = NulsContext.getServiceBean(TransactionCacheStorageService.class);
        } catch (Exception e) {
            Log.error(e);
        }
    }

    @Test
    public void test() {
        Transaction tx = null;
```

```
        long time = System.currentTimeMillis();
        for(int i = 0 ; i < 1000000 ; i ++) {
            tx = new TransactionPoTest(1);
            tx.setTime(i);
            tx.setRemark("asldfjsaldfjsldjfoijioj222fsdafasdfasdfasdfasdfasdfasdfsadfsa".getBytes());

            boolean success = service.putTx(tx);
            assert(success);
        }
        System.out.println("" + (System.currentTimeMillis() - time) + " ms");

        time = System.currentTimeMillis();

        int count = 0;
        while((tx = service.pollTx()) != null) {
            count++;
        }
        System.out.println(count);
        System.out.println("" + (System.currentTimeMillis() - time) + " ms");
        assert (count == 1000000);
    }
}
```