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

0:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-base\src\main\java\io\nuls\message\bus\manager\DispatchManager.java
*/

```
package io.nuls.message.bus.manager;

import io.nuls.core.tools.disruptor.DisruptorData;
import io.nuls.core.tools.disruptor.DisruptorUtil;
import io.nuls.kernel.module.service.ModuleService;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.message.bus.constant.MessageBusConstant;
import io.nuls.message.bus.module.MessageBusModuleBootstrap;
import io.nuls.message.bus.processor.MessageClassificationProcessor;
import io.nuls.message.bus.model.ProcessData;
import io.nuls.protocol.message.base.BaseMessage;

/**
 *
 * Message processing manager.
 *
 * @author: Charlie
 */
public class DispatchManager<M extends BaseMessage> {

    private static final DispatchManager INSTANCE = new DispatchManager();

    private DisruptorUtil<DisruptorData<ProcessData<M>>> disruptorService =
DisruptorUtil.getInstance();
    private String disruptorName = MessageBusConstant.DISRUPTOR_NAME;
    private MessageClassificationProcessor messageProcesser;

    public static DispatchManager getInstance() {
        return INSTANCE;
    }

    private DispatchManager() {
    }

    public final void init(boolean messageChecking) {
        NulsThreadFactory nulsThreadFactory = new
```

```
NulsThreadFactory(ModuleService.getInstance().getModuleId(MessageBusModuleBootstrap.class
), disruptorName);
        disruptorService.createDisruptor(disruptorName,
MessageBusConstant.DEFAULT_RING_BUFFER_SIZE, nulsThreadFactory);

        messageProcesser = new MessageClassificationProcessor();
        disruptorService.handleEventWith(disruptorName, messageProcesser);

        disruptorService.start(disruptorName);
    }

    public void shutdown() {
        messageProcesser.shutdown();
        disruptorService.shutdown(disruptorName);
    }

    public void offer(ProcessData<M> data) {
        disruptorService.offer(disruptorName, data);
    }
}

1:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-
base\src\main\java\io\nuls\message\bus\manager\HandlerManager.java
 */

package io.nuls.message.bus.manager;

import io.nuls.core.tools.param.AssertUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.message.bus.handler.intf.NulsMessageHandler;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

/**
 * Created by ln on 2018-05-23.
 */
public final class HandlerManager<M extends BaseMessage, H extends NulsMessageHandler<?
extends BaseMessage>> {
```

```
private final Map<Class, Set<String>> messageHandlerMapping = new HashMap<>();
private final Map<String, H> handlerMap = new HashMap<>();

private static final HandlerManager INSTANCE = new HandlerManager();

public static HandlerManager getInstance() {
    return INSTANCE;
}

private HandlerManager() {
}

public String registerMessageHandler(String handlerId, Class<M> messageClass, H handler) {
    AssertUtil.canNotEmpty(messageClass, "registerMessageHandler faild");
    AssertUtil.canNotEmpty(handler, "registerMessageHandler faild");
    if (StringUtils.isBlank(handlerId)) {
        handlerId = StringUtils.getNewUUID();
    }
    handlerMap.put(handlerId, handler);
    cacheHandlerMapping(messageClass, handlerId);
    return handlerId;
}

private void cacheHandlerMapping(Class<M> messageClass, String handlerId) {

    Set<String> ids = messageHandlerMapping.get(messageClass);
    if (null == ids) {
        ids = new HashSet<>();
    }
    ids.add(handlerId);
    messageHandlerMapping.put(messageClass, ids);
}

public Set<NulsMessageHandler> getHandlerList(Class<M> clazz) {
    Set<String> ids = messageHandlerMapping.get(clazz);
    Set<NulsMessageHandler> set = new HashSet<>();
    do {
        if (null == ids || ids.isEmpty()) {
            break;
        }
        for (String id : ids) {
```

```
                if (StringUtils.isBlank(id)) {
                    continue;
                }
                NulsMessageHandler handler = handlerMap.get(id);
                if (null == handler) {
                    continue;
                }
                set.add(handler);
            }
        } while (false);
        if (!clazz.equals(BaseMessage.class)) {
            set.addAll(getHandlerList((Class<M>) clazz.getSuperclass()));
        }
        return set;
    }

    public void removeMessageHandler(String handlerId) {
        handlerMap.remove(handlerId);
    }
}

2:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-
base\src\main\java\io\nuls\message\bus\model\ProcessData.java
 */

package io.nuls.message.bus.model;

import io.nuls.network.model.Node;
import io.nuls.protocol.message.base.BaseMessage;

/**
 * @author: Charlie
 */
public class ProcessData<T extends BaseMessage> {

    private final T data;

    private Node node;

    public ProcessData(T data){
        this.data = data;
    }
```

```
    public ProcessData(T data,  Node node){
        this.data = data;
        this.node = node;
    }

    public T getData(){
        return data;
    }

    public Node getNode() {
        return node;
    }

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

3:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-
base\src\main\java\io\nuls\message\bus\module\MessageBusModuleBootstrap.java
 */

package io.nuls.message.bus.module;

import io.nuls.message.bus.manager.DispatchManager;

/**
 * @author: Charlie
 */
public class MessageBusModuleBootstrap extends AbstractMessageBusModule {

    @Override
    public void init() {

    }

    @Override
    public void start() {
        DispatchManager.getInstance().init(true);
//        MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);
```

```
//        messageBusService.subscribeMessage(CommonDigestMessage.class, new
CommonDigestHandler());
//        messageBusService.subscribeMessage(GetMessageBodyMessage.class, new
GetMessageBodyHandler());
    }

    @Override
    public void shutdown() {
        DispatchManager.getInstance().shutdown();
    }

    @Override
    public void destroy() {
    }

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

4:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-base\src\main\java\io\nuls\message\bus\processor\MessageClassificationProcessor.java

```
 */
package io.nuls.message.bus.processor;

import com.lmax.disruptor.EventHandler;
import io.nuls.core.tools.disruptor.DisruptorData;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.message.bus.constant.MessageBusConstant;
import io.nuls.message.bus.handler.intf.NulsMessageHandler;
import io.nuls.message.bus.manager.HandlerManager;
import io.nuls.message.bus.model.ProcessData;
import io.nuls.message.bus.processor.thread.NulsMessageCall;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ExecutorService;
```

```
import java.util.concurrent.ThreadPoolExecutor;

/**
 * @author ln
 */
public class MessageClassificationProcessor<E extends BaseMessage> implements
EventHandler<DisruptorData<ProcessData<E>>> {

    private HandlerManager handlerManager = HandlerManager.getInstance();
    private Map<Class<? extends BaseMessage>, ExecutorService> handlerService = new
HashMap<>();

    @Override
    public void onEvent(DisruptorData<ProcessData<E>> disruptorData, long l, boolean b) throws
Exception {

        if (null == disruptorData || disruptorData.getData() == null) {
            Log.warn("there is null data in disruptorData!");
            return;
        }
        if (disruptorData.isStoped()) {
            disruptorData.setStoped(false);
            return;
        }

        ProcessData processData = disruptorData.getData();
        Class<? extends BaseMessage> serviceId = processData.getData().getClass();
        Set<NulsMessageHandler> handlers = handlerManager.getHandlerList(serviceId);
        ThreadPoolExecutor handlerExecutor = (ThreadPoolExecutor) handlerService.get(serviceId);
        if (handlerExecutor == null) {
            handlerExecutor = TaskManager.createThreadPool(1, 1000000, new
NulsThreadFactory(MessageBusConstant.MODULE_ID_MESSAGE_BUS, "disruptor-
processor"));
            handlerService.put(serviceId, handlerExecutor);
        }
        for (NulsMessageHandler handler : handlers) {
            handlerExecutor.execute(new NulsMessageCall(processData, handler));
            int size = handlerExecutor.getQueue().size();
            if (size > 1000 && (size % 1000 == 0)) {
                Log.info(serviceId + " queue size::::::::::::::::::" + size);
            }
        }
```

```
    }

    public void shutdown() {
        if (handlerService == null) {
            return;
        }
        for (Map.Entry<Class<? extends BaseMessage>, ExecutorService> entry :
handlerService.entrySet()) {
            entry.getValue().shutdown();
        }
    }
}

5:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-
base\src\main\java\io\nuls\message\bus\processor\thread\NulsMessageCall.java
 */

package io.nuls.message.bus.processor.thread;

import io.nuls.core.tools.log.Log;
import io.nuls.message.bus.handler.intf.NulsMessageHandler;
import io.nuls.message.bus.model.ProcessData;
import io.nuls.protocol.message.base.BaseMessage;

/**
 * @author: Charlie
 */
public class NulsMessageCall<T extends BaseMessage> implements Runnable {

    private final ProcessData<T> data;
    private final NulsMessageHandler<T> handler;

    public NulsMessageCall(ProcessData<T> data, NulsMessageHandler<T> handler) {
        this.data = data;
        this.handler = handler;
    }

    @Override
    public void run() {
        if (null == data || null == handler) {
            return;
        }
```

```
        try {
            long start = System.currentTimeMillis();
            handler.onMessage(data.getData(), data.getNode());
            if(Log.isDebugEnabled()) {
                Log.debug(data.getData().getClass() + ",use:" + (System.currentTimeMillis() - start));
            }
        } catch (Exception e) {
            Log.error(e);
        }
        return;
    }
}

6:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-
base\src\main\java\io\nuls\message\bus\service\impl\MessageBusServiceImpl.java
 */

package io.nuls.message.bus.service.impl;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.Result;
import io.nuls.message.bus.constant.MessageBusErrorCode;
import io.nuls.message.bus.handler.intf.NulsMessageHandler;
import io.nuls.message.bus.manager.DispatchManager;
import io.nuls.message.bus.manager.HandlerManager;
import io.nuls.message.bus.manager.MessageManager;
import io.nuls.message.bus.model.ProcessData;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.BroadcastResult;
import io.nuls.network.model.Node;
import io.nuls.network.service.NetworkService;
import io.nuls.protocol.message.base.BaseMessage;

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

/**
 * @author: Charlie
 */
@Service
```

```
public class MessageBusServiceImpl implements MessageBusService {

    @Autowired
    private NetworkService networkService;
    private HandlerManager handlerManager = HandlerManager.getInstance();
    private DispatchManager processorManager = DispatchManager.getInstance();

    @Override
    public String subscribeMessage(Class<? extends BaseMessage> messageClass,
NulsMessageHandler<? extends BaseMessage> messageHandler) {
        MessageManager.putMessage(messageClass);
        return handlerManager.registerMessageHandler(null, messageClass, messageHandler);
    }

    @Override
    public void unsubscribeMessage(String subscribeId) {
        this.handlerManager.removeMessageHandler(subscribeId);
    }

    @Override
    public void receiveMessage(BaseMessage message, Node node) {
        try {
            this.processorManager.offer(new ProcessData(message, node));
        } catch (Exception e) {
            Log.error(e);
        }
    }

    public void shutdown() {
        this.processorManager.shutdown();
    }

    @Override
    public Result<List<String>> broadcast(BaseMessage message, Node excludeNode, boolean
aysn, int percent) {
        BroadcastResult result = networkService.sendToAllNode(message, excludeNode, aysn,
percent);
        return getNodeIdListResult(result);
    }

    @Override
    public Result sendToNode(BaseMessage message, Node node, boolean aysn) {
```

```
        BroadcastResult result = networkService.sendToNode(message, node, aysn);
        if (!result.isSuccess()) {
            Log.error("send to node fail reason: " + result.getErrorCode().getMsg() + ":::" +
node.getId());
        }

        return new Result(result.isSuccess(), result.getErrorCode(), null);
    }

    @Override
    public Result<? extends BaseMessage> getMessageInstance(short moduleId, int type) {
        Class<? extends BaseMessage> clazz = MessageManager.getMessage(moduleId, type);
        if (null == clazz) {
            return Result.getFailed(MessageBusErrorCode.UNKOWN_MSG_TYPE);
        }
        BaseMessage message = null;
        try {
            message = clazz.newInstance();
        } catch (InstantiationException e) {
            Log.error(e);
            return Result.getFailed(MessageBusErrorCode.INSTANTIATION_EXCEPTION);
        } catch (IllegalAccessException e) {
            Log.error(e);
            return Result.getFailed(MessageBusErrorCode.ILLEGAL_ACCESS_EXCEPTION);
        }

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

    private Result<List<String>> getNodeIdListResult(BroadcastResult result) {
        List<String> list = new ArrayList<>();
        if (!result.isSuccess() || result.getBroadcastNodes() == null ||
result.getBroadcastNodes().isEmpty()) {
            return Result.getFailed(result.getErrorCode()).setData(list);
        }
        for (Node node : result.getBroadcastNodes()) {
            list.add(node.getId());
        }
        Result rs = new Result();
        rs.setSuccess(true);
        rs.setData(list);
        return rs;
```

```
    }
}

7:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-
base\src\test\java\io\nuls\message\bus\module\MessageBusModuleBootstrapTest.java
 */

package io.nuls.message.bus.module;

import io.nuls.message.bus.service.MessageBusService;
import io.nuls.message.bus.service.impl.MessageBusServiceImpl;
import org.junit.Before;
import org.junit.Test;
import java.lang.reflect.Field;
import static org.junit.Assert.*;

public class MessageBusModuleBootstrapTest {

    private MessageBusModuleBootstrap messageBusModuleBootstrap = new
MessageBusModuleBootstrap();

    /**
     * MessageBusModuleBootstrap MessageBusService
     * @throws Exception
     */
    @Before
    public void before() throws Exception{
        MessageBusService messageBusService = new MessageBusServiceImpl();
        Class processorManagerClass = messageBusModuleBootstrap.getClass();
        Field messageBusServiceField =
processorManagerClass.getDeclaredField("messageBusService");
        messageBusServiceField.setAccessible(true);
        assertNull(messageBusServiceField.get(messageBusModuleBootstrap));
        messageBusServiceField.set(messageBusModuleBootstrap, messageBusService);
        assertNotNull(messageBusServiceField.get(messageBusModuleBootstrap));
    }

    @Test
    public void start() {
        messageBusModuleBootstrap.start();
    }
```

```
    @Test
    public void shutdown() {
        messageBusModuleBootstrap.shutdown();
    }

    @Test
    public void destroy() {
        messageBusModuleBootstrap.destroy();
    }
}

8:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-
base\src\test\java\io\nuls\message\bus\service\impl\MessageBusServiceImplTest.java
 */

package io.nuls.message.bus.service.impl;

import TestHandler.BlockMessageHandler;
import io.nuls.kernel.model.Result;
import io.nuls.message.bus.handler.intf.NulsMessageHandler;
import io.nuls.message.bus.manager.DispatchManager;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.network.service.NetworkService;
import io.nuls.network.service.impl.NetworkServiceImpl;
import io.nuls.protocol.message.BlockMessage;
import org.junit.Before;
import org.junit.Test;

import java.lang.reflect.Field;
import java.util.List;
import java.util.Map;
import java.util.Set;

import static org.junit.Assert.*;

public class MessageBusServiceImplTest {

    private MessageBusService messageBusService = new MessageBusServiceImpl();

    private NulsMessageHandler messageHandler = null;
```

```
    private String handlerId = null;

    private Class block = null;

    @Before
    public void before() throws Exception{
        subscribe();

        // MessageBusModuleBootstrap MessageBusService
        NetworkService networkService = new NetworkServiceImpl();
        Field networkServiceField =
messageBusService.getClass().getDeclaredField("networkService");
        networkServiceField.setAccessible(true);
        assertNull(networkServiceField.get(messageBusService));
        networkServiceField.set(messageBusService, networkService);
        assertNotNull(networkServiceField.get(messageBusService));

    }

    private void subscribe(){
        block = BlockMessage.class;
        messageHandler = new BlockMessageHandler();
        handlerId = messageBusService.subscribeMessage(block, messageHandler);
    }

    /**
     *
     * handlerMap
     * messageHandlerMapping
     * Validate the subscription's return value
     * Verify the value of the handlerMap & messageHandlerMapping after subscribing
     */
    @Test
    public void subscribeMessage() throws Exception {
        subscribe();
        assertNotNull(handlerId);
        Field field = messageBusService.getClass().getDeclaredField("processorManager");
        field.setAccessible(true);
        DispatchManager processorManager = (DispatchManager) field.get(messageBusService);

        //handlerMap
        Class processorManagerClass = processorManager.getClass();
```

```
        Field handlerMapField = processorManagerClass.getDeclaredField("handlerMap");
        handlerMapField.setAccessible(true);
        Map<String, NulsMessageHandler> handlerMap = (Map<String, NulsMessageHandler>)
handlerMapField.get(processorManager);
        assertNotNull(handlerMap.get(handlerId));
        assertEquals(handlerMap.get(handlerId), messageHandler);

        //messageHandlerMapping
        Field messageHandlerMappingField =
processorManagerClass.getDeclaredField("messageHandlerMapping");
        messageHandlerMappingField.setAccessible(true);
        Map<Class, Set<String>> messageHandlerMapping = (Map<Class, Set<String>>)
messageHandlerMappingField.get(processorManager);
        assertNotNull(messageHandlerMapping.get(block));
        assertTrue(messageHandlerMapping.get(block).contains(handlerId));
    }

    /**
     *
     * handlerMap
     * Verify the value in handlerMap after unsubscribing
     * @throws Exception
     */
    @Test
    public void unsubscribeMessage() throws Exception {
        messageBusService.unsubscribeMessage(handlerId);

        Field field = messageBusService.getClass().getDeclaredField("processorManager");
        field.setAccessible(true);
        DispatchManager processorManager = (DispatchManager) field.get(messageBusService);

        //handlerMap
        Class processorManagerClass = processorManager.getClass();
        Field handlerMapField = processorManagerClass.getDeclaredField("handlerMap");
        handlerMapField.setAccessible(true);
        Map<String, NulsMessageHandler> handlerMap = (Map<String, NulsMessageHandler>)
handlerMapField.get(processorManager);
        assertNull(handlerMap.get(handlerId));

    }

    @Test
```

```
    public void receiveMessage() {
        BlockMessage blockMessage = new BlockMessage();
        Node node = new Node("192.168.1.90",8003,1);
        messageBusService.receiveMessage(blockMessage, node);
    }

    /**
     *
     * broadcast to nodes except "excludeNode"
     */
    @Test
    public void broadcastAndCache() {
        BlockMessage blockMessage = new BlockMessage();
        Node node = new Node("192.168.1.90",8003,1);
        boolean aysn = true;
        Result<List<String>> result =  messageBusService.broadcast(blockMessage, node,
aysn,100);
        assertTrue(result.isSuccess());
        assertTrue(result.getData().size()>0);
    }

    /**
     * send msg to one node
     */
    @Test
    public void sendToNode() {
        BlockMessage blockMessage = new BlockMessage();
        Node node = new Node("192.168.1.90",8003,1);
        boolean aysn = true;
        assertTrue(messageBusService.sendToNode(blockMessage, node, aysn).isSuccess());
    }
}
```

9:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\base\message-bus-base\src\test\java\TestHandler\BlockMessageHandler.java

10:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-bus\src\main\java\io\nuls\message\bus\constant\MessageBusConstant.java
*/

```
package io.nuls.message.bus.constant;
```

```
import io.nuls.kernel.constant.NulsConstant;

/**
 *
 * The relevant constants of the message-bus and some general constants are defined here.
 * @author: Charlie
 */
public interface MessageBusConstant extends NulsConstant {


    /**
     * The module id of the message-bus module
     */
    short MODULE_ID_MESSAGE_BUS = 6;

    /**
     * The name of the disruptor
     */
    String DISRUPTOR_NAME = "nuls-processing";

    /**
     *
     * The default number of threads in the thread pool
     */
    int THREAD_COUNT = 2 * Runtime.getRuntime().availableProcessors();

    /**
     *
     * The name of the thread pool
     */
    String THREAD_POOL_NAME = "nuls-process-dispatcher";

    /**
     * The default size of ringBuffer
     */
    int DEFAULT_RING_BUFFER_SIZE = 1 << 20;

    /**
     * hash
     */
    short MSG_TYPE_COMMON_MSG_HASH_MSG = 1;
```

```
    /**
     *
     * The message type is the message to get the message body
     */
    short MSG_TYPE_GET_MSG_BODY_MSG = 2;
}
```

11:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-bus\src\main\java\io\nuls\message\bus\constant\MessageBusErrorCode.java

```
 */

package io.nuls.message.bus.constant;


import io.nuls.kernel.constant.ErrorCode;
import io.nuls.kernel.constant.KernelErrorCode;

/**
 * @author: Charlie
 */
public interface MessageBusErrorCode extends KernelErrorCode {

    ErrorCode UNKOWN_MSG_TYPE= ErrorCode.init("60001");
}
```

12:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-bus\src\main\java\io\nuls\message\bus\filter\NulsMessageFilter.java

```
 */

package io.nuls.message.bus.filter;

import io.nuls.protocol.message.base.BaseMessage;

/**
 * Nuls
 * The Nuls event filter.
 *
 * @author: Charlie
 */
public interface NulsMessageFilter<T extends BaseMessage> {

    void doFilter(T data, NulsMessageFilterChain chain);
```

```
}

13:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-
bus\src\main\java\io\nuls\message\bus\filter\NulsMessageFilterChain.java
 */

package io.nuls.message.bus.filter;

import io.nuls.protocol.message.base.BaseMessage;

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

/**
 * @author: Charlie
 */
public class NulsMessageFilterChain {
    private List<NulsMessageFilter> list = new ArrayList<>();
    private ThreadLocal<Integer> index = new ThreadLocal<>();

    public boolean startDoFilter(BaseMessage message) {
        index.set(-1);
        doFilter(message);
        boolean result = index.get() == list.size();
        index.remove();
        return result;

    }

    public void doFilter(BaseMessage message) {
        index.set(1 + index.get());
        if (index.get() == list.size()) {
            return;
        }
        NulsMessageFilter filter = list.get(index.get());
        filter.doFilter(message, this);
    }

    public void addFilter(NulsMessageFilter<? extends BaseMessage> filter) {
        list.add(0, filter);
    }
}
```

14:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-bus\src\main\java\io\nuls\message\bus\handler\AbstractMessageHandler.java
*/

```
package io.nuls.message.bus.handler;

import io.nuls.kernel.context.NulsContext;
import io.nuls.message.bus.filter.NulsMessageFilter;
import io.nuls.message.bus.filter.NulsMessageFilterChain;
import io.nuls.message.bus.handler.intf.NulsMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.protocol.message.base.BaseMessage;

/**
 * ()
 * Message cmd implementation class (abstract)
 * @author: Charlie
 */
public abstract class AbstractMessageHandler<T extends BaseMessage> implements NulsMessageHandler<T> {

    protected MessageBusService messageBusService = NulsContext.getServiceBean(MessageBusService.class);

    private NulsMessageFilterChain filterChain = new NulsMessageFilterChain();

    @Override
    public void addFilter(NulsMessageFilter<T> filter) {
        filterChain.addFilter(filter);
    }

    @Override
    public NulsMessageFilterChain getFilterChian() {
        return filterChain;
    }
}
```

15:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-bus\src\main\java\io\nuls\message\bus\handler\intf\NulsMessageHandler.java
*/

```
package io.nuls.message.bus.handler.intf;

import io.nuls.kernel.exception.NulsException;
import io.nuls.message.bus.filter.NulsMessageFilter;
import io.nuls.message.bus.filter.NulsMessageFilterChain;
import io.nuls.network.model.Node;
import io.nuls.protocol.message.base.BaseMessage;

/**
 *
 * @author: Charlie
 */
public interface NulsMessageHandler<T extends BaseMessage> {

    /**
     *
     * add a filter
     *
     * @param filter
     */
    void addFilter(NulsMessageFilter<T> filter);

    /**
     *
     * Get a FilterChain
     * @return NulsMessageFilterChain
     */
    NulsMessageFilterChain getFilterChian();

    void onMessage(T message, Node fromNode) throws NulsException;
}

16:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-
bus\src\main\java\io\nuls\message\bus\manager\MessageManager.java
 */

package io.nuls.message.bus.manager;

import io.nuls.protocol.message.base.BaseMessage;

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

```
/**
 * @author: Niels Wang
 */
public class MessageManager {

    private static final Map<String, Class<? extends BaseMessage>> MESSAGE_MAP = new
HashMap<>();

    private static void putMessage(short moduleId, int type, Class<? extends BaseMessage>
msgClass) {
        MESSAGE_MAP.put(moduleId + "_" + type, msgClass);
    }

    public static Class<? extends BaseMessage> getMessage(short moduleId, int type) {
        return MESSAGE_MAP.get(moduleId + "_" + type);
    }

    public static void putMessage(Class<? extends BaseMessage> msgClass) {
        try {
            BaseMessage message = msgClass.newInstance();
            putMessage(message.getHeader().getModuleId(), message.getHeader().getMsgType(),
msgClass);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

17:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-
bus\src\main\java\io\nuls\message\bus\module\AbstractMessageBusModule.java
 */

package io.nuls.message.bus.module;

import io.nuls.kernel.module.BaseModuleBootstrap;
import io.nuls.message.bus.constant.MessageBusConstant;

/**
 * @author: Charlie
 */
public abstract class AbstractMessageBusModule extends BaseModuleBootstrap {
```

```
    public AbstractMessageBusModule(){
        super(MessageBusConstant.MODULE_ID_MESSAGE_BUS);
    }
}

18:F:\git\coin\nuls\nuls-1.1.3\nuls\message-bus-module\message-
bus\src\main\java\io\nuls\message\bus\service\MessageBusService.java
 */

package io.nuls.message.bus.service;

import io.nuls.kernel.model.Result;
import io.nuls.message.bus.handler.intf.NulsMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.List;

/**
 *
 * The message-bus module provides the definition of the external service interface
 *
 * @author: Charlie
 */
public interface MessageBusService {

    /**
     *
     * Subscribe to message
     *
     * @param messageClass    class
     * @param messageClass   The class object that needs to subscribe to the message.
     * @param messageHandler
     * @param messageHandler The message message
     * @return The id of the subscription message.
     */
    String subscribeMessage(Class<? extends BaseMessage> messageClass,
NulsMessageHandler<? extends BaseMessage> messageHandler);



    /**
```

```
     *
     * unsubscribe
     *
     * @param subscribeId id.
     * @param subscribeId The id of the message message.
     */
    void unsubscribeMessage(String subscribeId);

    /**
     * ,
     * Receive the message and place the message on the message bus.
     *
     * @param message
     * @param message Received message.
     * @param node    , .
     * @param node    The message comes as to which node.
     */
    void receiveMessage(BaseMessage message, Node node);

    /**
     *
     * broadcast to nodes except "excludeNode"
     *
     * @param message     The message was broadcast.
     * @param excludeNode  The node that is not passed.
     * @param aysn         Asynchronous execution
     * @return Return all broadcasted node id list
     */
    Result<List<String>> broadcast(BaseMessage message, Node excludeNode, boolean aysn, int
percent);


    /**
     *
     * send msg to one node
     *
     * @param message The message you want to sent
     * @param node    The node that received the message
     * @param aysn     Asynchronous execution
     * @return Return whether sent successfully
     */
    Result sendToNode(BaseMessage message, Node node, boolean aysn);
```

```
    /**
     *
     * Instantiate a message object based on message type and module identity.
     */
    Result<? extends BaseMessage> getMessageInstance(short moduleId, int type);
}

19:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\cache\NodeCacheManager.java
 */
package io.nuls.network.cache;

import io.nuls.cache.CacheMap;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.protocol.message.P2PNodeBody;

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

public class NodeCacheManager {

    private static NodeCacheManager instance = new NodeCacheManager();

    private NodeCacheManager() {

    }

    public static NodeCacheManager getInstance() {
        return instance;
    }

    //5
    private CacheMap<String, P2PNodeBody> cacheNodeMap = new
CacheMap<>(NetworkConstant.CACHE_P2P_NODE, 8, String.class, P2PNodeBody.class, 60 *
5, 0, null);

    private CacheMap<String, Set<String>> cacheIpMap = new
CacheMap<>(NetworkConstant.CACHE_P2P_IP, 2, String.class, HashSet.class, 60, 0, null);

    public void cacheNode(P2PNodeBody body) {
        cacheNodeMap.put(body.getId(), body);
```

```
    }

    public P2PNodeBody getNode(String id) {
        return cacheNodeMap.get(id);
    }

    public void cacheIpSet(Set ipSet) {
        cacheIpMap.put("ipSet", ipSet);
    }

    public Set<String> getIpSet() {
        return cacheIpMap.get("ipSet");
    }
}

20:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\connection\netty\ClientChannelHandler.java
 */

package io.nuls.network.connection.netty;

import io.netty.buffer.ByteBuf;
import io.netty.channel.Channel;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.channel.socket.SocketChannel;
import io.netty.util.Attribute;
import io.netty.util.AttributeKey;
import io.nuls.core.tools.log.Log;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.manager.ConnectionManager;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.Node;
import io.nuls.network.util.SendNodeInfoThread;

import java.io.IOException;

public class ClientChannelHandler extends ChannelInboundHandlerAdapter {

    private NodeManager nodeManager = NodeManager.getInstance();

    private ConnectionManager connectionManager = ConnectionManager.getInstance();
```

```
    private AttributeKey<Node> key = AttributeKey.valueOf("node");

    private NetworkParam networkParam = NetworkParam.getInstance();

    public ClientChannelHandler() {

    }

    @Override
    public void channelRegistered(ChannelHandlerContext ctx) throws Exception {
        super.channelRegistered(ctx);
        Attribute<Node> nodeAttribute = ctx.channel().attr(key);
        Node node = nodeAttribute.get();
        node.setCanConnect(false);
    }

    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
        super.channelActive(ctx);
        Channel channel = ctx.channel();
        Attribute<Node> nodeAttribute = channel.attr(key);
        Node node = nodeAttribute.get();

        SocketChannel socketChannel = (SocketChannel) ctx.channel();
        String remoteIP = socketChannel.remoteAddress().getHostString();
        //
        if (networkParam.getLocalIps().contains(remoteIP) &&
!nodeManager.isSeedNode(remoteIP)) {
            SendNodeInfoThread.getInstance().start();
            channel.close();
        } else {
            //
            node.setCanConnect(true);
            node.setFailCount(0);
            boolean result = nodeManager.processConnectedNode(node, channel);
            if (!result) {
                channel.close();
            }
        }
    }
```

```
    @Override
    public void channelInactive(ChannelHandlerContext ctx) throws Exception {
        super.channelInactive(ctx);
        Attribute<Node> nodeAttribute = ctx.channel().attr(key);
        Node node = nodeAttribute.get();
        if (node != null) {
            nodeManager.removeNode(node);
        } else {
            SocketChannel socketChannel = (SocketChannel) ctx.channel();
            String remoteIP = socketChannel.remoteAddress().getHostString();
            int port = socketChannel.remoteAddress().getPort();
            Log.info("-----------------client channelInactive  node is null -----------------" + remoteIP + ":" +
port);
        }
    }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        try {
            Attribute<Node> nodeAttribute = ctx.channel().attr(key);
            Node node = nodeAttribute.get();

            if (node != null) {
                if (node.isAlive()) {
                    ByteBuf buf = (ByteBuf) msg;
                    try {
                        connectionManager.receiveMessage(buf, node);
                    } finally {
                        buf.release();
                    }
                    //                NetworkThreadPool.doRead(buf, node);
                }
            } else {
                SocketChannel socketChannel = (SocketChannel) ctx.channel();
                String remoteIP = socketChannel.remoteAddress().getHostString();
                int port = socketChannel.remoteAddress().getPort();
                Log.info("-----------------client channelRead  node is null -----------------" + remoteIP + ":" +
port);
            }
        } catch (Exception e) {
            throw e;
        }
```

```
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        if (!(cause instanceof IOException)) {
            //Log.error(cause);
            Log.error("======================");
        }
        ctx.channel().close();
    }

}

21:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\connection\netty\codec\NulsNettyDecoder.java
 */
package io.nuls.network.connection.netty.codec;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.*;
import io.nuls.core.tools.log.Log;
import io.nuls.network.constant.NetworkParam;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.nio.ByteOrder;
import java.util.List;

import static io.nuls.network.constant.NetworkConstant.MAX_FRAME_LENGTH;

/**
 * @author tangyi
 * @date 2018/8/29
 * @description ByteToMessageDecoderLengthFieldBasedFrameDecoder
 * LengthFieldBasedFrameDecoderdecodefinal
 */
public class NulsNettyDecoder extends ByteToMessageDecoder {

    private ByteOrder byteOrder;
    private int maxFrameLength;
    private int lengthFieldOffset;
```

```
    private int lengthFieldLength;
    private int lengthFieldEndOffset;
    private int lengthAdjustment;
    private int initialBytesToStrip;
    private boolean failFast;
    private boolean discardingTooLongFrame;
    private long tooLongFrameLength;
    private long bytesToDiscard;

    /**
     * Nuls default
     * @param maxFrameLength
     */
    public NulsNettyDecoder(int maxFrameLength) {
        this(maxFrameLength, 0, 0, 0, 0, true, ByteOrder.BIG_ENDIAN);
    }

    private NulsNettyDecoder(int maxFrameLength, int lengthFieldOffset, int lengthFieldLength,
                             int lengthAdjustment, int initialBytesToStrip, boolean failFast, ByteOrder
byteOrder) {
        this.maxFrameLength = maxFrameLength;
        this.lengthFieldOffset = lengthFieldOffset;
        this.lengthFieldLength = lengthFieldLength;
        this.lengthAdjustment = lengthAdjustment;
        this.initialBytesToStrip = initialBytesToStrip;
        this.failFast = failFast;
        this.byteOrder = byteOrder;
        this.lengthFieldEndOffset = lengthFieldOffset + lengthFieldLength;

        if (byteOrder == null) {
            throw new NullPointerException("byteOrder");
        }

        if (maxFrameLength <= 0) {
            throw new IllegalArgumentException(
                    "maxFrameLength must be a positive integer: " +
                            maxFrameLength);
        }

        if (lengthFieldOffset < 0) {
            throw new IllegalArgumentException(
                    "lengthFieldOffset must be a non-negative integer: " +
```

```
                    lengthFieldOffset);
        }


        if (lengthFieldOffset > maxFrameLength - lengthFieldLength) {
            throw new IllegalArgumentException(
                    "maxFrameLength (" + maxFrameLength + ") " +
                            "must be equal to or greater than " +
                            "lengthFieldOffset (" + lengthFieldOffset + ") + " +
                            "lengthFieldLength (" + lengthFieldLength + ").");
        }
    }

    @Override
    /*
      decode8magic numberencoder(NulsNettyEncoder)
      encoder(LengthFieldPrepender)

     */
    public final void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws
Exception {
        long sysMagicNumber = NetworkParam.getInstance().getPacketMagic();
        long decodeNumberAtPos0 = in.getUnsignedIntLE(0);
        long decodeNumberAtPos8 = in.getUnsignedIntLE(8);
        Log.info("Decode start: sysMagicNumber=" + sysMagicNumber + ", decodeNumberAtPos0="
+ decodeNumberAtPos0 + ", decodeNumberAtPos8=" + decodeNumberAtPos8);

        Object decoded;
        if (sysMagicNumber == decodeNumberAtPos0) {
            // 0magic number
            Log.info("NEW  VERSION decode!!!!!decodeNumberAtPos0=" + decodeNumberAtPos0);
            //decoded = decode(ctx, in);

            ByteBuf frame = in.retainedDuplicate();
            out.add(frame);
            in.skipBytes(in.readableBytes());
        } else if (sysMagicNumber == decodeNumberAtPos8) {
            // 8magic number
            Log.info("OLD  VERSION decode!!!!!decodeNumberAtPos8=" + decodeNumberAtPos8);

            // decode protected
            Class cls = Class.forName("io.netty.handler.codec.LengthFieldBasedFrameDecoder");
```

```
            Constructor c = cls.getConstructor(int.class, int.class, int.class, int.class, int.class);
            LengthFieldBasedFrameDecoder preDecode = (LengthFieldBasedFrameDecoder)
c.newInstance(MAX_FRAME_LENGTH, 0, 8, 0, 8);
            Method decodeMethod = cls.getDeclaredMethod("decode", ChannelHandlerContext.class,
ByteBuf.class);
            decodeMethod.setAccessible(true);
            decoded = decodeMethod.invoke(preDecode, ctx, in);

            if (decoded != null) {
                out.add(decoded);
            }
        }
    }

    /*
    protected Object decode(ChannelHandlerContext ctx, ByteBuf in) throws Exception {

        //discardingTooLongFrame
        if (discardingTooLongFrame) {
            long bytesToDiscard = this.bytesToDiscard;
            //Math.min(bytesToDiscard, in.readableBytes())
            //bytesToDiscard
            int localBytesToDiscard = (int) Math.min(bytesToDiscard, in.readableBytes());
            //ByteBufskipBytes
            in.skipBytes(localBytesToDiscard);
            //bytesToDiscard
            bytesToDiscard -= localBytesToDiscard;
            this.bytesToDiscard = bytesToDiscard;
            //discardingTooLongFrame
            failIfNecessary(false);
        }

        if (in.readableBytes() < lengthFieldEndOffset) {
            return null;
        }

        int actualLengthFieldOffset = in.readerIndex() + lengthFieldOffset;
        //lengthFieldOffset
        //6
        //1ByteBufgetUnsignedByte
        //2ByteBufgetUnsignedShort
        //3ByteBufgetUnsignedMedium
```

```
        //4ByteBufgetUnsignedInt
        //8ByteBufgetLong
        //DecoderException
        //0
        long frameLength = getUnadjustedFrameLength(in, actualLengthFieldOffset,
lengthFieldLength, byteOrder);

        //0lengthFieldEndOffsetCorruptedFrameException
        if (frameLength < 0) {
            in.skipBytes(lengthFieldEndOffset);
            throw new CorruptedFrameException(
                    "negative pre-adjustment length field: " + frameLength);
        }

        //lengthFieldEndOffsetlengthAdjustment
        frameLength += lengthAdjustment + lengthFieldEndOffset;
        //lengthFieldEndOffsetCorruptedFrameException
        if (frameLength < lengthFieldEndOffset) {
            in.skipBytes(lengthFieldEndOffset);
            throw new CorruptedFrameException(
                    "Adjusted frame length (" + frameLength + ") is less " +
                            "than lengthFieldEndOffset: " + lengthFieldEndOffset);
        }

        //ByteBuf
        //discardingTooLongFrame
        if (frameLength > maxFrameLength) {
            //frameLengthByteBuf
            //discard
            //discardingTooLongFrametrue
            //failIfNecessary
            long discard = frameLength - in.readableBytes();
            tooLongFrameLength = frameLength;

            if (discard < 0) {
                // buffer contains more bytes then the frameLength so we can discard all now
                in.skipBytes((int) frameLength);
            } else {
                // Enter the discard mode and discard everything received so far.
                discardingTooLongFrame = true;
                bytesToDiscard = discard;
                in.skipBytes(in.readableBytes());
```

```
            }
            failIfNecessary(true);
            return null;
        }

        //frameLengthI/O
        // never overflows because it's less than maxFrameLength
        int frameLengthInt = (int) frameLength;
        if (in.readableBytes() < frameLengthInt) {
            return null;
        }

        //frameLengthCorruptedFrameException
        if (initialBytesToStrip > frameLengthInt) {
            in.skipBytes(frameLengthInt);
            throw new CorruptedFrameException(
                    "Adjusted frame length (" + frameLength + ") is less " +
                            "than initialBytesToStrip: " + initialBytesToStrip);
        }

        //ByteBufskipBytesByteBuf
        in.skipBytes(initialBytesToStrip);

        // extract frame
        int readerIndex = in.readerIndex();
        int actualFrameLength = frameLengthInt - initialBytesToStrip;
        //extractFrame
        //ByteBufByteBufByteBuf
        //ByteBuf+actualFrameLength
        ByteBuf frame = extractFrame(ctx, in, readerIndex, actualFrameLength);
        in.readerIndex(readerIndex + actualFrameLength);
        return frame;
    }

    private void failIfNecessary(boolean firstDetectionOfTooLongFrame) {
        if (bytesToDiscard == 0) {
            // Reset to the initial state and tell the handlers that
            // the frame was too large.
            long tooLongFrameLength = this.tooLongFrameLength;
            this.tooLongFrameLength = 0;
            discardingTooLongFrame = false;
            if (!failFast || firstDetectionOfTooLongFrame) {
```

```
                fail(tooLongFrameLength);
            }
        } else {
            // Keep discarding and notify handlers if necessary.
            if (failFast && firstDetectionOfTooLongFrame) {
                fail(tooLongFrameLength);
            }
        }
    }

    private long getUnadjustedFrameLength(ByteBuf buf, int offset, int length, ByteOrder order) {
        //noinspection deprecation
        buf = buf.order(order);
        long frameLength;
        Log.info("switch, length=" + length);
        switch (length) {
            case 0:
                frameLength = 0;
                break;
            case 1:
                frameLength = buf.getUnsignedByte(offset);
                break;
            case 2:
                frameLength = buf.getUnsignedShort(offset);
                break;
            case 3:
                frameLength = buf.getUnsignedMedium(offset);
                break;
            case 4:
                frameLength = buf.getUnsignedInt(offset);
                break;
            case 8:
                frameLength = buf.getLong(offset);
                break;
            default:
                throw new DecoderException(
                        "unsupported lengthFieldLength: " + lengthFieldLength + " (expected: 1, 2, 3, 4, 8,
or 0)");
        }
        return frameLength;
    }
```

```
    private ByteBuf extractFrame(ChannelHandlerContext ctx, ByteBuf buffer, int index, int length) {
        return buffer.retainedSlice(index, length);
    }

    private void fail(long frameLength) {
        if (frameLength > 0) {
            throw new TooLongFrameException(
                    "Adjusted frame length exceeds " + maxFrameLength +
                            ": " + frameLength + " - discarded");
        } else {
            throw new TooLongFrameException(
                    "Adjusted frame length exceeds " + maxFrameLength +
                            " - discarding");
        }
    }
    */
}

22:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\connection\netty\codec\NulsNettyEncoder.java
 */
package io.nuls.network.connection.netty.codec;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.MessageToMessageEncoder;
import io.nuls.core.tools.log.Log;

import java.nio.ByteOrder;
import java.util.List;

/**
 * @author tangyi
 * @date 2018/8/28
 * @description
 */
public class NulsNettyEncoder extends MessageToMessageEncoder<ByteBuf> {

    private ByteOrder byteOrder;
    private int lengthFieldLength;
    private boolean lengthIncludesLengthFieldLength;
    private int lengthAdjustment;
```

```
    public NulsNettyEncoder(int lengthFieldLength) {
        this(ByteOrder.BIG_ENDIAN, lengthFieldLength, false, 0);
    }

    private NulsNettyEncoder(ByteOrder byteOrder, int lengthFieldLength, boolean
lengthIncludesLengthFieldLength, int lengthAdjustment) {
        this.byteOrder = byteOrder;
        this.lengthFieldLength = lengthFieldLength;
        this.lengthIncludesLengthFieldLength = lengthIncludesLengthFieldLength;
        this.lengthAdjustment = lengthAdjustment;
    }

    @SuppressWarnings("deprecation")
    @Override
    public final void encode(ChannelHandlerContext ctx, ByteBuf msg, List<Object> out) {
        int length = msg.readableBytes() + lengthAdjustment;
        if (lengthIncludesLengthFieldLength) {
            length += lengthFieldLength;
        }

        if (length < 0) {
            throw new IllegalArgumentException(
                    "Adjusted frame length (" + length + ") is less than zero");
        }

        switch (lengthFieldLength) {
            case 0:
                Log.info("NEW  VERSION encode!!!!!");
                break;
            case 1:
                if (length >= 256) {
                    throw new IllegalArgumentException(
                            "length does not fit into a byte: " + length);
                }
                out.add(ctx.alloc().buffer(1).order(byteOrder).writeByte((byte) length));
                break;
            case 2:
                if (length >= 65536) {
                    throw new IllegalArgumentException(
                            "length does not fit into a short integer: " + length);
                }
```

```
            out.add(ctx.alloc().buffer(2).order(byteOrder).writeShort((short) length));
            break;
        case 3:
            if (length >= 16777216) {
                throw new IllegalArgumentException(
                        "length does not fit into a medium integer: " + length);
            }
            out.add(ctx.alloc().buffer(3).order(byteOrder).writeMedium(length));
            break;
        case 4:
            out.add(ctx.alloc().buffer(4).order(byteOrder).writeInt(length));
            break;
        case 8:
            out.add(ctx.alloc().buffer(8).order(byteOrder).writeLong(length));
            break;
        default:
            throw new Error("should not reach here");
      }
      out.add(msg.retain());
   }
}

23:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\connection\netty\HeartbeatServerHandler.java
 */

package io.nuls.network.connection.netty;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.channel.socket.SocketChannel;
import io.netty.handler.timeout.IdleState;
import io.netty.handler.timeout.IdleStateEvent;
import io.nuls.core.tools.log.Log;

/**
 * @desription:
 * @author: PierreLuo
 */
public class HeartbeatServerHandler extends ChannelInboundHandlerAdapter {

    @Override
```

```
    public void userEventTriggered(ChannelHandlerContext ctx, Object evt) throws Exception {

        if (evt instanceof IdleStateEvent) {  // 2
//            IdleStateEvent event = (IdleStateEvent) evt;
//            String type = "";
//            if (event.state() == IdleState.READER_IDLE) {
//                type = "read idle";
//            } else if (event.state() == IdleState.WRITER_IDLE) {
//                type = "write idle";
//            } else if (event.state() == IdleState.ALL_IDLE) {
//                type = "all idle";
//            }
//            SocketChannel channel = (SocketChannel) ctx.channel();
//            Log.info(ctx.channel().remoteAddress() + "timeout type" + type);
//            Log.info(" ---------------------- HeartbeatServerHandler ---------------------- ");
//            Log.info("localInfo: "+channel.localAddress().getHostString()+":" +
channel.localAddress().getPort());
//            Log.info("remoteInfo: "+channel.remoteAddress().getHostString()+":" +
channel.remoteAddress().getPort());
            ctx.channel().close();

        } else {
            super.userEventTriggered(ctx, evt);
        }
    }
}

24:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\connection\netty\NettyClient.java
 */

package io.nuls.network.connection.netty;

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelFutureListener;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioSocketChannel;
import io.netty.util.AttributeKey;
```

```
import io.nuls.core.tools.log.Log;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.Node;

import static io.nuls.network.constant.NetworkConstant.CONNETCI_TIME_OUT;


public class NettyClient {

    public static EventLoopGroup worker = new NioEventLoopGroup();

    Bootstrap boot;

    private SocketChannel socketChannel;

    private Node node;

    private NodeManager nodeManager = NodeManager.getInstance();

    public NettyClient(Node node) {
        this.node = node;
        boot = new Bootstrap();

        AttributeKey<Node> key = null;
        synchronized (NettyClient.class) {
            if (AttributeKey.exists("node")) {
                key = AttributeKey.valueOf("node");
            } else {
                key = AttributeKey.newInstance("node");
            }
        }
        boot.attr(key, node);
        boot.group(worker)
                .channel(NioSocketChannel.class)
//                .option(ChannelOption.SO_BACKLOG, 128)
                .option(ChannelOption.TCP_NODELAY, true)          //Send messages immediately
                .option(ChannelOption.SO_KEEPALIVE, true)
                .option(ChannelOption.SO_SNDBUF, 128 * 1024)
                .option(ChannelOption.SO_RCVBUF, 128 * 1024)
                .option(ChannelOption.CONNECT_TIMEOUT_MILLIS, CONNETCI_TIME_OUT)
                .handler(new NulsChannelInitializer<>(new ClientChannelHandler()));
    }
```

```
    public void start() {
        try {
            ChannelFuture future = boot.connect(node.getIp(), node.getSeverPort()).addListener(new
ChannelFutureListener() {
                @Override
                public void operationComplete(ChannelFuture future) throws Exception {
                    if (future.isSuccess()) {
                        socketChannel = (SocketChannel) future.channel();
                    } else {
//                        Log.error("Client connect to host error: " + future.cause() + ", remove node: " +
node.getId());
//                        System.out.println("Client connect fail: " + future.cause() + ", remove node: " +
node.getId());
                        nodeManager.removeNode(node.getId());
                    }
                }
            });
            future.channel().closeFuture().awaitUninterruptibly();
        } catch (Exception e) {
            //maybe time out or refused or something
            if (socketChannel != null) {
                socketChannel.close();
            }
            Log.error("Client start exception:" + e.getMessage() + ", remove node: " + node.getId());
            nodeManager.removeNode(node.getId());
        }
    }

}

25:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\connection\netty\NettyServer.java
 */

package io.nuls.network.connection.netty;

import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
```

```
import io.netty.channel.socket.nio.NioServerSocketChannel;

public class NettyServer {

    private int port;

    private ServerBootstrap serverBootstrap;
    private static EventLoopGroup boss;
    private static EventLoopGroup worker;

    public NettyServer(int port) {
        this.port = port;
    }

    public void init() {
        boss = new NioEventLoopGroup();
        worker = new NioEventLoopGroup();
        serverBootstrap = new ServerBootstrap();
        serverBootstrap.group(boss, worker)
                .channel(NioServerSocketChannel.class)
//                .childOption(ChannelOption.SO_BACKLOG, 128)
                .childOption(ChannelOption.TCP_NODELAY, true)           //Send messages
immediately
                .childOption(ChannelOption.SO_KEEPALIVE, true)
                .childOption(ChannelOption.SO_SNDBUF, 128 * 1024)
                .childOption(ChannelOption.SO_RCVBUF, 128 * 1024)
                .childHandler(new NulsChannelInitializer<>(new ServerChannelHandler()));
    }

    public void start() throws InterruptedException {
        try {
            // Start the server.
            ChannelFuture future = serverBootstrap.bind(port).sync();
            // Wait until the server socket is closed.
            future.channel().closeFuture().sync();
        } catch (InterruptedException e) {
            throw e;
        } finally {
            // Shut down all event loops to terminate all threads.
            boss.shutdownGracefully();
            worker.shutdownGracefully();
        }
```

```
    }

    public void shutdown() {
        boss.shutdownGracefully();
        worker.shutdownGracefully();
    }
}

26:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\connection\netty\NioChannelMap.java
 */

package io.nuls.network.connection.netty;

import io.netty.channel.socket.SocketChannel;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class NioChannelMap {

    private static Map<String, SocketChannel> map = new ConcurrentHashMap<>();

    public static void add(String channelId, SocketChannel channel) {
        map.put(channelId, channel);
    }

    public static SocketChannel get(String channelId) {
        return map.get(channelId);
    }

    public static void remove(String channelId) {
        map.remove(channelId);
    }

    public static boolean containsKey(String channelId) {
        return map.containsKey(channelId);
    }

    public static Map<String, SocketChannel> channels() {
        return map;
    }
```

}

27:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\connection\netty\NulsChannelInitializer.java
*/

```
package io.nuls.network.connection.netty;

import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelPipeline;
import io.netty.channel.socket.SocketChannel;
import io.netty.handler.timeout.IdleStateHandler;

import java.util.concurrent.TimeUnit;

import static io.nuls.network.constant.NetworkConstant.*;

public class NulsChannelInitializer<T extends ChannelInboundHandlerAdapter> extends ChannelInitializer<SocketChannel> {



    private T t;

    public NulsChannelInitializer(T t) {
        this.t = t;
    }

    @Override
    protected void initChannel(SocketChannel socketChannel) throws Exception {
        ChannelPipeline p = socketChannel.pipeline();
        p.addLast("idle", new IdleStateHandler(READ_IDEL_TIME_OUT, WRITE_IDEL_TIME_OUT, ALL_IDEL_TIME_OUT, TimeUnit.SECONDS));
        //p.addLast("decoder", new LengthFieldBasedFrameDecoder(ByteOrder.LITTLE_ENDIAN, MAX_FRAME_LENGTH, 4, 4, 6, 0, true));
        //p.addLast("encoder0", new LengthFieldPrepender(8, false));
        p.addLast("decoder", new NulsMessageDecoder());
        p.addLast("encoder0", new NulsMessageEncoder());
        p.addLast("heartbeat", new HeartbeatServerHandler());
        p.addLast(t);
    }
```

```
}
```

28:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\connection\netty\NulsLengthFieldBasedFrameDecoder.java

```
package io.nuls.network.connection.netty;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.LengthFieldBasedFrameDecoder;

import java.nio.ByteOrder;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/7
 */
public class NulsLengthFieldBasedFrameDecoder extends LengthFieldBasedFrameDecoder {

    public NulsLengthFieldBasedFrameDecoder(int maxFrameLength, int lengthFieldOffset, int lengthFieldLength, int lengthAdjustment, int initialBytesToStrip) {
        super(maxFrameLength, lengthFieldOffset, lengthFieldLength, lengthAdjustment, initialBytesToStrip);
    }

    public NulsLengthFieldBasedFrameDecoder(ByteOrder byteOrder, int maxFrameLength, int lengthFieldOffset, int lengthFieldLength, int lengthAdjustment, int initialBytesToStrip, boolean failFast) {
        super(byteOrder, maxFrameLength, lengthFieldOffset, lengthFieldLength, lengthAdjustment, initialBytesToStrip, failFast);
    }

    @Override
    public Object decode(ChannelHandlerContext ctx, ByteBuf in) throws Exception {
        return super.decode(ctx, in);
    }
}
```

29:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\connection\netty\NulsMessageDecoder.java

```
package io.nuls.network.connection.netty;
```

```
import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.ByteToMessageDecoder;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.network.constant.NetworkParam;

import java.nio.ByteOrder;
import java.util.List;

import static io.nuls.network.constant.NetworkConstant.MAX_FRAME_LENGTH;

/**
 * @desription:
 * @author: PierreLuo
 * @date: 2018/8/7
 */
public class NulsMessageDecoder extends ByteToMessageDecoder {

    private NulsLengthFieldBasedFrameDecoder oldDecoder = new
NulsLengthFieldBasedFrameDecoder(MAX_FRAME_LENGTH, 0, 8, 0, 8);
    private NulsLengthFieldBasedFrameDecoder newDecoder = new
NulsLengthFieldBasedFrameDecoder(ByteOrder.LITTLE_ENDIAN, MAX_FRAME_LENGTH, 4, 4,
6, 0, true);



    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) throws
Exception {
        long sysMagicNumber = NetworkParam.getInstance().getPacketMagic();
        long readMagicNumber = in.getUnsignedIntLE(0);
        if (sysMagicNumber == readMagicNumber) {
            //
            Object decoded = newDecoder.decode(ctx, in);
            if (decoded != null) {
                out.add(decoded);
            }
        } else {
            readMagicNumber = in.getUnsignedIntLE(8);
            if (sysMagicNumber == readMagicNumber) {
                //
                Object decoded = oldDecoder.decode(ctx, in);
```

```
            if (decoded != null) {
                out.add(decoded);
            }
        }
    }
}
```

30:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\connection\netty\NulsMessageEncoder.java

```
package io.nuls.network.connection.netty;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.MessageToMessageEncoder;
import io.nuls.kernel.context.NulsContext;

import java.util.List;

/**
 * @author: PierreLuo
 * @date: 2018/7/30
 */
public class NulsMessageEncoder extends MessageToMessageEncoder<ByteBuf> {
    @Override
    protected void encode(ChannelHandlerContext ctx, ByteBuf msg, List<Object> out) throws Exception {
        int length = msg.readableBytes();
        if(NulsContext.MAIN_NET_VERSION > 1) {
            // new protocol
            out.add(msg.retain());
        } else if(NulsContext.MAIN_NET_VERSION == 1){
            // old protocol
            out.add(ctx.alloc().buffer(8).writeLong(length));
            out.add(msg.retain());
        }
    }
}
```

31:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\connection\netty\ServerChannelHandler.java

```
 */
```

```
package io.nuls.network.connection.netty;

import io.netty.buffer.ByteBuf;
import io.netty.channel.Channel;
import io.netty.channel.ChannelHandler;
import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.channel.socket.SocketChannel;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.network.IpUtil;
import io.nuls.kernel.context.NulsContext;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.manager.BroadcastHandler;
import io.nuls.network.manager.ConnectionManager;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.message.HandshakeMessage;
import io.nuls.network.protocol.message.NetworkMessageBody;
import io.nuls.network.util.NetworkThreadPool;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Map;

/**
 * @author Vivi
 */

@ChannelHandler.Sharable
public class ServerChannelHandler extends ChannelInboundHandlerAdapter {

    private NodeManager nodeManager = NodeManager.getInstance();

    private NetworkParam networkParam = NetworkParam.getInstance();

    private BroadcastHandler broadcastHandler = BroadcastHandler.getInstance();

    private ConnectionManager connectionManager = ConnectionManager.getInstance();

    @Override
```

```
    public void channelRegistered(ChannelHandlerContext ctx) throws Exception {
        super.channelRegistered(ctx);
        SocketChannel channel = (SocketChannel) ctx.channel();

        String remoteIP = channel.remoteAddress().getHostString();
        //
        if (networkParam.getLocalIps().contains(remoteIP)) {
//            Log.info("---------------------- ------------------------ " + nodeId);
            ctx.channel().close();
            return;
        }

        //
        // ip
        Map<String, Node> nodeMap = nodeManager.getConnectedNodes();
        for (Node node : nodeMap.values()) {
            if (node.getIp().equals(remoteIP)) {
                if (node.getType() == Node.OUT) {
//                    Log.info("---------------- ip   ------------------" + nodeId);
                    ctx.channel().close();
                    return;
//
//                    String localIP = InetAddress.getLocalHost().getHostAddress();
//                    boolean isLocalServer = IpUtil.judgeLocalIsServer(localIP, remoteIP);
//                    //
//                    if (!isLocalServer) {
//                        //
//                        System.out.println("----------------    ------------------");
//                        ctx.channel().close();
//                        return;
//                    } else {
//                        //
////                        System.out.println("-----------------sever client register each other remove node-----
------------" + node.getId());
//                        nodeManager.removeNode(node.getId());
//                    }
                }
            }
        }

        // if More than 10 in nodes of the same IP, close this channel
        // ip10
```

```
        int count = 0;
        for (Node n : nodeMap.values()) {
            if (n.getIp().equals(remoteIP)) {
                count++;
                if (count >= NetworkConstant.SAME_IP_MAX_COUNT) {
                    ctx.channel().close();
                    return;
                }
            }
        }
    }

    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
        super.channelActive(ctx);

        Channel channel = ctx.channel();
        SocketChannel socketChannel = (SocketChannel) ctx.channel();
//        String nodeId = IpUtil.getNodeId(channel.remoteAddress());
//        System.out.println("---------------------- server channelActive ------------------------- " + nodeId);

        String channelId = ctx.channel().id().asLongText();
//        NioChannelMap.add(channelId, channel);
        Node node = new Node(socketChannel.remoteAddress().getHostString(),
socketChannel.remoteAddress().getPort(), Node.IN);
        node.setStatus(Node.CONNECT);
        boolean success = nodeManager.processConnectedNode(node, channel);
        if (!success) {
            ctx.channel().close();
            return;
        }
        //ip
        NetworkMessageBody body = new
NetworkMessageBody(NetworkConstant.HANDSHAKE_SEVER_TYPE, networkParam.getPort(),
                NulsContext.getInstance().getBestHeight(),
NulsContext.getInstance().getBestBlock().getHeader().getHash(),
                socketChannel.remoteAddress().getHostString());
        HandshakeMessage handshakeMessage = new HandshakeMessage(body);
        broadcastHandler.broadcastToNode(handshakeMessage, node, false);
    }

//    @Override
```

```
//    public void channelInactive(ChannelHandlerContext ctx) throws Exception {
//        super.channelInactive(ctx);
//
//    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) throws Exception {
//        Log.info("----------------- server exceptionCaught --------------------");
        if (!(cause instanceof IOException)) {
            Log.error(cause);
        }
        ctx.channel().close();
    }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
        SocketChannel channel = (SocketChannel) ctx.channel();
        String nodeId = IpUtil.getNodeId(channel.remoteAddress());
        try {
            Node node = nodeManager.getNode(nodeId);
            if (node != null && node.isAlive()) {
                ByteBuf buf = (ByteBuf) msg;
//                NetworkThreadPool.doRead(buf, node);
                try {
                    connectionManager.receiveMessage(buf, node);
                } finally {
                    buf.release();
                }
            }
        } catch (Exception e) {
//            System.out.println(" ---------------------- server channelRead exception---------------------- "
+ nodeId);
            e.printStackTrace();
            throw e;
        }
    }

    @Override
    public void channelUnregistered(ChannelHandlerContext ctx) throws Exception {
        super.channelUnregistered(ctx);
        SocketChannel channel = (SocketChannel) ctx.channel();
        String nodeId = IpUtil.getNodeId(channel.remoteAddress());
```

```
//        Log.info(" ---------------------- server channelInactive ------------------------ " + nodeId);

        Node node = nodeManager.getNode(nodeId);
        if (node != null) {
            nodeManager.removeNode(nodeId);
        }
    }

}

32:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\manager\BroadcastHandler.java
 */

package io.nuls.network.manager;

import io.netty.buffer.Unpooled;
import io.netty.channel.ChannelFuture;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.network.constant.NetworkErrorCode;
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.protocol.message.base.BaseMessage;
import io.nuls.protocol.message.base.MessageHeader;

import java.util.*;

public class BroadcastHandler {

    private static BroadcastHandler instance = new BroadcastHandler();

    private BroadcastHandler() {

    }

    public static BroadcastHandler getInstance() {
        return instance;
    }
```

```
    private NetworkParam networkParam = NetworkParam.getInstance();

    private NodeManager nodeManager = NodeManager.getInstance();

    public BroadcastResult broadcastToAllNode(BaseMessage msg, Node excludeNode, boolean
asyn, int percent) {
        if (nodeManager.getAvailableNodes().isEmpty()) {
            return new BroadcastResult(false,
NetworkErrorCode.NET_BROADCAST_NODE_EMPTY);
        }
        return broadcastToList(nodeManager.getAvailableNodes(), msg, excludeNode, asyn,
percent);
    }

    public BroadcastResult broadcastToHalfNode(BaseMessage msg, Node excludeNode, boolean
asyn) {

        if (nodeManager.getAvailableNodes().isEmpty()) {
            return new BroadcastResult(false,
NetworkErrorCode.NET_BROADCAST_NODE_EMPTY);
        }
        List<Node> nodeList = new ArrayList<>();
        int i = 0;
        for (Node node : nodeManager.getAvailableNodes()) {
            i++;
            if (i % 2 == 1) {
                nodeList.add(node);
            }
        }

        return broadcastToList(nodeList, msg, excludeNode, asyn, 50);
    }

    public BroadcastResult broadcastToNode(BaseMessage msg, Node sendNode, boolean asyn)
{
        if (sendNode == null) {
            return new BroadcastResult(false, NetworkErrorCode.NET_NODE_NOT_FOUND);
        }
        return broadcastToANode(msg, sendNode, asyn);
    }
```

```
    public BroadcastResult broadcastToNodeGroup(BaseMessage msg, String groupName,
boolean asyn) {
        NodeGroup group = nodeManager.getNodeGroup(groupName);
        if (group == null || group.size() == 0) {
            return new BroadcastResult(false,
NetworkErrorCode.NET_BROADCAST_NODE_EMPTY);
        }
        return broadcastToList(group.getNodes().values(), msg, null, asyn, 100);
    }

    public BroadcastResult broadcastToNodeGroup(BaseMessage msg, String groupName, Node
excludeNode, boolean asyn) {
        NodeGroup group = nodeManager.getNodeGroup(groupName);
        if (group == null || group.size() == 0) {
            return new BroadcastResult(false,
NetworkErrorCode.NET_BROADCAST_NODE_EMPTY);
        }
        return broadcastToList(group.getNodes().values(), msg, excludeNode, asyn, 100);
    }

    private BroadcastResult broadcastToList(Collection<Node> nodeList, BaseMessage message,
Node excludeNode, boolean asyn, int percent) {
        BroadcastResult result = new BroadcastResult();
        try {
            int successCount = 0;
            int minCount = 5;
            //
            if (nodeList.size() > minCount && percent < 100) {
                int needCount = nodeList.size() * percent / 100;
                if (needCount < minCount) {
                    needCount = minCount;
                }
                Set<Integer> set = new HashSet<>();
                while (true) {
                    Random rand = new Random();
                    int ran = rand.nextInt(nodeList.size());
                    set.add(ran);
                    if (set.size() == needCount + 1) {
                        break;
                    }
                }
```

```
            int nodeListIndex = 0;
            Collection<Node> nodeBroadcastList = new ArrayList<>();
            for (Node node : nodeList) {
                if (set.contains(nodeListIndex)) {
                    if (excludeNode != null && node.getId().equals(excludeNode.getId())) {
                        nodeListIndex++;
                        continue;
                    }
                    nodeBroadcastList.add(node);
                    if (nodeBroadcastList.size() == needCount) {
                        break;
                    }
                }
                nodeListIndex++;
            }
            nodeList = nodeBroadcastList;
        }
        for (Node node : nodeList) {
            if (excludeNode != null && node.getId().equals(excludeNode.getId())) {
                continue;
            }
            BroadcastResult br = broadcastToNode(message, node, asyn);
            if (br.isSuccess()) {
                successCount++;
                result.getBroadcastNodes().add(node);
            } else if (br.getErrorCode().equals(NetworkErrorCode.NET_MESSAGE_ERROR)) {
                return br;
            }
        }

        if (successCount == 0) {
            return new BroadcastResult(false, NetworkErrorCode.NET_BROADCAST_FAIL);
        }
    } catch (Exception e) {
        return new BroadcastResult(false, NetworkErrorCode.NET_MESSAGE_ERROR);
    }
    result.setSuccess(true);
    result.setErrorCode(KernelErrorCode.SUCCESS);
    return result;
}

public BroadcastResult broadcastToANode(BaseMessage message, Node node, boolean asyn)
```

```
{
        if (!node.isAlive()) {
            return new BroadcastResult(false, NetworkErrorCode.NET_NODE_DEAD);
        }
        if (node.getChannel() == null || !node.getChannel().isActive()) {
            return new BroadcastResult(false, NetworkErrorCode.NET_NODE_MISS_CHANNEL);
        }
        try {
            MessageHeader header = message.getHeader();
            byte[] serialize = header.serialize();
            header.setMagicNumber(networkParam.getPacketMagic());

            BaseNulsData body = message.getMsgBody();
            header.setLength(body.size());

            ChannelFuture future =
node.getChannel().writeAndFlush(Unpooled.wrappedBuffer(message.serialize()));
            if (!asyn) {
                future.await();
                boolean success = future.isSuccess();
                if (!success) {
                    return new BroadcastResult(false, NetworkErrorCode.NET_BROADCAST_FAIL);
                }
            }
        } catch (Exception e) {
            Log.error(e);
            return new BroadcastResult(false, NetworkErrorCode.NET_MESSAGE_ERROR);
        }
        return new BroadcastResult(true, KernelErrorCode.SUCCESS);
    }
}

33:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\manager\ConnectionManager.java
 */

package io.nuls.network.manager;

import io.netty.buffer.ByteBuf;
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.thread.manager.TaskManager;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.connection.netty.NettyServer;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.message.filter.MessageFilterChain;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.VersionMessage;
import io.nuls.network.util.HeartBeatThread;
import io.nuls.network.util.NetworkThreadPool;
import io.nuls.protocol.message.base.BaseMessage;
import io.nuls.protocol.message.base.MessageHeader;

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

public class ConnectionManager {

    private static ConnectionManager instance = new ConnectionManager();

    private ConnectionManager() {
    }

    public static ConnectionManager getInstance() {
        return instance;
    }

    private NetworkParam networkParam = NetworkParam.getInstance();

    private NettyServer nettyServer;

    private NodeManager nodeManager;

    private BroadcastHandler broadcastHandler;

    private NetworkMessageHandlerFactory messageHandlerFactory =
NetworkMessageHandlerFactory.getInstance();
```

```
    private MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);

    private HeartBeatThread heartBeatThread;

    public void init() {
        nodeManager = NodeManager.getInstance();
        broadcastHandler = BroadcastHandler.getInstance();
        nettyServer = new NettyServer(networkParam.getPort());
        nettyServer.init();
//        eventBusService = NulsContext.getServiceBean(EventBusService.class);
//        messageHandlerFactory = network.getMessageHandlerFactory();
    }

    public void start() {
        TaskManager.createAndRunThread(NetworkConstant.NETWORK_MODULE_ID, "node
server start", new Runnable() {
            @Override
            public void run() {
                try {
                    nettyServer.start();
                } catch (InterruptedException e) {
                    Log.error(e);
                }
            }
        }, false);
        ;
        heartBeatThread = new
HeartBeatThread(messageHandlerFactory.getHandler(VersionMessage.class.getName()));
        TaskManager.createAndRunThread(NetworkConstant.NETWORK_MODULE_ID, "heart-
beat", this.heartBeatThread);
    }

    public void connectionNode(Node node) {
        node.setStatus(Node.CONNECT);
        NetworkThreadPool.doConnect(node);
//        TaskManager.asynExecuteRunnable(new Runnable() {
//            @Override
//            public void run() {
//                try {
//                    NettyClient client = new NettyClient(node);
```

```
//                System.out.println("----------------------run----" + node.getId());
//                client.start();
//            } catch (Exception e) {
//                e.printStackTrace();
//                Log.error(e);
//            }
//        }
//    });
//    TaskManager.createAndRunThread(NetworkConstant.NETWORK_MODULE_ID, "node
connection", new Runnable() {
//        @Override
//        public void run() {
//            NettyClient client = new NettyClient(node);
//            client.start();
//        }
//    }, true);
  }

  public void receiveMessage(ByteBuf buffer, Node node) throws NulsException {
    List<BaseMessage> list;
    try {
      list = new ArrayList<>();
      byte[] bytes = new byte[buffer.readableBytes()];
      buffer.readBytes(bytes);
      NulsByteBuffer byteBuffer = new NulsByteBuffer(bytes);
      while (!byteBuffer.isFinished()) {
        MessageHeader header = byteBuffer.readNulsData(new MessageHeader());
        byteBuffer.setCursor(byteBuffer.getCursor() - header.size());
        BaseMessage message =
getMessageBusService().getMessageInstance(header.getModuleId(),
header.getMsgType()).getData();
        message = byteBuffer.readNulsData(message);
        list.add(message);
      }
      for (BaseMessage message : list) {
        if (MessageFilterChain.getInstance().doFilter(message)) {
          MessageHeader header = message.getHeader();

          if (node.getMagicNumber() == 0L) {
            node.setMagicNumber(header.getMagicNumber());
          }
```

```
                processMessage(message, node);
            } else {
                node.setStatus(Node.BAD);
                Log.debug("-------------------- receive message filter remove node --------------------------"
+ node.getId());
                nodeManager.removeNode(node.getId());
            }
        }
    } catch (Exception e) {
        throw new NulsException(KernelErrorCode.DATA_ERROR, e);
    } finally {
        buffer.clear();
    }
}


private void processMessage(BaseMessage message, Node node) {
    if (message == null) {
        return;
    }
    if (isNetworkMessage(message)) {
        if (node.getStatus() != Node.HANDSHAKE && !isHandShakeMessage(message)) {
            return;
        }
        asynExecute(message, node);
    } else {
        if (!node.isHandShake()) {
            return;
        }
        messageBusService.receiveMessage(message, node);
    }
}

private void asynExecute(BaseMessage message, Node node) {
    NetworkThreadPool.asynNetworkMessage(message, node, heartBeatThread,
messageHandlerFactory, this);
//      if (message.getHeader().getMsgType() == NetworkConstant.NETWORK_VERSION) {
//          heartBeatThread.offerMessage(message, node);
//          return;
//      }
//      BaseNetworkMeesageHandler handler = messageHandlerFactory.getHandler(message);
//
```

```
//        TaskManager.asynExecuteRunnable(new Runnable() {
//            @Override
//            public void run() {
//                try {
//                    NetworkEventResult messageResult = handler.process(message, node);
//                    processMessageResult(messageResult, node);
//                } catch (Exception e) {
//                    e.printStackTrace();
//                    Log.error(e);
//                }
//            }
//
//            @Override
//            public String toString() {
//                StringBuilder log = new StringBuilder();
//                log.append("event: " + message.toString())
//                        .append(", hash: " + message.getHash())
//                        .append(", Node: " + node.toString());
//                return log.toString();
//            }
//        });
    }

    public void processMessageResult(NetworkEventResult messageResult, Node node) {
        if (!node.isAlive()) {
            return;
        }
        if (messageResult == null || !messageResult.isSuccess()) {
            return;
        }
        if (messageResult.getReplyMessage() != null) {
            broadcastHandler.broadcastToNode((BaseMessage) messageResult.getReplyMessage(),
node, true);
        }
    }

    private boolean isNetworkMessage(BaseMessage message) {
        return message.getHeader().getModuleId() == NetworkConstant.NETWORK_MODULE_ID;
    }

    private boolean isHandShakeMessage(BaseMessage message) {
        if (message.getHeader().getMsgType() == NetworkConstant.NETWORK_HANDSHAKE ||
```

```
                message.getHeader().getMsgType() == NetworkConstant.NETWORK_P2P_NODE) {
            return true;
        }
        return false;
    }

    public MessageBusService getMessageBusService() {
        if (messageBusService == null) {
            messageBusService = NulsContext.getServiceBean(MessageBusService.class);
        }
        return messageBusService;
    }

    public void shutdown() {
        nettyServer.shutdown();
        nodeManager.shutdown();
    }
}

34:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\manager\NetworkMessageHandlerFactory.java
 */

package io.nuls.network.manager;

import io.nuls.network.message.impl.*;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.*;
import io.nuls.protocol.message.base.BaseMessage;

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

public class NetworkMessageHandlerFactory {

    private Map<String, BaseNetworkMeesageHandler> handlerMap = new HashMap<>();

    private static NetworkMessageHandlerFactory INSTANCE = new
NetworkMessageHandlerFactory();

    public static NetworkMessageHandlerFactory getInstance() {
        return INSTANCE;
```

```
    }

    private NetworkMessageHandlerFactory() {
        handlerMap.put(HandshakeMessage.class.getName(),
HandshakeMessageHandler.getInstance());
        handlerMap.put(GetVersionMessage.class.getName(),
GetVersionMessageHandler.getInstance());
        handlerMap.put(VersionMessage.class.getName(), VersionMessageHandler.getInstance());
        handlerMap.put(GetNodesMessage.class.getName(),
GetNodesMessageHandler.getInstance());
        handlerMap.put(NodesMessage.class.getName(), NodesMessageHandler.getInstance());
        handlerMap.put(GetNodesIpMessage.class.getName(),
GetNodesIpMessageHandler.getInstance());
        handlerMap.put(NodesIpMessage.class.getName(), NodesIpMessageHandler.getInstance());
        handlerMap.put(P2PNodeMessage.class.getName(),
P2pNodeMessageHandler.getInstance());
    }

    public BaseNetworkMeesageHandler getHandler(BaseMessage message) {
        return handlerMap.get(message.getClass().getName());
    }

    public BaseNetworkMeesageHandler getHandler(String handlerKey) {
        return handlerMap.get(handlerKey);
    }
}

35:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\manager\NodeDiscoverHandler.java
package io.nuls.network.manager;

import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.message.GetVersionMessage;
import io.nuls.network.protocol.message.NetworkMessageBody;

import java.util.Collection;

/**
```

```
 * @author vivi
 */
public class NodeDiscoverHandler implements Runnable {

    private static NodeDiscoverHandler instance = new NodeDiscoverHandler();

    private NodeDiscoverHandler() {
    }

    public static NodeDiscoverHandler getInstance() {
        return instance;
    }

    private NetworkParam networkParam = NetworkParam.getInstance();

    private NodeManager nodesManager = NodeManager.getInstance();

    private BroadcastHandler broadcastHandler = BroadcastHandler.getInstance();

    private boolean running = false;

    public void start() {
        running = true;
        TaskManager.createAndRunThread(NetworkConstant.NETWORK_MODULE_ID,
"NetworkNodeDiscover", this);
    }

    /**
     * Inquire more of the other nodes to the connected nodes
     *
     * @param size
     */
    public void findOtherNode(int size) {
//        NodeMessageBody messageBody = new NodeMessageBody();
//        messageBody.setLength(size);
//        //ipip
//        List<String> ipList = new ArrayList<>();
//        for (Node node : nodesManager.getAvailableNodes()) {
//            ipList.add(node.getIp());
//        }
//        messageBody.setIpList(ipList);
//        GetNodesMessage message = new GetNodesMessage(messageBody);
```

```
//        List<Node> nodeList = new ArrayList<>(nodesManager.getAvailableNodes());
//        Collections.shuffle(nodeList);
//        for (int i = 0; i < nodeList.size(); i++) {
//            if (i == 2) {
//                break;
//            }
//            Node node = nodeList.get(i);
//            broadcastHandler.broadcastToNode(message, node, true);
//        }
    }

    /**
     * 10
     */

    @Override
    public void run() {
        Thread.currentThread().setPriority(Thread.MIN_PRIORITY);

        while (running) {
            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            Collection<Node> nodeList = nodesManager.getAvailableNodes();
            NetworkMessageBody body = new
NetworkMessageBody(NetworkConstant.HANDSHAKE_CLIENT_TYPE, networkParam.getPort(),
                    NulsContext.getInstance().getBestHeight(),
NulsContext.getInstance().getBestBlock().getHeader().getHash());
            GetVersionMessage getVersionMessage = new GetVersionMessage(body);

            for (Node node : nodeList) {
                if (node.getType() == Node.OUT) {
                    broadcastHandler.broadcastToNode(getVersionMessage, node, true);
                }
            }
        }
    }
}
```

```
36:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\manager\NodeManager.java
 */

package io.nuls.network.manager;

import io.netty.channel.Channel;
import io.netty.channel.socket.SocketChannel;
import io.nuls.core.tools.date.DateUtil;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.network.IpUtil;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.model.Node;
import io.nuls.network.model.NodeGroup;
import io.nuls.network.protocol.message.NetworkMessageBody;
import io.nuls.network.protocol.message.P2PNodeBody;
import io.nuls.network.protocol.message.P2PNodeMessage;
import io.nuls.network.storage.service.NetworkStorageService;

import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.locks.ReentrantLock;

public class NodeManager implements Runnable {

    private static NodeManager instance = new NodeManager();

    private NodeManager() {

    }

    public static NodeManager getInstance() {
        return instance;
    }

    private NetworkParam networkParam = NetworkParam.getInstance();
```

```
    private Map<String, NodeGroup> nodeGroups = new ConcurrentHashMap<>();

    private ReentrantLock lock = new ReentrantLock();

    private ConnectionManager connectionManager;

    private NodeDiscoverHandler nodeDiscoverHandler;

    private NetworkStorageService networkStorageService;

    private BroadcastHandler broadcastHandler;

    boolean running;

    //
    volatile boolean connectedMySelf;
    //
    volatile boolean isSeed;

    private List<Node> seedNodes;
    //
    private Map<String, Node> disConnectNodes = new ConcurrentHashMap<>();
    //
    private Map<String, Node> connectedNodes = new ConcurrentHashMap<>();
    //
    private Map<String, Node> handShakeNodes = new ConcurrentHashMap<>();

    /**
     * (outGroup)(inGroup)
     */
    public void init() {
        connectionManager = ConnectionManager.getInstance();
        nodeDiscoverHandler = NodeDiscoverHandler.getInstance();
        broadcastHandler = BroadcastHandler.getInstance();
        // init default NodeGroup
        NodeGroup inNodes = new NodeGroup(NetworkConstant.NETWORK_NODE_IN_GROUP);
        NodeGroup outNodes = new
NodeGroup(NetworkConstant.NETWORK_NODE_OUT_GROUP);
        nodeGroups.put(inNodes.getName(), inNodes);
        nodeGroups.put(outNodes.getName(), outNodes);
        //
        for (String ip : IpUtil.getIps()) {
```

```
            if (isSeedNode(ip)) {
//                networkParam.setMaxInCount(networkParam.getMaxInCount() * 2);
                isSeed = true;
            }
        }
    }

    /**
     *
     *
     */
    public void start() {
        //ip
        String externalIp = getNetworkStorageService().getExternalIp();
        if (externalIp != null) {
            networkParam.getLocalIps().add(externalIp);
        }

        List<Node> nodeList = getNetworkStorageService().getLocalNodeList();
        nodeList.addAll(getSeedNodes());
        for (Node node : nodeList) {
            addNode(node);
        }
        running = true;
        TaskManager.createAndRunThread(NetworkConstant.NETWORK_MODULE_ID,
"NetworkNodeManager", this);
        nodeDiscoverHandler.start();
    }

    /**
     *
     */
    public void reset() {
        Log.debug("---!!!!!!!----------network nodeManager reset----------!!!!!!-----");
//        for (Node node : disConnectNodes.values()) {
//            node.setFailCount(NetworkConstant.CONEECT_FAIL_MAX_COUNT);
//        }
        for (Node node : handShakeNodes.values()) {
            removeNode(node);
        }
    }
```

```
public Node getNode(String nodeId) {
    Node node = disConnectNodes.get(nodeId);
    if (node == null) {
        node = connectedNodes.get(nodeId);
    }
    if (node == null) {
        node = handShakeNodes.get(nodeId);
    }
    return node;
}

public Node getHandshakeNode(String nodeId) {
    Node node = handShakeNodes.get(nodeId);
    return node;
}

/**
 *
 */
public Map<String, Node> getNodes() {
    Map<String, Node> nodeMap = new HashMap<>();
    nodeMap.putAll(disConnectNodes);
    nodeMap.putAll(connectedNodes);
    nodeMap.putAll(handShakeNodes);
    return nodeMap;
}

public List<Node> getCanConnectNodes() {
    List nodeList = new ArrayList();
    for (Node node : disConnectNodes.values()) {
        if (node.getType() == Node.OUT && node.isCanConnect()) {
            nodeList.add(node);
        }
    }
    for (Node node : connectedNodes.values()) {
        if (node.getType() == Node.OUT) {
            nodeList.add(node);
        }
    }
    for (Node node : handShakeNodes.values()) {
        if (node.getType() == Node.OUT) {
            nodeList.add(node);
```

```
        }
    }
    return nodeList;
}

public Map<String, Node> getConnectedNodes() {
    Map<String, Node> nodeMap = new HashMap<>();
    nodeMap.putAll(connectedNodes);
    nodeMap.putAll(handShakeNodes);
    return nodeMap;
}

public Collection<Node> getAvailableNodes() {
    return handShakeNodes.values();
}

public NodeGroup getNodeGroup(String groupName) {
    return nodeGroups.get(groupName);
}

/**
 *
 */
public boolean addNode(Node node) {
    //
    if (networkParam.getLocalIps().contains(node.getIp())) {
        return false;
    }
    if (node.getStatus() != Node.WAIT) {
        return false;
    }
    lock.lock();
    try {
        //ip
        Collection<Node> nodeMap = getNodes().values();
        int count = 0;
        for (Node n : nodeMap) {
            if (node.getIp().equals(n.getIp()) && n.getType() == Node.OUT) {
                return false;
            }
            if (node.isCanConnect()) {
                count++;
```

```
            }
        }
        if (count >= 50) {
            return false;
        }

        node.setType(Node.OUT);
        node.setTestConnect(false);
        disConnectNodes.put(node.getId(), node);
        connectionManager.connectionNode(node);
        return true;
    } finally {
        lock.unlock();
    }
}

/**
 *
 */
public boolean processConnectedNode(Node node, Channel channel) {
    lock.lock();
    try {
        if (connectedNodes.containsKey(node.getId()) ||
handShakeNodes.containsKey(node.getId())) {
            return false;
        }
        //ip
        if (node.getType() == Node.OUT) {
            for (Node n : getConnectedNodes().values()) {
                if (n.getIp().equals(node.getIp())) {
                    return false;
                }
            }
        }
        node.setChannel(channel);
        disConnectNodes.remove(node.getId());
        connectedNodes.put(node.getId(), node);
        return true;
    } finally {
        lock.unlock();
    }
}
```

```
    public void removeNode(String nodeId) {
        Node node = getNode(nodeId);
        if (node != null) {
            removeNode(node);
        } else {
            Log.info("------------remove node is null-----------" + nodeId);
            getNetworkStorageService().deleteNode(nodeId);
        }
    }

    /**
     *
     *
     */
    public void removeNode(Node node) {
        lock.lock();
        try {
            if (node.getChannel() != null && node.getChannel().isActive()) {
                node.getChannel().close();
                return;
            }
            node.destroy();
            removeNodeFromGroup(node);
            removeNodeHandler(node);
        } finally {
            lock.unlock();
        }
    }

    public void removeHandshakeNode(String nodeId) {
        Node node = getHandshakeNode(nodeId);
        if (node != null) {
            removeNode(node);
        } else {
//            Log.info("------------removeHandshakeNode node is null-----------" + nodeId);
            getNetworkStorageService().deleteNode(nodeId);
        }
    }

    private void removeNodeHandler(Node node) {
        //
```

```
        if (node.getType() == Node.BAD || node.getType() == Node.IN) {
            disConnectNodes.remove(node.getId());
            connectedNodes.remove(node.getId());
            handShakeNodes.remove(node.getId());
            if (node.getStatus() == Node.BAD) {
                getNetworkStorageService().deleteNode(node.getId());
            }
            return;
        }
        if (isSeedNode(node.getIp())) {
            disConnectNodes.remove(node.getId());
            connectedNodes.remove(node.getId());
            handShakeNodes.remove(node.getId());
            return;
        }
        //ip
        if (isSeedNode(node.getIp()) || networkParam.getLocalIps().contains(node.getIp())) {
            disConnectNodes.remove(node.getId());
            return;
        }

        if (connectedNodes.containsKey(node.getId())) {
            connectedNodes.remove(node.getId());
        }
        if (handShakeNodes.containsKey(node.getId())) {
            handShakeNodes.remove(node.getId());
        }

        if (node.getFailCount() <= NetworkConstant.CONNECT_FAIL_MAX_COUNT) {
            node.setLastFailTime(TimeService.currentTimeMillis());
            if (!disConnectNodes.containsKey(node.getId())) {
                disConnectNodes.put(node.getId(), node);
            }
        } else {
            disConnectNodes.remove(node.getId());
            getNetworkStorageService().deleteNode(node.getId());
        }
    }

    public boolean handshakeNode(String groupName, Node node, NetworkMessageBody
versionMessage) {
        lock.lock();
```

```
    try {
        if (!checkFullHandShake(node)) {
            return false;
        }
        if (!connectedNodes.containsKey(node.getId())) {
            return false;
        }
        node.setStatus(Node.HANDSHAKE);
        node.setBestBlockHash(versionMessage.getBestBlockHash());
        node.setBestBlockHeight(versionMessage.getBestBlockHeight());

        connectedNodes.remove(node.getId());
        handShakeNodes.put(node.getId(), node);
        return addNodeToGroup(groupName, node);
    } finally {
        lock.unlock();
    }
}

public void saveNode(Node node) {
    if (!isSeedNode(node.getIp())) {
        getNetworkStorageService().saveNode(node);
    }

}

public void saveExternalIp(String ip) {
    NetworkParam.getInstance().getLocalIps().add(ip);
    getNetworkStorageService().saveExternalIp(ip);
}

private NetworkStorageService getNetworkStorageService() {
    if (null == this.networkStorageService) {
        this.networkStorageService = NulsContext.getServiceBean(NetworkStorageService.class);
    }
    return this.networkStorageService;
}

public void tryToConnectMySelf() {
    String externalIp = getNetworkStorageService().getExternalIp();
    if (StringUtils.isBlank(externalIp)) {
        return;
```

```
        }
        //IPip
        if (!connectedMySelf) {
            connectedMySelf = true;
            Node node = new Node();
            node.setIp(externalIp);
            node.setPort(networkParam.getPort());
            node.setSeverPort(networkParam.getPort());
            node.setType(Node.OUT);
            connectionManager.connectionNode(node);
        }
    }

    /**
     *
     */
    public void broadNodeSever() {
        String externalIp = getNetworkStorageService().getExternalIp();
        if (!StringUtils.isBlank(externalIp)) {
            P2PNodeBody p2PNodeBody = new P2PNodeBody(externalIp, networkParam.getPort());
            P2PNodeMessage message = new P2PNodeMessage(p2PNodeBody);
            broadcastHandler.broadcastToAllNode(message, null, true, 100);
        }
    }

    private boolean checkFullHandShake(Node node) {
        if (node.getType() == Node.IN) {
            NodeGroup inGroup = getNodeGroup(NetworkConstant.NETWORK_NODE_IN_GROUP);
            return inGroup.size() < networkParam.getMaxInCount();
        } else {
            NodeGroup outGroup =
getNodeGroup(NetworkConstant.NETWORK_NODE_OUT_GROUP);
            return outGroup.size() < networkParam.getMaxOutCount();
        }
    }

    /**
     *
     */
    public boolean addNodeToGroup(String groupName, Node node) {
        NodeGroup nodeGroup = nodeGroups.get(groupName);
        if (nodeGroup == null) {
```

```
            //todo throw new NulsExcetpion
            //throw new RuntimeException("group not found");
            return false;
        }
        if (groupName.equals(NetworkConstant.NETWORK_NODE_IN_GROUP) &&
nodeGroup.size() >= networkParam.getMaxInCount()) {
            return false;
        }
        if (groupName.equals(NetworkConstant.NETWORK_NODE_OUT_GROUP) &&
nodeGroup.size() >= networkParam.getMaxOutCount()) {
            return false;
        }
        node.addGroup(groupName);
        nodeGroup.addNode(node);
        return true;
    }

    private void removeNodeFromGroup(Node node) {
        for (String groupName : node.getGroupSet()) {
            NodeGroup group = nodeGroups.get(groupName);
            if (group != null) {
                group.removeNode(node.getId());
            }
        }
        node.getGroupSet().clear();
    }

    /**
     *
     */
    public List<Node> getSeedNodes() {
        if (seedNodes != null) {
            return seedNodes;
        }
        seedNodes = new ArrayList<>();
        for (String seedIp : networkParam.getSeedIpList()) {
            String[] ipPort = seedIp.split(":");
            seedNodes.add(new Node(ipPort[0], Integer.parseInt(ipPort[1]), Integer.parseInt(ipPort[1]),
Node.OUT));
        }
        return seedNodes;
    }
```

```
/**
 *
 */
public boolean isSeedNode(String ip) {
    for (Node node : getSeedNodes()) {
        if (node.getIp().equals(ip)) {
            return true;
        }
    }
    return false;
}

/**
 * 2
 */
private void removeSeedNode() {
    Collection<Node> nodes = handShakeNodes.values();
    int count = 0;
    List<String> seedIpList = networkParam.getSeedIpList();
    Collections.shuffle(seedIpList);

    for (Node n : nodes) {
        if (seedIpList.contains(n.getIp())) {
            count++;
            if (count > 1) {
                removeNode(n);
            }
        }
    }
}

private boolean checkIpExist(String ip) {
    Collection<Node> nodeMap = getNodes().values();
    for (Node node : nodeMap) {
        if (node.getIp().equals(ip)) {
            return true;
        }
    }
    return false;
}
```

```
    @Override
    public void run() {
        Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
        while (running) {
            try {
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            /*System.out.println("--------disConnectNodes:" + disConnectNodes.size());
            for (Node node : disConnectNodes.values()) {
                System.out.println(node.toString());
            }

            System.out.println("--------connectedNodes:" + connectedNodes.size());
            for (Node node : connectedNodes.values()) {
                System.out.println(node.toString());
            }

            System.out.println("--------handShakeNodes:" + handShakeNodes.size());
            for (Node node : handShakeNodes.values()) {
                System.out.println(node.toString());
            }*/

            if (handShakeNodes.size() > 9) {
                removeSeedNode();
            } else if (handShakeNodes.size() <= 2) {
                //
                for (Node node : getSeedNodes()) {
                    addNode(node);
                }
            } else if (handShakeNodes.size() < networkParam.getMaxOutCount() &&
connectedNodes.size() == 0) {
                for (Node node : disConnectNodes.values()) {
                    if (node.isCanConnect() && node.getStatus() == Node.WAIT) {
                        connectionManager.connectionNode(node);
                    }
                }
            }

            //
```

```
            long now = TimeService.currentTimeMillis();
            for (Node node : disConnectNodes.values()) {
                if (node.getStatus() == Node.WAIT) {
                    if (node.isCanConnect() && now > node.getLastFailTime() + 5 *
DateUtil.MINUTE_TIME) {
                        connectionManager.connectionNode(node);
                    } else if (now > node.getLastFailTime() + node.getFailCount() *
DateUtil.MINUTE_TIME) {
                        connectionManager.connectionNode(node);
                    }
                }
            }
        }
    }

    public void shutdown() {
        running = false;
        for (Node node : handShakeNodes.values()) {
            if(node.getType() == Node.OUT) {
                removeNode(node);
            }
        }
    }
}

37:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\filter\impl\MagicNumberFilter.java
 */

package io.nuls.network.message.filter.impl;

import io.nuls.network.message.filter.NulsMessageFilter;
import io.nuls.protocol.message.base.BaseMessage;
import io.nuls.protocol.message.base.MessageHeader;

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

public class MagicNumberFilter implements NulsMessageFilter{

    private Set<Long> magicSet = new HashSet<>();
```

```
    private static MagicNumberFilter instance = new MagicNumberFilter();

    private MagicNumberFilter() {

    }

    public static MagicNumberFilter getInstance() {
        return instance;
    }

    @Override
    public boolean filter(BaseMessage message) {
        MessageHeader header = message.getHeader();
        return magicSet.contains(header.getMagicNumber());
    }

    public void addMagicNum(Long magicNum) {
        magicSet.add(magicNum);
    }

    public void removeMagicNum(Long magicNum) {
        magicSet.remove(magicNum);
    }
}
```

38:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\message\filter\MessageFilterChain.java

```
package io.nuls.network.message.filter;


import io.nuls.protocol.message.base.BaseMessage;

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

/**
 * @author vivi
 */
public class MessageFilterChain {

    private static MessageFilterChain messageFilterChain = new MessageFilterChain();
```

```
    protected List<NulsMessageFilter> filters;

    private MessageFilterChain() {
        filters = new ArrayList<>();
    }

    public static MessageFilterChain getInstance() {
        return messageFilterChain;
    }


    public void addFilter(NulsMessageFilter filter) {
        filters.add(filter);
    }

    public void deleteFilter(NulsMessageFilter filter) {
        if (filters.contains(filter)) {
            filters.remove(filter);
        }
    }

    public boolean doFilter(BaseMessage message) {
        for (int i = 0; i < filters.size(); i++) {
            if (!filters.get(i).filter(message)) {
                return false;
            }
        }
        return true;
    }
}
```

39:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\message\filter\NulsMessageFilter.java

```
package io.nuls.network.message.filter;

import io.nuls.protocol.message.base.BaseMessage;

/**
 * @author vivi
 */
public interface NulsMessageFilter {
```

```
    boolean filter(BaseMessage message);
}

40:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\GetNodesIpMessageHandler.java
 */

package io.nuls.network.message.impl;

import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.NodeMessageBody;
import io.nuls.network.protocol.message.NodesIpMessage;
import io.nuls.protocol.message.base.BaseMessage;

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

public class GetNodesIpMessageHandler implements BaseNetworkMeesageHandler {

    private static GetNodesIpMessageHandler instance = new GetNodesIpMessageHandler();

    private GetNodesIpMessageHandler() {

    }

    public static GetNodesIpMessageHandler getInstance() {
        return instance;
    }

    private NodeManager nodeManager = NodeManager.getInstance();

    @Override
    public NetworkEventResult process(BaseMessage message, Node node) {
        Collection<Node> availableNodes = nodeManager.getNodes().values();
        List<String> ipList = new ArrayList<>();
        for (Node n : availableNodes) {
            ipList.add(n.getIp());
        }
```

```
        NodeMessageBody messageBody = new NodeMessageBody();
//        messageBody.setIpList(ipList);
        NodesIpMessage nodesIpMessage = new NodesIpMessage(messageBody);

        return new NetworkEventResult(true, nodesIpMessage);
    }
}

41:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\GetNodesMessageHandler.java
 */

package io.nuls.network.message.impl;

import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.GetNodesMessage;
import io.nuls.network.protocol.message.NodeMessageBody;
import io.nuls.network.protocol.message.NodesMessage;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.*;

public class GetNodesMessageHandler implements BaseNetworkMeesageHandler {

    private static GetNodesMessageHandler instance = new GetNodesMessageHandler();

    private GetNodesMessageHandler() {

    }

    public static GetNodesMessageHandler getInstance() {
        return instance;
    }


    private NodeManager nodeManager = NodeManager.getInstance();

    @Override
```

```
    public NetworkEventResult process(BaseMessage message, Node node) {
        GetNodesMessage getNodesMessage = (GetNodesMessage) message;

        NodeMessageBody body = getNodesMessage.getMsgBody();
//        body.getIpList().add(node.getIp());
//        List<Node> nodeList = getAvailableNodes(body.getLength(), body.getIpList());
        body = new NodeMessageBody();
//        body.setNodeList(nodeList);
        NodesMessage nodesMessage = new NodesMessage(body);
        return new NetworkEventResult(true, nodesMessage);
    }

    private List<Node> getAvailableNodes(int length, List<String> ipList) {
        List<Node> nodeList = new ArrayList<>();
        List<Node> availableNodes = new ArrayList<>(nodeManager.getAvailableNodes());
        Collections.shuffle(availableNodes);
        Set<String> ipSet = new HashSet<>();
        ipSet.addAll(ipList);
        for (Node node : availableNodes) {
            if (ipSet.contains(node.getIp())) {
                continue;
            }
            if (node.getSeverPort() == null || node.getSeverPort() == 0) {
                continue;
            }
            Node newNode = new Node();
            newNode.setIp(node.getIp());
            newNode.setPort(node.getSeverPort());
            newNode.setSeverPort(node.getSeverPort());
            ipSet.add(node.getIp());
            nodeList.add(newNode);
            if (nodeList.size() == length) {
                break;
            }
        }
        return nodeList;
    }
}

42:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\GetVersionMessageHandler.java
 */
```

```
package io.nuls.network.message.impl;

import io.nuls.kernel.context.NulsContext;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.GetVersionMessage;
import io.nuls.network.protocol.message.NetworkMessageBody;
import io.nuls.network.protocol.message.VersionMessage;
import io.nuls.protocol.message.base.BaseMessage;

public class GetVersionMessageHandler implements BaseNetworkMeesageHandler {

    private static GetVersionMessageHandler instance = new GetVersionMessageHandler();

    private GetVersionMessageHandler() {

    }

    public static GetVersionMessageHandler getInstance() {
        return instance;
    }

    private NodeManager nodeManager = NodeManager.getInstance();

    private NetworkParam networkParam = NetworkParam.getInstance();

    @Override
    public NetworkEventResult process(BaseMessage message, Node node) {
        GetVersionMessage getVersionMessage = (GetVersionMessage) message;
        NetworkMessageBody body = getVersionMessage.getMsgBody();

        if (body.getBestBlockHeight() < 0) {
            node.setStatus(Node.BAD);
            nodeManager.removeNode(node.getId());
            return null;
        }
        node.setBestBlockHeight(body.getBestBlockHeight());
```

```
        node.setBestBlockHash(body.getBestBlockHash());

        NetworkMessageBody myVersionBody = new
NetworkMessageBody(NetworkConstant.HANDSHAKE_CLIENT_TYPE, networkParam.getPort(),
                NulsContext.getInstance().getBestHeight(),
NulsContext.getInstance().getBestBlock().getHeader().getHash());
        return new NetworkEventResult(true, new VersionMessage(myVersionBody));
    }

}

43:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\HandshakeMessageHandler.java
 */

package io.nuls.network.message.impl;

import io.netty.channel.socket.SocketChannel;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.HandshakeMessage;
import io.nuls.network.protocol.message.NetworkMessageBody;
import io.nuls.network.protocol.message.NodeMessageBody;
import io.nuls.network.protocol.message.NodesMessage;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.List;

public class HandshakeMessageHandler implements BaseNetworkMeesageHandler {

    private NodeManager nodeManager = NodeManager.getInstance();

    private NetworkParam networkParam = NetworkParam.getInstance();

    private static HandshakeMessageHandler instance = new HandshakeMessageHandler();
```

```
    private HandshakeMessageHandler() {

    }

    public static HandshakeMessageHandler getInstance() {
        return instance;
    }

    @Override
    public NetworkEventResult process(BaseMessage message, Node node) {
        HandshakeMessage handshakeMessage = (HandshakeMessage) message;
        SocketChannel socketChannel = (SocketChannel) node.getChannel();
        NetworkMessageBody body = handshakeMessage.getMsgBody();


        boolean isServer = false;     //
        boolean isSuccess = false;

        if (body.getHandshakeType() == NetworkConstant.HANDSHAKE_SEVER_TYPE) {
            isSuccess =
nodeManager.handshakeNode(NetworkConstant.NETWORK_NODE_OUT_GROUP, node, body);
        } else {
            isServer = true;
            isSuccess =
nodeManager.handshakeNode(NetworkConstant.NETWORK_NODE_IN_GROUP, node, body);
        }

        if (!isSuccess) {
            if (socketChannel != null) {
                Log.debug("localInfo: " + socketChannel.localAddress().getHostString() + ":" +
socketChannel.localAddress().getPort());
                Log.debug("handshake failed, close the connetion.");
                socketChannel.close();
                return null;
            }
        }
        //ip
        node.setFailCount(0);
        node.setSeverPort(body.getSeverPort());
        node.setBestBlockHash(body.getBestBlockHash());
        node.setBestBlockHeight(body.getBestBlockHeight());
        if (node.getType() == Node.OUT) {
```

```
            nodeManager.saveNode(node);
        }
        if (nodeManager.isSeedNode(node.getIp())) {
            nodeManager.saveExternalIp(body.getNodeIp());
        }

        if (!isServer) {
            //
            nodeManager.tryToConnectMySelf();

            body = new NetworkMessageBody(NetworkConstant.HANDSHAKE_CLIENT_TYPE,
networkParam.getPort(),
                    NulsContext.getInstance().getBestHeight(),
NulsContext.getInstance().getBestBlock().getHeader().getHash(),
                    socketChannel.remoteAddress().getHostString());
            return new NetworkEventResult(true, new HandshakeMessage(body));
        } else {
            //
            List<Node> nodeList = nodeManager.getCanConnectNodes();
            for (int i = nodeList.size() - 1; i >= 0; i--) {
                Node n = nodeList.get(i);
                if (nodeManager.isSeedNode(n.getIp())) {
                    nodeList.remove(i);
                } else if (n.getId().equals(node.getId())) {
                    nodeList.remove(i);
                }
            }
            NodeMessageBody messageBody = new NodeMessageBody(nodeList);
            NodesMessage nodesMessage = new NodesMessage(messageBody);
            return new NetworkEventResult(true, nodesMessage);
        }
    }
}

44:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\NodesIpMessageHandler.java
 */

package io.nuls.network.message.impl;

import io.nuls.kernel.func.TimeService;
import io.nuls.network.constant.NetworkParam;
```

```
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.NodeMessageBody;
import io.nuls.network.protocol.message.NodesIpMessage;
import io.nuls.protocol.message.base.BaseMessage;

public class NodesIpMessageHandler implements BaseNetworkMeesageHandler {

    private static NodesIpMessageHandler instance = new NodesIpMessageHandler();

    private NodesIpMessageHandler() {

    }

    public static NodesIpMessageHandler getInstance() {
        return instance;
    }

    private NetworkParam networkParam = NetworkParam.getInstance();

    @Override
    public NetworkEventResult process(BaseMessage message, Node node) {
        NodesIpMessage handshakeMessage = (NodesIpMessage) message;
        NodeMessageBody body = handshakeMessage.getMsgBody();

//        for(String ip : body.getIpList()) {
//            networkParam.getIpMap().put(ip, TimeService.currentTimeMillis());
//        }
        return null;
    }
}

45:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\NodesMessageHandler.java
 */

package io.nuls.network.message.impl;

import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
```

```
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.NodeMessageBody;
import io.nuls.network.protocol.message.NodesMessage;
import io.nuls.protocol.message.base.BaseMessage;

public class NodesMessageHandler implements BaseNetworkMeesageHandler {

    private static NodesMessageHandler instance = new NodesMessageHandler();

    private NodesMessageHandler() {

    }

    public static NodesMessageHandler getInstance() {
        return instance;
    }

    private NodeManager nodeManager = NodeManager.getInstance();

    @Override
    public NetworkEventResult process(BaseMessage message, Node node) {
        NodesMessage nodesMessage = (NodesMessage) message;
        NodeMessageBody body = nodesMessage.getMsgBody();
        for (Node newNode : body.getNodeList()) {
            nodeManager.addNode(newNode);
        }
        return null;
    }
}

46:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\P2pNodeMessageHandler.java
 */

package io.nuls.network.message.impl;

import io.nuls.network.cache.NodeCacheManager;
import io.nuls.network.manager.BroadcastHandler;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
```

```
import io.nuls.network.protocol.message.P2PNodeBody;
import io.nuls.network.protocol.message.P2PNodeMessage;
import io.nuls.protocol.message.base.BaseMessage;

public class P2pNodeMessageHandler implements BaseNetworkMeesageHandler {

    private NodeManager nodeManager = NodeManager.getInstance();

    private static P2pNodeMessageHandler instance = new P2pNodeMessageHandler();

    private NodeCacheManager nodeCacheManager = NodeCacheManager.getInstance();

    private BroadcastHandler broadcastHandler = BroadcastHandler.getInstance();

    private P2pNodeMessageHandler() {

    }

    public static P2pNodeMessageHandler getInstance() {
        return instance;
    }

    @Override
    public NetworkEventResult process(BaseMessage message, Node node) {
        P2PNodeMessage nodeMessage = (P2PNodeMessage) message;
        P2PNodeBody nodeBody = nodeMessage.getMsgBody();
        P2PNodeBody cacheBody = nodeCacheManager.getNode(nodeBody.getId());
        //
        if (cacheBody != null) {
//            Log.info("---------------------cacheBody is not null-----------------------------" +
cacheBody.toString());
            Node node1 = nodeManager.getNode(cacheBody.getId());
            if(node1 != null) {
                node1.setFailCount(0);
            }
            return null;
        }
        //
        Node newNode = new Node(nodeBody.getNodeIp(), nodeBody.getSeverPort(),
nodeBody.getSeverPort(), Node.OUT);
        nodeManager.addNode(newNode);
        nodeCacheManager.cacheNode(nodeBody);
```

```
        //
        broadcastHandler.broadcastToAllNode(message, node, true,100);
        return null;
    }
}

47:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\message\impl\VersionMessageHandler.java
 */

package io.nuls.network.message.impl;

import io.nuls.kernel.func.TimeService;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.network.protocol.message.NetworkMessageBody;
import io.nuls.network.protocol.message.VersionMessage;
import io.nuls.protocol.message.base.BaseMessage;

public class VersionMessageHandler implements BaseNetworkMeesageHandler {

    private static VersionMessageHandler instance = new VersionMessageHandler();

    private VersionMessageHandler() {

    }

    public static VersionMessageHandler getInstance() {
        return instance;
    }

    private NodeManager nodeManager = NodeManager.getInstance();

    @Override
    public NetworkEventResult process(BaseMessage message, Node node) {
        VersionMessage versionMessage = (VersionMessage) message;
        NetworkMessageBody body = versionMessage.getMsgBody();

        if (body.getBestBlockHeight() < 0) {
            node.setStatus(Node.BAD);
```

```
            nodeManager.removeNode(node.getId());
            return null;
        }
        node.setBestBlockHeight(body.getBestBlockHeight());
        node.setBestBlockHash(body.getBestBlockHash());
        node.setTimeOffset(TimeService.currentTimeMillis() - body.getNetworkTime());
        return null;
    }
}

48:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\module\impl\NettyNetworkModuleBootstrap.java
 */

package io.nuls.network.module.impl;

import io.nuls.core.tools.network.IpUtil;
import io.nuls.db.constant.DBConstant;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.message.bus.manager.MessageManager;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.manager.ConnectionManager;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.message.filter.MessageFilterChain;
import io.nuls.network.message.filter.impl.MagicNumberFilter;
import io.nuls.network.module.AbstractNetworkModule;
import io.nuls.network.protocol.message.*;
import io.nuls.protocol.constant.ProtocolConstant;

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

import static io.nuls.network.constant.NetworkConstant.*;

public class NettyNetworkModuleBootstrap extends AbstractNetworkModule {

    private ConnectionManager connectionManager = ConnectionManager.getInstance();

    private NodeManager nodeManager = NodeManager.getInstance();

    @Override
```

```
    public void init() {
        initNetworkParam();
        initOther();
        connectionManager.init();
        nodeManager.init();
    }

    private void initNetworkParam() {
        NetworkParam networkParam = NetworkParam.getInstance();
        networkParam.setPort(NulsConfig.MODULES_CONFIG.getCfgValue(NETWORK_SECTION,
NETWORK_SERVER_PORT, 8003));
networkParam.setPacketMagic(NulsConfig.MODULES_CONFIG.getCfgValue(NETWORK_SECTI
ON, NETWORK_MAGIC, 123456789));
networkParam.setMaxInCount(NulsConfig.MODULES_CONFIG.getCfgValue(NETWORK_SECTI
ON, NETWORK_NODE_MAX_IN, 30));
networkParam.setMaxOutCount(NulsConfig.MODULES_CONFIG.getCfgValue(NETWORK_SECT
ION, NETWORK_NODE_MAX_OUT, 10));
        networkParam.setLocalIps(IpUtil.getIps());
        String seedIp =
NulsConfig.MODULES_CONFIG.getCfgValue(NetworkConstant.NETWORK_SECTION,
NetworkConstant.NETWORK_SEED_IP, "192.168.1.131:8003");
        List<String> ipList = new ArrayList<>();
        for (String ip : seedIp.split(",")) {
            ipList.add(ip);
        }
        networkParam.setSeedIpList(ipList);
    }

    private void initOther() {
MagicNumberFilter.getInstance().addMagicNum(NetworkParam.getInstance().getPacketMagic());
     MessageFilterChain.getInstance().addFilter(MagicNumberFilter.getInstance());
        MessageManager.putMessage(HandshakeMessage.class);
        MessageManager.putMessage(GetVersionMessage.class);
        MessageManager.putMessage(VersionMessage.class);
        MessageManager.putMessage(GetNodesMessage.class);
        MessageManager.putMessage(NodesMessage.class);
        MessageManager.putMessage(GetNodesIpMessage.class);
        MessageManager.putMessage(NodesIpMessage.class);
        MessageManager.putMessage(P2PNodeMessage.class);
    }

    @Override
```

```
    public void start() {
        this.waitForDependencyRunning(DBConstant.MODULE_ID_DB,
ProtocolConstant.MODULE_ID_PROTOCOL);
        connectionManager.start();
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        nodeManager.start();
    }

    @Override
    public void shutdown() {

    }

    @Override
    public void destroy() {

    }

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

49:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\service\impl\NetworkServiceImpl.java
*/

```
package io.nuls.network.service.impl;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.manager.BroadcastHandler;
import io.nuls.network.manager.NodeManager;
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 io.nuls.protocol.message.base.BaseMessage;

import java.util.Collection;
import java.util.List;
import java.util.Map;

@Service
public class NetworkServiceImpl implements NetworkService {

    private NodeManager nodeManager = NodeManager.getInstance();

    private BroadcastHandler broadcastHandler = BroadcastHandler.getInstance();

    @Override
    public void removeNode(String nodeId) {
        nodeManager.removeHandshakeNode(nodeId);
    }

    @Override
    public Node getNode(String nodeId) {
        return nodeManager.getNode(nodeId);
    }

    @Override
    public Map<String, Node> getNodes() {
        return nodeManager.getNodes();
    }

    @Override
    public Collection<Node> getAvailableNodes() {
        return nodeManager.getAvailableNodes();
    }

    @Override
    public List<Node> getCanConnectNodes() {
        return nodeManager.getCanConnectNodes();
    }

    @Override
    public NodeGroup getNodeGroup(String groupName) {
```

```
        return nodeManager.getNodeGroup(groupName);
    }

    @Override
    public BroadcastResult sendToAllNode(BaseNulsData nulsData, boolean asyn, int percent) {
        BaseMessage baseMessage = (BaseMessage) nulsData;
        return broadcastHandler.broadcastToAllNode(baseMessage, null, asyn, percent);
    }

    @Override
    public BroadcastResult sendToAllNode(BaseNulsData nulsData, Node excludeNode, boolean
asyn, int percent) {
        BaseMessage baseMessage = (BaseMessage) nulsData;
        return broadcastHandler.broadcastToAllNode(baseMessage, excludeNode, asyn,percent);
    }

    @Override
    public BroadcastResult sendToNode(BaseNulsData nulsData, Node node, boolean asyn) {
        BaseMessage baseMessage = (BaseMessage) nulsData;
        return broadcastHandler.broadcastToNode(baseMessage, node, asyn);
    }

    @Override
    public BroadcastResult sendToGroup(BaseNulsData nulsData, String groupName, boolean
asyn) {
        BaseMessage baseMessage = (BaseMessage) nulsData;
        return broadcastHandler.broadcastToNodeGroup(baseMessage, groupName, asyn);
    }

    @Override
    public BroadcastResult sendToGroup(BaseNulsData nulsData, String groupName, Node
excludeNode, boolean asyn) {
        BaseMessage baseMessage = (BaseMessage) nulsData;
        return broadcastHandler.broadcastToNodeGroup(baseMessage, groupName, excludeNode,
asyn);
    }

    @Override
    public void reset() {
        Log.warn("------network reset");
        nodeManager.reset();
    }
```

```
    @Override
    public NetworkParam getNetworkParam() {
        return NetworkParam.getInstance();
    }
}

50:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\util\HeartBeatThread.java
 */

package io.nuls.network.util;

import io.nuls.core.tools.log.Log;
import io.nuls.network.message.impl.VersionMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

/**
 * @author: Niels Wang
 * @date: 2018/7/10
 */
public class HeartBeatThread implements Runnable {

    private final BaseNetworkMeesageHandler handler;
    private BlockingQueue<MessageContainer> queue = new LinkedBlockingQueue<>();

    public HeartBeatThread(BaseNetworkMeesageHandler handler) {
        this.handler = handler;
    }

    @Override
    public void run() {
        while (true) {
            try {
                process();
            } catch (Throwable e) {
                Log.error(e);
```

```
            }
        }
    }

    private void process() throws InterruptedException {
        MessageContainer mc = queue.take();
        handler.process(mc.getMessage(), mc.getNode());
    }

    public void offerMessage(BaseMessage message, Node node) {
        queue.offer(new MessageContainer(message, node));
    }
}

51:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\util\MessageContainer.java
 */
package io.nuls.network.util;

import io.nuls.network.model.Node;
import io.nuls.protocol.message.base.BaseMessage;

/**
 * @author Niels
 * @date 2018/7/10
 */
public class MessageContainer {

    private BaseMessage message;

    private Node node;

    public MessageContainer(BaseMessage message,Node node){
        this.message = message;
        this.node = node;
    }

    public BaseMessage getMessage() {
        return message;
    }

    public Node getNode() {
```

```
        return node;
    }
}

52:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\main\java\io\nuls\network\util\NetworkThreadPool.java
 */
package io.nuls.network.util;

import io.nuls.core.tools.log.Log;
import io.nuls.network.connection.netty.NettyClient;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.manager.ConnectionManager;
import io.nuls.network.manager.NetworkMessageHandlerFactory;
import io.nuls.network.manager.NodeManager;
import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.network.protocol.handler.BaseNetworkMeesageHandler;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.concurrent.*;

public class NetworkThreadPool {

    private static final ExecutorService executor = new ThreadPoolExecutor(0,
Integer.MAX_VALUE,
            60L, TimeUnit.SECONDS,
            new SynchronousQueue<Runnable>());

    public static void asynNetworkMessage(BaseMessage message, Node node, HeartBeatThread
heartBeatThread, NetworkMessageHandlerFactory messageHandlerFactory, ConnectionManager
connectionManager) {
        executor.submit(new Runnable() {
            @Override
            public void run() {
                if (message.getHeader().getMsgType() == NetworkConstant.NETWORK_VERSION) {
                    heartBeatThread.offerMessage(message, node);
                    return;
                }
                BaseNetworkMeesageHandler handler =
messageHandlerFactory.getHandler(message);
                try {
```

```
                NetworkEventResult messageResult = handler.process(message, node);
                connectionManager.processMessageResult(messageResult, node);
            } catch (Exception e) {
                e.printStackTrace();
                Log.error(e);
            }
        }
    });
}

public static void doConnect(Node node) {

    executor.submit(new Runnable() {
        @Override
        public void run() {
            NettyClient client = new NettyClient(node);
            client.start();
        }
    });
}
}
```

53:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-base\src\main\java\io\nuls\network\util\SendNodeInfoThread.java

```
 */

package io.nuls.network.util;

import io.nuls.core.tools.date.DateUtil;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.manager.NodeManager;

/**
 * @author: Niels Wang
 * @date: 2018/7/10
 */
public class SendNodeInfoThread implements Runnable {

    boolean running = false;
```

```
    private static SendNodeInfoThread instance = new SendNodeInfoThread();

    private NodeManager nodeManager = NodeManager.getInstance();

    private SendNodeInfoThread() {

    }

    public static SendNodeInfoThread getInstance() {
        return instance;
    }

    @Override
    public void run() {
        while (true) {
            try {
                nodeManager.broadNodeSever();
                Thread.sleep(10 * DateUtil.MINUTE_TIME);
            } catch (Throwable e) {
                Log.error(e);
            }
        }
    }

    public void start() {
        if (!running) {
            running = true;
            TaskManager.createAndRunThread(NetworkConstant.NETWORK_MODULE_ID, "send-
node-info", this);
        }
    }

}

54:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
base\src\test\java\io\nuls\network\test\NetworkTest.java
 */

package io.nuls.network.test;

import io.nuls.kernel.lite.annotation.Autowired;
```

```
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.protocol.message.GetNodesIpMessage;
import io.nuls.network.protocol.message.HandshakeMessage;
import io.nuls.network.protocol.message.NetworkMessageBody;
import io.nuls.network.protocol.message.NodeMessageBody;
import org.junit.Test;

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

public class NetworkTest {

    @Test
    public void testNetworkMessage() {

        try {
//            NetworkMessageBody body = new
NetworkMessageBody(NetworkConstant.HANDSHAKE_SEVER_TYPE, 4567,
//                    10001, NulsDigestData.calcDigestData ("a1b2c3d4e5gf6g7h8i9j10".getBytes()));
//            HandshakeMessage handshakeMessage = new HandshakeMessage(body);
//
System.out.println(handshakeMessage.getMsgBody().getBestBlockHash().getDigestHex());
//            HandshakeMessage handshakeMessage1 = new HandshakeMessage();
//            handshakeMessage1.parse(handshakeMessage.serialize());
//
System.out.println(handshakeMessage1.getMsgBody().getBestBlockHash().getDigestHex());

            List<String> list = new ArrayList<>();
            list.add("adsda");
            list.add("adsda");
            list.add("adsda");
            list.add("adsda");
            list.add("adsda");
            NodeMessageBody nodeMessageBody = new NodeMessageBody();
//            nodeMessageBody.setIpList(list);
            GetNodesIpMessage getNodesIpMessage = new
GetNodesIpMessage(nodeMessageBody);
            getNodesIpMessage.serialize();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
}
```

55:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-protocol\src\main\java\io\nuls\network\protocol\handler\BaseNetworkMeesageHandler.java
*/

```
package io.nuls.network.protocol.handler;

import io.nuls.network.model.NetworkEventResult;
import io.nuls.network.model.Node;
import io.nuls.protocol.message.base.BaseMessage;

public interface BaseNetworkMeesageHandler {

    NetworkEventResult process(BaseMessage message, Node node);
}
```

56:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-protocol\src\main\java\io\nuls\network\protocol\message\BaseNetworkMessage.java
*/

```
package io.nuls.network.protocol.message;

import io.nuls.kernel.model.BaseNulsData;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.protocol.message.base.BaseMessage;
import io.nuls.protocol.model.basic.NulsBytesData;

/**
 * @author Vivi
 */
public abstract class BaseNetworkMessage<T extends BaseNulsData> extends BaseMessage<T> {

    /**
     *
     * @param msgType
     */
    public BaseNetworkMessage(short msgType) {
        super(NetworkConstant.NETWORK_MODULE_ID, msgType);
    }
```

```
}
```

57:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-protocol\src\main\java\io\nuls\network\protocol\message\GetNodesIpMessage.java

```
 */

package io.nuls.network.protocol.message;


import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class GetNodesIpMessage extends BaseNetworkMessage<NodeMessageBody>{

    /**
     *
     */
    public GetNodesIpMessage() {
        super(NetworkConstant.NETWORK_GET_NODEIP);
    }

    @Override
    protected NodeMessageBody parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new NodeMessageBody());
    }

    public GetNodesIpMessage(NodeMessageBody body) {
        this();
        this.setMsgBody(body);
    }
}
```

58:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-protocol\src\main\java\io\nuls\network\protocol\message\GetNodesMessage.java

```
 */

package io.nuls.network.protocol.message;


import io.nuls.kernel.exception.NulsException;
```

```
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class GetNodesMessage extends BaseNetworkMessage<NodeMessageBody>{

    /**
     *
     */
    public GetNodesMessage() {
        super(NetworkConstant.NETWORK_GET_NODE);
    }

    @Override
    protected NodeMessageBody parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new NodeMessageBody());
    }

    public GetNodesMessage(NodeMessageBody body) {
        this();
        this.setMsgBody(body);
    }
}

59:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
protocol\src\main\java\io\nuls\network\protocol\message\GetVersionMessage.java
 */

package io.nuls.network.protocol.message;


import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class GetVersionMessage extends BaseNetworkMessage<NetworkMessageBody>{

    /**
     *
     */
    public GetVersionMessage() {
        super(NetworkConstant.NETWORK_GET_VERSION);
```

```
    }

    @Override
    protected NetworkMessageBody parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new NetworkMessageBody());
    }

    public GetVersionMessage(NetworkMessageBody body) {
        this();
        this.setMsgBody(body);
    }
}

60:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
protocol\src\main\java\io\nuls\network\protocol\message\HandshakeMessage.java
 */

package io.nuls.network.protocol.message;


import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class HandshakeMessage extends BaseNetworkMessage<NetworkMessageBody> {
    /**
     *
     */

    public HandshakeMessage() {
        super(NetworkConstant.NETWORK_HANDSHAKE);
    }

    @Override
    protected NetworkMessageBody parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new NetworkMessageBody());
    }

    public HandshakeMessage(NetworkMessageBody body) {
        this();
```

```
        this.setMsgBody(body);
    }

}

61:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
protocol\src\main\java\io\nuls\network\protocol\message\NetworkMessageBody.java
 */

package io.nuls.network.protocol.message;

import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
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.protostuff.Tag;

import java.io.IOException;

public class NetworkMessageBody extends BaseNulsData {

    private int handshakeType;

    private int severPort;

    private long bestBlockHeight;

    private NulsDigestData bestBlockHash;

    private long networkTime;

    private String nodeIp;

    private String version;

    public NetworkMessageBody() {
        this.version = NulsConfig.VERSION;
```

```
    }

    public NetworkMessageBody(int handshakeType, int severPort, long bestBlockHeight,
NulsDigestData bestBlockHash) {
        this.handshakeType = handshakeType;
        this.severPort = severPort;
        this.bestBlockHeight = bestBlockHeight;
        this.bestBlockHash = bestBlockHash;
        this.networkTime = TimeService.currentTimeMillis();
        this.version = NulsConfig.VERSION;
    }

    public NetworkMessageBody(int handshakeType, int severPort, long bestBlockHeight,
NulsDigestData bestBlockHash, String ip) {
        this(handshakeType, severPort, bestBlockHeight, bestBlockHash);
        this.nodeIp = ip;
    }

    @Override
    public int size() {
        int s = 0;
        s += SerializeUtils.sizeOfUint16(); // handshakeType
        s += SerializeUtils.sizeOfUint16(); // severPort
        s += SerializeUtils.sizeOfUint32(); // bestBlockHeight
        s += bestBlockHash.size();
        s += SerializeUtils.sizeOfUint48(); // networkTime
        s += SerializeUtils.sizeOfString(nodeIp);
        s += SerializeUtils.sizeOfString(version);
        return s;
    }

    /**
     * serialize important field
     */
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeUint16(handshakeType);
        stream.writeUint16(severPort);
        stream.writeUint32(bestBlockHeight);
        stream.write(bestBlockHash.serialize());
        stream.writeUint48(networkTime);
        stream.writeString(nodeIp);
```

```
    stream.writeString(version);
}

@Override
public void parse(NulsByteBuffer buffer) throws NulsException {
    handshakeType = buffer.readUint16();
    severPort = buffer.readUint16();
    bestBlockHeight = buffer.readUint32();
    bestBlockHash = buffer.readHash();
    networkTime = buffer.readUint48();
    nodeIp = buffer.readString();
    version = buffer.readString();
}

public int getHandshakeType() {
    return handshakeType;
}

public void setHandshakeType(int handshakeType) {
    this.handshakeType = handshakeType;
}

public int getSeverPort() {
    return severPort;
}

public void setSeverPort(int severPort) {
    this.severPort = severPort;
}

public long getBestBlockHeight() {
    return bestBlockHeight;
}

public void setBestBlockHeight(long bestBlockHeight) {
    this.bestBlockHeight = bestBlockHeight;
}

public NulsDigestData getBestBlockHash() {
    return bestBlockHash;
}
```

```
    public void setBestBlockHash(NulsDigestData bestBlockHash) {
        this.bestBlockHash = bestBlockHash;
    }

    public long getNetworkTime() {
        return networkTime;
    }

    public void setNetworkTime(long networkTime) {
        this.networkTime = networkTime;
    }

    public String getNodeIp() {
        return nodeIp;
    }

    public void setNodeIp(String nodeIp) {
        this.nodeIp = nodeIp;
    }

    public static void main(String[] args) throws IOException, NulsException {
        NetworkMessageBody networkMessageBody = new NetworkMessageBody();
        networkMessageBody.setSeverPort(1001);
        networkMessageBody.setNetworkTime(20003L);
        networkMessageBody.setHandshakeType(1);
        networkMessageBody.setBestBlockHeight(4003L);
        networkMessageBody.setBestBlockHash(new NulsDigestData());

        byte[] bytes = networkMessageBody.serialize();
        NetworkMessageBody n2 = new NetworkMessageBody();
        n2.parse(bytes,0);
    }

    public String getVersion() {
        return version;
    }

    public void setVersion(String version) {
        this.version = version;
    }
}
```

62:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-protocol\src\main\java\io\nuls\network\protocol\message\NodeMessageBody.java
 */

```
package io.nuls.network.protocol.message;

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.network.model.Node;

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

public class NodeMessageBody extends BaseNulsData {
//
//    private int length;
//
//    private List<String> ipList;

    private List<Node> nodeList;

    public NodeMessageBody() {

    }

    public NodeMessageBody(List nodeList) {
        this.nodeList = nodeList;
    }

    /**
     * serialize important field
     */
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
//        stream.writeUint16(length);
//        int count = ipList == null ? 0 : ipList.size();
//        stream.writeVarInt(count);
```

```
//        if (null != ipList) {
//            for (String ip : ipList) {
//                stream.writeString(ip);
//            }
//        }
        int count = nodeList == null ? 0 : nodeList.size();
        stream.writeVarInt(count);
        if (null != nodeList) {
            for (Node node : nodeList) {
                stream.writeNulsData(node);
            }
        }
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
//        length = byteBuffer.readUint16();
//
//        List<String> ipList = new ArrayList<>();
//        int count = (int) byteBuffer.readVarInt();
//        for (int i = 0; i < count; i++) {
//            ipList.add(byteBuffer.readString());
//        }
//        this.ipList = ipList;

        List<Node> nodeList = new ArrayList<>();
        int count = (int) byteBuffer.readVarInt();
        for (int i = 0; i < count; i++) {
            nodeList.add(byteBuffer.readNulsData(new Node()));
        }
        this.nodeList = nodeList;
    }

    @Override
    public int size() {
        int s = 0;
//        s += SerializeUtils.sizeOfUint16(); //length
//        s += SerializeUtils.sizeOfVarInt(ipList == null ? 0 : ipList.size());
//        if (null != ipList) {
//            for (String ip : ipList) {
//                s += SerializeUtils.sizeOfString(ip);
//            }
```

```
//        }

        s += SerializeUtils.sizeOfVarInt(nodeList == null ? 0 : nodeList.size());
        if (nodeList != null) {
            for (Node node : nodeList) {
                s += node.size();
            }
        }
        return s;
    }

//    public int getLength() {
//        return length;
//    }
//
//    public void setLength(int length) {
//        this.length = length;
//    }
//
//    public List<String> getIpList() {
//        return ipList;
//    }
//
//    public void setIpList(List<String> ipList) {
//        this.ipList = ipList;
//    }

    public List<Node> getNodeList() {
        return nodeList;
    }

    public void setNodeList(List<Node> nodeList) {
        this.nodeList = nodeList;
    }

}

63:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
protocol\src\main\java\io\nuls\network\protocol\message\NodesIpMessage.java
 */

package io.nuls.network.protocol.message;
```

```
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class NodesIpMessage extends BaseNetworkMessage<NodeMessageBody>{

    /**
     *
     */
    public NodesIpMessage() {
        super(NetworkConstant.NETWORK_NODEIP);
    }

    @Override
    protected NodeMessageBody parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new NodeMessageBody());
    }

    public NodesIpMessage(NodeMessageBody body) {
        this();
        this.setMsgBody(body);
    }
}

64:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
protocol\src\main\java\io\nuls\network\protocol\message\NodesMessage.java
 */

package io.nuls.network.protocol.message;


import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class NodesMessage extends BaseNetworkMessage<NodeMessageBody>{

    /**
     *
```

```
     */
    public NodesMessage() {
        super(NetworkConstant.NETWORK_NODE);
    }

    @Override
    protected NodeMessageBody parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new NodeMessageBody());
    }

    public NodesMessage(NodeMessageBody body) {
        this();
        this.setMsgBody(body);
    }
}
```

65:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-protocol\src\main\java\io\nuls\network\protocol\message\P2PNodeBody.java

```
 */

package io.nuls.network.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;

public class P2PNodeBody extends BaseNulsData {

    private String nodeIp;

    private int severPort;

    public P2PNodeBody() {

    }
```

```
public P2PNodeBody(String nodeIp, int severPort) {
    this.nodeIp = nodeIp;
    this.severPort = severPort;
}

@Override
public int size() {
    int s = 0;
    s += SerializeUtils.sizeOfUint16(); // severPort
    s += SerializeUtils.sizeOfString(nodeIp);
    return s;
}

/**
 * serialize important field
 */
@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeUint16(severPort);
    stream.writeString(nodeIp);
}

@Override
public void parse(NulsByteBuffer buffer) throws NulsException {
    severPort = buffer.readUint16();
    nodeIp = buffer.readString();
}


public int getSeverPort() {
    return severPort;
}

public void setSeverPort(int severPort) {
    this.severPort = severPort;
}

public String getNodeIp() {
    return nodeIp;
}
```

```
    public void setNodeIp(String nodeIp) {
        this.nodeIp = nodeIp;
    }

    public String getId() {
        return nodeIp + ":" + severPort;
    }

    @Override
    public String toString() {
        return "P2PNodeBody{" +
                "nodeIp='" + nodeIp + '\'' +
                ", severPort=" + severPort +
                '}';
    }
}

66:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
protocol\src\main\java\io\nuls\network\protocol\message\P2PNodeMessage.java
 */

package io.nuls.network.protocol.message;


import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class P2PNodeMessage extends BaseNetworkMessage<P2PNodeBody> {
    /**
     *
     */

    public P2PNodeMessage() {
        super(NetworkConstant.NETWORK_P2P_NODE);
    }

    @Override
    protected P2PNodeBody parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException
{
        return byteBuffer.readNulsData(new P2PNodeBody());
    }
```

```
    public P2PNodeMessage(P2PNodeBody body) {
        this();
        this.setMsgBody(body);
    }

}
```

67:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-protocol\src\main\java\io\nuls\network\protocol\message\VersionMessage.java

```
 */

package io.nuls.network.protocol.message;


import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.network.constant.NetworkConstant;

public class VersionMessage extends BaseNetworkMessage<NetworkMessageBody>{

    /**
     *
     */
    public VersionMessage() {
        super(NetworkConstant.NETWORK_VERSION);
    }

    @Override
    protected NetworkMessageBody parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new NetworkMessageBody());
    }

    public VersionMessage(NetworkMessageBody body) {
        this();
        this.setMsgBody(body);
    }
}
```

68:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-rpc\src\main\java\io\nuls\network\rpc\cmd\GetNetInfoProcessor.java

```
 */

package io.nuls.network.rpc.cmd;

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.RestFulUtils;

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription());
        return bulider.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getnetinfo  --get network information";
    }

    @Override
    public boolean argsValidate(String[] args) {
        if(args.length !=1){
            return false;
        }
        return true;
    }
```

```
    @Override
    public CommandResult execute(String[] args) {
        RpcClientResult result = restFul.get("/network/info", null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}
```

69:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-rpc\src\main\java\io\nuls\network\rpc\cmd\GetNetNodesProcessor.java

```
 */

package io.nuls.network.rpc.cmd;

import io.nuls.kernel.model.CommandResult;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandBuilder;
import io.nuls.kernel.utils.RestFulUtils;

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder bulider = new CommandBuilder();
        bulider.newLine(getCommandDescription());
        return bulider.toString();
    }

    @Override
```

```
    public String getCommandDescription() {
        return "getnetnodes --get IP of network nodes";
    }

    @Override
    public boolean argsValidate(String[] args) {
        if(args.length !=1){
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        RpcClientResult result = restFul.get("/network/nodes", null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(CommandResult.dataTransformList(result));
    }
}

70:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
rpc\src\main\java\io\nuls\network\rpc\model\NetworkInfoDto.java
 */

package io.nuls.network.rpc.model;

import io.nuls.core.tools.date.DateUtil;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

@ApiModel(value = "networkInfoJSON")
public class NetworkInfoDto {

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

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

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

```
private String timeOffset;

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

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

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

public NetworkInfoDto() {

}

public NetworkInfoDto(long localBestHeight, long netBestHeight, long offsetTime) {
    this.localBestHeight = localBestHeight;
    this.netBestHeight = netBestHeight;
    this.timeOffset = DateUtil.getOffsetStringDate(offsetTime);
}

public Long getLocalBestHeight() {
    return localBestHeight;
}

public void setLocalBestHeight(Long localBestHeight) {
    this.localBestHeight = localBestHeight;
}

public Long getNetBestHeight() {
    return netBestHeight;
}

public void setNetBestHeight(Long netBestHeight) {
    this.netBestHeight = netBestHeight;
}

public String getTimeOffset() {
    return timeOffset;
}

public void setTimeOffset(String timeOffset) {
```

```
        this.timeOffset = timeOffset;
    }

    public int getInCount() {
        return inCount;
    }

    public void setInCount(int inCount) {
        this.inCount = inCount;
    }

    public int getOutCount() {
        return outCount;
    }

    public void setOutCount(int outCount) {
        this.outCount = outCount;
    }

    public boolean isMastUpGrade() {
        return mastUpGrade;
    }

    public void setMastUpGrade(boolean mastUpGrade) {
        this.mastUpGrade = mastUpGrade;
    }
}

71:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
rpc\src\main\java\io\nuls\network\rpc\model\NodeDto.java
 */
package io.nuls.network.rpc.model;

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

@ApiModel(value = "nodeInfoJSON")
public class NodeDto {

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

```
    private int port;

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

    public int getPort() {
        return port;
    }

    public void setPort(int port) {
        this.port = port;
    }
}

72:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
rpc\src\main\java\io\nuls\network\rpc\resource\NetworkResource.java
 */

package io.nuls.network.rpc.resource;

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.Result;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.network.cache.NodeCacheManager;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.network.constant.NetworkParam;
import io.nuls.network.model.Node;
import io.nuls.network.model.NodeGroup;
import io.nuls.network.rpc.model.NetworkInfoDto;
import io.nuls.network.rpc.model.NodeDto;
import io.nuls.network.service.NetworkService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
```

```
import io.swagger.annotations.ApiResponses;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.util.*;

@Path("/network")
@Api(value = "/network", description = "network")
@Component
public class NetworkResource {

    @Autowired
    private NetworkService networkService;

    private NodeCacheManager nodeCacheManager = NodeCacheManager.getInstance();

    @GET
    @Path("/info/")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = NetworkInfoDto.class)
    })
    public RpcClientResult getNetworkInfo() {
        NetworkInfoDto info = new
NetworkInfoDto(NulsContext.getInstance().getBestBlock().getHeader().getHeight(),
                NulsContext.getInstance().getNetBestBlockHeight(), TimeService.getNetTimeOffset());

        NodeGroup inGroup =
networkService.getNodeGroup(NetworkConstant.NETWORK_NODE_IN_GROUP);
        NodeGroup outGroup =
networkService.getNodeGroup(NetworkConstant.NETWORK_NODE_OUT_GROUP);
        int count = 0;
        for (Node node : inGroup.getNodes().values()) {
            if (node.getStatus() == Node.HANDSHAKE) {
                count += 1;
            }
        }
        info.setInCount(count);
```

```
    count = 0;
    for (Node node : outGroup.getNodes().values()) {
        if (node.getStatus() == Node.HANDSHAKE) {
            count += 1;
        }
    }
    info.setOutCount(count);
    info.setMastUpGrade(NulsContext.mastUpGrade);
    Result result = Result.getSuccess();
    result.setData(info);
    return result.toRpcClientResult();
}

@GET
@Path("/nodes")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("IP [3.7.2]")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = String[].class)
})
public RpcClientResult getNode() {
    Set<String> ipSet = nodeCacheManager.getIpSet();
    if (ipSet == null || ipSet.isEmpty()) {
        ipSet = new HashSet<>();
        List<Node> nodeList = networkService.getCanConnectNodes();
        for (Node node : nodeList) {
            ipSet.add(node.getIp());
        }
        nodeCacheManager.cacheIpSet(ipSet);
    }
    Result result = Result.getSuccess();
    Map<String, Set<String>> map = new HashMap<>();
    map.put("list", ipSet);
    result.setData(map);
    return result.toRpcClientResult();
}

@GET
@Path("/peers")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("[3.7.2]")
@ApiResponses(value = {
```

```
        @ApiResponse(code = 200, message = "success", response = NodeDto.class)
    })
    public RpcClientResult getPeers() {
        List<Node> nodeList = networkService.getCanConnectNodes();
        Result result = Result.getSuccess();
        List<NodeDto> dtoList = new ArrayList<>();
        for (Node node : nodeList) {
            NodeDto dto = new NodeDto();
            dto.setIp(node.getIp());
            dto.setPort(node.getPort());
            dtoList.add(dto);
        }
        Map<String, List<NodeDto>> map = new HashMap<>();
        map.put("list", dtoList);
        result.setData(map);
        return result.toRpcClientResult();
    }
}
```

73:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-storage\src\main\java\io\nuls\network\storage\constant\NetworkStorageConstant.java
*/

```
package io.nuls.network.storage.constant;

public interface NetworkStorageConstant {

    String DB_NAME_NETWORK_NODE = "network_node";

    String DB_NAME_EXTERNAL_IP = "external_ip";

    long NODE_DB_CACHE_SIZE = 1024 * 1024;
}
```

74:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-storage\src\main\java\io\nuls\network\storage\po\NetworkTransferTool.java
*/

```
package io.nuls.network.storage.po;

import io.nuls.network.model.Node;
```

```
public class NetworkTransferTool {

    public static NodePo toPojo(Node node) {
        NodePo po = new NodePo();
        po.setId(node.getId());
        po.setIp(node.getIp());
        po.setPort(node.getSeverPort());
        po.setLastTime(node.getLastTime());
        po.setLastFailTime(node.getLastFailTime());
        po.setFailCount(node.getFailCount());
        return po;
    }

    public static void toPojo(Node node, NodePo po) {
        po.setId(node.getId());
        po.setIp(node.getIp());
        po.setPort(node.getSeverPort());
        po.setLastTime(node.getLastTime());
        po.setLastFailTime(node.getLastFailTime());
        po.setFailCount(node.getFailCount());
    }

    public static Node toNode(NodePo po) {
        Node node = new Node();
        node.setId(po.getId());
        node.setIp(po.getIp());
        node.setPort(po.getPort());
        node.setSeverPort(po.getPort());
        node.setFailCount(0);
        node.setLastTime(po.getLastTime());
        node.setLastFailTime(po.getLastFailTime());
        return node;
    }
}

75:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
storage\src\main\java\io\nuls\network\storage\po\NodePo.java
 */

package io.nuls.network.storage.po;

public class NodePo {
```

```
private String id;

private String ip;

private Integer port;

private Long lastTime;

private Long lastFailTime;

private Integer failCount;

public String getId() {
    return id;
}

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

public String getIp() {
    return ip;
}

public void setIp(String ip) {
    this.ip = ip;
}

public Integer getPort() {
    return port;
}

public void setPort(Integer port) {
    this.port = port;
}

public Long getLastFailTime() {
    return lastFailTime;
}

public void setLastFailTime(Long lastFailTime) {
```

```
        this.lastFailTime = lastFailTime;
    }

    public Integer getFailCount() {
        return failCount;
    }

    public void setFailCount(Integer failCount) {
        this.failCount = failCount;
    }

    public Long getLastTime() {
        return lastTime;
    }

    public void setLastTime(Long lastTime) {
        this.lastTime = lastTime;
    }
}

76:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
storage\src\main\java\io\nuls\network\storage\service\impl\NetworkStorageServiceImpl.java
 */

package io.nuls.network.storage.service.impl;


import io.nuls.db.constant.DBConstant;
import io.nuls.db.service.DBService;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.network.model.Node;
import io.nuls.network.storage.constant.NetworkStorageConstant;
import io.nuls.network.storage.po.NetworkTransferTool;
import io.nuls.network.storage.po.NodePo;
import io.nuls.network.storage.service.NetworkStorageService;

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

```
import java.util.Set;

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

@Component
public class NetworkStorageServiceImpl implements NetworkStorageService, InitializingBean {

    @Autowired
    private DBService dbService;

    @Override
    public List<Node> getLocalNodeList() {
        List<NodePo> poList =
getDbService().values(NetworkStorageConstant.DB_NAME_NETWORK_NODE, NodePo.class);
        if (poList == null) {
            return new ArrayList<>();
        }
        List<Node> nodeList = new ArrayList<>();
        for (NodePo po : poList) {
            nodeList.add(NetworkTransferTool.toNode(po));
        }
        return nodeList;
    }

    @Override
    public List<Node> getLocalNodeList(int size, Set<String> ipSet) {
        List<Node> nodeList = new ArrayList<>();
        List<NodePo> poList =
getDbService().values(NetworkStorageConstant.DB_NAME_NETWORK_NODE, NodePo.class);
        if (poList == null) {
            return nodeList;
        }
        int count = 0;
        for (int i = poList.size() - 1; i >= 0; i--) {
            NodePo po = poList.get(i);
            if (ipSet.contains(po.getIp())) {
                continue;
            }
            nodeList.add(NetworkTransferTool.toNode(po));
            count++;
            if (count >= size) {
                break;
```

```
            }
        }
        return nodeList;
    }

    @Override
    public void saveNode(Node node) {
        NodePo po =
getDbService().getModel(NetworkStorageConstant.DB_NAME_NETWORK_NODE,
bytes(node.getId()), NodePo.class);
        if (po != null) {
            NetworkTransferTool.toPojo(node, po);
        } else {
            po = NetworkTransferTool.toPojo(node);
        }
        getDbService().putModel(NetworkStorageConstant.DB_NAME_NETWORK_NODE,
bytes(node.getId()), po);
    }

    @Override
    public void deleteNode(String nodeId) {
        getDbService().delete(NetworkStorageConstant.DB_NAME_NETWORK_NODE,
bytes(nodeId));
    }

    @Override
    public void saveExternalIp(String ip) {
        getDbService().put(DBConstant.BASE_AREA_NAME,
NetworkStorageConstant.DB_NAME_EXTERNAL_IP.getBytes(), ip.getBytes());
    }

    @Override
    public String getExternalIp() {
        byte[] bytes = getDbService().get(DBConstant.BASE_AREA_NAME,
NetworkStorageConstant.DB_NAME_EXTERNAL_IP.getBytes());
        if (bytes != null) {
            return new String(bytes);
        }
        return null;
    }
```

```
    private DBService getDbService() {
        if (dbService == null) {
            dbService = NulsContext.getServiceBean(DBService.class);
        }
        return dbService;
    }

    @Override
    public void afterPropertiesSet() throws NulsException {
        getDbService().createArea(NetworkStorageConstant.DB_NAME_NETWORK_NODE);
    }
}

77:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\base\network-
storage\src\main\java\io\nuls\network\storage\service\NetworkStorageService.java
 */

package io.nuls.network.storage.service;

import io.nuls.network.model.Node;

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

public interface NetworkStorageService {

    List<Node> getLocalNodeList();

    List<Node> getLocalNodeList(int size, Set<String> ipSet);

    void saveNode(Node node);

    void deleteNode(String nodeId);

    void saveExternalIp(String ip);

    String getExternalIp();
}

78:F:\git\coin\nuls\nuls-1.1.3\nuls\network-
module\network\src\main\java\io\nuls\network\constant\NetworkConstant.java
 */
```

```
package io.nuls.network.constant;

public interface NetworkConstant {

    short NETWORK_MODULE_ID = 4;


    /**
     * -----------[netty configs ]------------
     */
    int READ_IDEL_TIME_OUT = 0;
    int WRITE_IDEL_TIME_OUT = 0;
    int ALL_IDEL_TIME_OUT = 100;
    int MAX_FRAME_LENGTH = 10 * 1024 * 1024;
    int CONNETCI_TIME_OUT = 6000;
    int SAME_IP_MAX_COUNT = 10;
    int CONNECT_FAIL_MAX_COUNT = 6;

    /**
     * --------[network configs] -------
     */
    String NETWORK_SECTION = "network";
    String NETWORK_SERVER_PORT = "network.server.port";
    String NETWORK_MAGIC = "network.magic";
    String NETWORK_NODE_MAX_IN = "network.max.in";
    String NETWORK_NODE_MAX_OUT = "network.max.out";
    String NETWORK_SEED_IP = "network.seed.ip";
    String NETWORK_NODE_IN_GROUP = "inGroup";
    String NETWORK_NODE_OUT_GROUP = "outGroup";
    String CACHE_P2P_NODE = "cacheNode";
    String CACHE_P2P_IP = "cacheIP";

    int HANDSHAKE_SEVER_TYPE = 2;
    int HANDSHAKE_CLIENT_TYPE = 1;

    //network message type
    short NETWORK_GET_VERSION = 1;
    short NETWORK_VERSION = 2;
    short NETWORK_GET_NODE = 3;
    short NETWORK_NODE = 4;
    short NETWORK_GET_NODEIP = 5;
```

```
    short NETWORK_NODEIP = 6;
    short NETWORK_HANDSHAKE = 7;
    short NETWORK_P2P_NODE = 8;

}
```

79:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\network\src\main\java\io\nuls\network\constant\NetworkErrorCode.java

```
 *
 */
package io.nuls.network.constant;


import io.nuls.kernel.constant.ErrorCode;
import io.nuls.kernel.constant.KernelErrorCode;

/**
 * Created by Niels on 2017/9/27.
 */
public interface NetworkErrorCode extends KernelErrorCode {

    ErrorCode NET_SERVER_START_ERROR = ErrorCode.init("40001");
    ErrorCode NET_MESSAGE_ERROR = ErrorCode.init("40002");
    ErrorCode NET_MESSAGE_XOR_ERROR = ErrorCode.init("40003");
    ErrorCode NET_MESSAGE_LENGTH_ERROR = ErrorCode.init("40004");
    ErrorCode NET_NODE_GROUP_ALREADY_EXISTS = ErrorCode.init("40006");
    ErrorCode NET_NODE_AREA_ALREADY_EXISTS = ErrorCode.init("40007");
    ErrorCode NET_NODE_GROUP_NOT_FOUND = ErrorCode.init("40008");
    ErrorCode NET_NODE_AREA_NOT_FOUND = ErrorCode.init("40009");
    ErrorCode NET_NODE_NOT_FOUND = ErrorCode.init("40010");
    ErrorCode NET_BROADCAST_FAIL = ErrorCode.init("40011");
    ErrorCode NET_BROADCAST_NODE_EMPTY = ErrorCode.init("40012");
    ErrorCode NET_NODE_DEAD = ErrorCode.init("40013");
    ErrorCode NET_NODE_MISS_CHANNEL = ErrorCode.init("40014");
}
```

80:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\network\src\main\java\io\nuls\network\constant\NetworkParam.java

```
 */

package io.nuls.network.constant;
```

```
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

public class NetworkParam {

    private static NetworkParam instance = new NetworkParam();

    public static NetworkParam getInstance() {
        return instance;
    }

    private NetworkParam() {
    }

    private int port;

    private long packetMagic;

    private int maxInCount;

    private int maxOutCount;

    private Set<String> localIps;

    private List<String> seedIpList;

    public int getPort() {
        return port;
    }

    public void setPort(int port) {
        this.port = port;
    }

    public long getPacketMagic() {
        return packetMagic;
    }

    public void setPacketMagic(long packetMagic) {
        this.packetMagic = packetMagic;
```

```
    }

    public int getMaxInCount() {
        return maxInCount;
    }

    public void setMaxInCount(int maxInCount) {
        this.maxInCount = maxInCount;
    }

    public int getMaxOutCount() {
        return maxOutCount;
    }

    public void setMaxOutCount(int maxOutCount) {
        this.maxOutCount = maxOutCount;
    }

    public Set<String> getLocalIps() {
        return localIps;
    }

    public void setLocalIps(Set<String> localIps) {
        this.localIps = localIps;
    }

    public List<String> getSeedIpList() {
        return seedIpList;
    }

    public void setSeedIpList(List<String> seedIpList) {
        this.seedIpList = seedIpList;
    }

}

81:F:\git\coin\nuls\nuls-1.1.3\nuls\network-
module\network\src\main\java\io\nuls\network\model\BroadcastResult.java
 *
 */
package io.nuls.network.model;
```

```
import io.nuls.kernel.constant.ErrorCode;

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

/**
 * @author vivi
 */
public class BroadcastResult implements Cloneable {

    private boolean success;

    private ErrorCode errorCode;

    private String hash;

    private List<Node> broadcastNodes;

    private int waitReplyCount;

    private int repliedCount;

    public BroadcastResult() {
        broadcastNodes = new ArrayList<>();
    }

    public BroadcastResult(boolean success, ErrorCode errorCode) {
        this();
        this.success = success;
        this.errorCode = errorCode;
    }

    public BroadcastResult(boolean success, ErrorCode errorCode, List<Node> broadcastNodes) {
        this(success, errorCode);
        this.broadcastNodes = broadcastNodes;
    }

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean success) {
```

```
    this.success = success;
}

public String getHash() {
    return hash;
}

public void setHash(String hash) {
    this.hash = hash;
}

public List<Node> getBroadcastNodes() {
    return broadcastNodes;
}

public void setBroadcastNodes(List<Node> broadcastNodes) {
    this.broadcastNodes = broadcastNodes;
}

public int getWaitReplyCount() {
    return waitReplyCount;
}

public void setWaitReplyCount(int waitReplyCount) {
    this.waitReplyCount = waitReplyCount;
}

public int getRepliedCount() {
    return repliedCount;
}

public void setRepliedCount(int repliedCount) {
    this.repliedCount = repliedCount;
}

public ErrorCode getErrorCode() {
    return errorCode;
}

public void setErrorCode(ErrorCode errorCode) {
    this.errorCode = errorCode;
}
```

```
    @Override
    public Object clone() {
        BroadcastResult result = new BroadcastResult();
        result.setHash(this.hash);
        result.setSuccess(this.success);
        result.setWaitReplyCount(this.waitReplyCount);
        result.setRepliedCount(this.repliedCount);
        result.setBroadcastNodes(this.broadcastNodes);
        result.setErrorCode(this.getErrorCode());
        return result;
    }


}

82:F:\git\coin\nuls\nuls-1.1.3\nuls\network-
module\network\src\main\java\io\nuls\network\model\NetworkEventResult.java
 */

package io.nuls.network.model;

import io.nuls.kernel.model.BaseNulsData;

public class NetworkEventResult {

    private boolean success;

    private BaseNulsData replyMessage;

    public NetworkEventResult(boolean success, BaseNulsData replyMessage) {
        this.success = success;
        this.replyMessage = replyMessage;
    }

    public boolean isSuccess() {
        return success;
    }

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

```
    public BaseNulsData getReplyMessage() {
        return replyMessage;
    }

    public void setReplyMessage(BaseNulsData replyMessage) {
        this.replyMessage = replyMessage;
    }
}
```

83:F:\git\coin\nuls\nuls-1.1.3\nuls\network-module\network\src\main\java\io\nuls\network\model\Node.java

```
 *
 */
package io.nuls.network.model;

import io.netty.channel.Channel;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

/**
 * @author vivi
 */
public class Node extends BaseNulsData {

    private String id;

    private String ip;

    private Integer port;

    private Integer severPort = 0;
```

```
private long magicNumber;

private Long lastTime;

private Long lastFailTime;

private Integer failCount;

private Long bestBlockHeight;

private NulsDigestData bestBlockHash;

private Set<String> groupSet;

private long timeOffset;

private String externalIp;

private boolean canConnect;

private boolean testConnect;

private Channel channel;

@Override
public int size() {
    int s = 0;
    s += SerializeUtils.sizeOfUint32();
    s += SerializeUtils.sizeOfUint16();
    s += SerializeUtils.sizeOfString(ip);
    return s;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeUint32(magicNumber);
    stream.writeUint16(port);
    stream.writeString(ip);
}

@Override
```

```
public void parse(NulsByteBuffer buffer) throws NulsException {
    magicNumber = buffer.readUint32();
    severPort = buffer.readUint16();
    port = severPort;
    ip = buffer.readString();
    this.groupSet = ConcurrentHashMap.newKeySet();
}

/**
 * 1: inNode ,  2: outNode
 */
public final static int IN = 1;
public final static int OUT = 2;
private int type;

/**
 * 0: wait , 1: connecting, 2: handshake 3: close
 */
public final static int WAIT = 0;
public final static int CONNECT = 1;
public final static int HANDSHAKE = 2;
public final static int BAD = 3;
private volatile int status;


public Node() {
    this.status = WAIT;
    this.canConnect = false;
    groupSet = ConcurrentHashMap.newKeySet();
}

public Node(String ip, int port, int type) {
    this();
    this.ip = ip;
    this.port = port;
    if (type == Node.OUT) {
        this.severPort = port;
    }
    this.type = type;
}

public Node(String ip, int port, int severPort, int type) {
```

```
    this(ip, port, type);
    this.severPort = severPort;
}

public Node(String id, String ip, int port, int serverPort, int type) {
    this(ip, port, serverPort, type);
    this.id = id;
}

public void destroy() {
    this.lastFailTime = TimeService.currentTimeMillis();
    this.setFailCount(this.getFailCount() + 1);
    this.channel = null;
    this.status = Node.WAIT;
}

public boolean isHandShake() {
    return this.status == Node.HANDSHAKE;
}

public boolean isAlive() {
    return this.status == Node.CONNECT || status == Node.HANDSHAKE;
}

public void addToGroup(NodeGroup nodeGroup) {
    if (nodeGroup != null) {
        this.groupSet.add(nodeGroup.getName());
    }
}

public void removeFromGroup(NodeGroup nodeGroup) {
    if (nodeGroup != null) {
        this.groupSet.remove(nodeGroup.getName());
    }
}

public void addGroup(String groupName) {
    this.groupSet.add(groupName);
}

@Override
public String toString() {
```

```
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    sb.append("id:" + getId() + ",");
    sb.append("type:" + type + ",");
    sb.append("status:" + status + ",");
    sb.append("canConnect:" + canConnect + ",");
    sb.append("failCount:" + failCount + "}");

    return sb.toString();
}

@Override
public boolean equals(Object obj) {
    Node other = (Node) obj;
    if (StringUtils.isBlank(other.getId())) {
        return false;
    }
    return other.getId().equals(this.getId());
}

public int getType() {
    return type;
}

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

public String getIp() {
    return ip;
}

public void setIp(String ip) {
    this.ip = ip;
}

public int getStatus() {
    return status;
}

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

```
}

public Long getLastTime() {
    return lastTime;
}

public void setLastTime(Long lastTime) {
    this.lastTime = lastTime;
}

public Integer getFailCount() {
    if (failCount == null) {
        failCount = 0;
    }
    return failCount;
}

public void setFailCount(Integer failCount) {
    this.failCount = failCount;
}

public Integer getPort() {
    return port;
}

public void setPort(Integer port) {
    this.port = port;
}

public long getMagicNumber() {
    return magicNumber;
}

public void setMagicNumber(long magicNumber) {
    this.magicNumber = magicNumber;
}

public int getGroupCount(String groupName) {
    return this.groupSet.size();
}

public Set<String> getGroupSet() {
```

```
    return this.groupSet;
}

public Long getLastFailTime() {
    if (lastFailTime == null) {
        lastFailTime = 0L;
    }
    return lastFailTime;
}

public void setLastFailTime(Long lastFailTime) {
    this.lastFailTime = lastFailTime;
}

public String getId() {
    return ip + ":" + port;
}

public String getPoId() {
    if (severPort == null || severPort == 0) {
        severPort = port;
    }
    id = ip + ":" + severPort;
    return id;
}

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

public Integer getSeverPort() {
    return severPort;
}

public void setSeverPort(Integer severPort) {
    this.severPort = severPort;
}

public boolean isCanConnect() {
    return canConnect;
}
```

```
public void setCanConnect(boolean canConnect) {
    this.canConnect = canConnect;
}

public long getBestBlockHeight() {
    return bestBlockHeight;
}

public NulsDigestData getBestBlockHash() {
    return bestBlockHash;
}

public void setBestBlockHeight(Long bestBlockHeight) {
    this.bestBlockHeight = bestBlockHeight;
}

public void setBestBlockHash(NulsDigestData bestBlockHash) {
    this.bestBlockHash = bestBlockHash;
}

public long getTimeOffset() {
    return timeOffset;
}

public void setTimeOffset(long timeOffset) {
    this.timeOffset = timeOffset;
}

public String getExternalIp() {
    return externalIp;
}

public void setExternalIp(String externalIp) {
    this.externalIp = externalIp;
}

public boolean isTestConnect() {
    return testConnect;
}

public void setTestConnect(boolean testConnect) {
    this.testConnect = testConnect;
```

```
    }

    public Channel getChannel() {
        return channel;
    }

    public void setChannel(Channel channel) {
        this.channel = channel;
    }
}

84:F:\git\coin\nuls\nuls-1.1.3\nuls\network-
module\network\src\main\java\io\nuls\network\model\NodeArea.java
 *
 */
package io.nuls.network.model;

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.network.constant.NetworkErrorCode;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

/**
 * author Facjas
 * date 2018/1/19.
 */
public class NodeArea {
    private String areaName;
    private Map<String, NodeGroup> nodegroups = new ConcurrentHashMap<>();


    public NodeArea(String areaName){
        this.areaName = areaName;
        nodegroups = new ConcurrentHashMap<>();
    }
    public NodeArea(String areaName, Map<String,NodeGroup> nodegroups){
        this(areaName);
        this.nodegroups = nodegroups;
        for(NodeGroup ng : nodegroups.values()){
            addGroup(ng);
```

```
            ng.addtoArea(this);
        }
    }

    public String getAreaName(){
        return areaName;
    }

    public void setAreaName(String areaName){
        if(!StringUtils.isNull(areaName)) {
            this.areaName = areaName;
        }
    }

    public NodeGroup getNodeGroup(String groupName){
        if(!StringUtils.isNull(groupName)) {
            return nodegroups.get(groupName);
        }
        return null;
    }

    public boolean hasGroup(String groupName){
        if(!StringUtils.isNull(groupName)) {
            return nodegroups.containsKey(groupName);
        }
        return false;
    }

    public void addGroup(String groupName, NodeGroup nodeGroup){
        if(!StringUtils.isNull(groupName) && nodeGroup !=null){
            if(nodegroups.containsKey(groupName)){
                throw new
NulsRuntimeException(NetworkErrorCode.NET_NODE_AREA_ALREADY_EXISTS);
            }
            nodegroups.put(groupName,nodeGroup);
            nodeGroup.addtoArea(this);
        }
    }

    public void addGroup(NodeGroup nodeGroup){
        addGroup(nodeGroup.getName(),nodeGroup);
    }
```

```
    public void removeGroup(String groupName){
        if(!StringUtils.isNull(groupName)){
            if(!nodegroups.containsKey(groupName)){
                return;
            }
            NodeGroup nodeGroup = nodegroups.get(groupName);
            nodeGroup.removeFromArea(this);
            nodegroups.remove(groupName);
        }
    }

    @Override
    public String toString(){
        StringBuilder sb = new StringBuilder();
        sb.append("{areaName:'").append(this.getAreaName()).append("'}");
        return sb.toString();
    }
}

85:F:\git\coin\nuls\nuls-1.1.3\nuls\network-
module\network\src\main\java\io\nuls\network\model\NodeGroup.java
 *
 */
package io.nuls.network.model;

import java.util.Map;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

/**
 * @author vivi
 */
public class NodeGroup {
    private String groupName;
    private Set<NodeArea> areaSet;
    private Map<String, Node> nodeMap;

    public NodeGroup(String groupName) {
        this.groupName = groupName;
        nodeMap = new ConcurrentHashMap<>();
        areaSet = ConcurrentHashMap.newKeySet();
```

```
}

public Map<String, Node> getNodes() {
    return nodeMap;
}

public void addNode(Node p) {
    if (nodeMap.containsKey(p.getId())) {
        return;
    }
    this.nodeMap.put(p.getId(), p);
    p.addToGroup(this);
}

public void removeNode(Node node) {
    this.nodeMap.remove(node.getId());
}

public void removeNode(String nodeId) {
    this.nodeMap.remove(nodeId);
}

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

public void removeAll() {
    nodeMap = new ConcurrentHashMap<>();
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{NodeGroup:{groupName:'").append(this.getName()).append("',");
    sb.append("nodeMap:[");
    for (Node n : nodeMap.values()) {
        sb.append(n.toString()).append(",");
    }
    sb.deleteCharAt(sb.length() - 1);

    sb.append("],areaSet:[");
    for (NodeArea na : areaSet) {
```

```
            sb.append(na.toString()).append(",");
        }
        sb.deleteCharAt(sb.length() - 1);
        sb.append("]}");

        return sb.toString();
    }


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

    public String getName() {
        return groupName;
    }

    public Set<NodeArea> getAreaSet() {
        return this.areaSet;
    }

    public int getAreaCount() {
        return this.areaSet.size();
    }

    public void addtoArea(NodeArea nodeArea) {
        if (nodeArea != null) {
            this.areaSet.add(nodeArea);
        }
    }

    public void removeFromArea(NodeArea nodeArea) {
        if (nodeArea != null) {
            this.areaSet.remove(nodeArea);
        }
    }
}

86:F:\git\coin\nuls\nuls-1.1.3\nuls\network-
module\network\src\main\java\io\nuls\network\module\AbstractNetworkModule.java
 */
```

```
package io.nuls.network.module;

import io.nuls.kernel.module.BaseModuleBootstrap;
import io.nuls.network.constant.NetworkConstant;

public abstract class AbstractNetworkModule extends BaseModuleBootstrap {

    protected AbstractNetworkModule() {
        super(NetworkConstant.NETWORK_MODULE_ID);
    }
}

87:F:\git\coin\nuls\nuls-1.1.3\nuls\network-
module\network\src\main\java\io\nuls\network\service\NetworkService.java
 *
 */

package io.nuls.network.service;

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 java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Set;

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

    /**
     *
     * Disconnect the connection with the node
     *
     * @param nodeId the id of node
     */
    void removeNode(String nodeId);
```

```
/**
 *
 * get node by id
 *
 * @param nodeId the id of node
 * @return Node
 */
Node getNode(String nodeId);

/**
 *
 * get all nodes
 *
 * @return Map
 */
Map<String, Node> getNodes();

/**
 *
 * get connected nodes
 *
 * @return Collection
 */
Collection<Node> getAvailableNodes();

/**
 *
 * get connectable nodes
 *
 * @return List
 */
List<Node> getCanConnectNodes();

/**
 *
 * get NodeGroup by name
 *
 * @param  groupName groupName
 * @return NodeGroup
 */
NodeGroup getNodeGroup(String groupName);
```

```
    /**
     *
     * Send message to all connected nodes
     *
     * @param nulsData message
     * @param asyn     Whether or not asynchronous
     * @return BroadcastResult
     */
    BroadcastResult sendToAllNode(BaseNulsData nulsData, boolean asyn, int percent);

    /**
     *
     * Send message to all connected nodes
     *
     * @param event event
     * @param excludeNode node that does not need to be send
     * @param asyn        Whether or not asynchronous
     * @return BroadcastResult
     */
    BroadcastResult sendToAllNode(BaseNulsData event, Node excludeNode, boolean asyn, int
percent);

    /**
     * send message to node
     *
     * @param event event
     * @param node node
     * @param asyn  Whether or not asynchronous
     * @return BroadcastResult
     */
    BroadcastResult sendToNode(BaseNulsData event, Node node, boolean asyn);

    /**
     *
     * send message to nodeGroup
     *
     * @param  event event
     * @param groupName groupName
     * @param asyn asyn
     * @return BroadcastResult
     */
```

```
    BroadcastResult sendToGroup(BaseNulsData event, String groupName, boolean asyn);

    /**
     *
     * send message to nodeGroup
     *
     * @param event event
     * @param groupName groupName
     * @param excludeNode node that does not need to be send
     * @param asyn asyn
     * @return BroadcastResult
     */
    BroadcastResult sendToGroup(BaseNulsData event, String groupName, Node excludeNode,
boolean asyn);

    /**
     *
     * reset network module
     */
    void reset();

    /**
     *
     * Get network configuration information
     *
     * @return NetworkParam
     */
    NetworkParam getNetworkParam();
}

88:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\cache\DataCacher.java
 */

package io.nuls.protocol.base.cache;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.protocol.constant.MessageDataType;

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

```
import java.util.concurrent.CompletableFuture;

/**
 * @author: Niels Wang
 */
public class DataCacher<T> {

    private final MessageDataType type;

    public DataCacher(MessageDataType type) {
        this.type = type;
    }

    private Map<NulsDigestData, CompletableFuture<T>> cacher = new HashMap<>();

    public CompletableFuture<T> addFuture(NulsDigestData hash) {
        CompletableFuture future = new CompletableFuture<>();
//        Log.info(type + "future" + hash);
        cacher.put(hash, future);
        return future;
    }

    public boolean callback(NulsDigestData hash, T t) {
        return this.callback(hash, t, true);
    }

    public boolean callback(NulsDigestData hash, T t, boolean log) {
        CompletableFuture<T> future = cacher.get(hash);
        if (future == null) {
            if (log) {
                Log.warn("Time out: ({}) : {}", type, hash.getDigestHex());
            }
            return false;
        }
        future.complete(t);
//        Log.info(type + "future" + hash);
        cacher.remove(hash);
        return true;
    }

    public void notFound(NulsDigestData hash) {
        CompletableFuture<T> future = cacher.get(hash);
```

```
        if (future == null) {
            return;
        }
        future.complete(null);
//        Log.info(type + "not fountfuture" + hash);
        cacher.remove(hash);
    }

    public void removeFuture(NulsDigestData hash) {
//        Log.info(type + "future" + hash);
        cacher.remove(hash);
    }
}

89:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\cache\ProtocolCacheHandler.java
 */

package io.nuls.protocol.base.cache;

import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.constant.MessageDataType;
import io.nuls.protocol.model.BlockHashResponse;
import io.nuls.protocol.model.CompleteParam;
import io.nuls.protocol.model.NotFound;

import java.util.concurrent.CompletableFuture;
import java.util.concurrent.Future;

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


    private static DataCacher<Block> blockByHashCacher = new
DataCacher<>(MessageDataType.BLOCK);
    private static DataCacher<Block> blockByHeightCacher = new
DataCacher<>(MessageDataType.BLOCK);
//    private static DataCacher<TxGroup> txGroupCacher = new
```

```
DataCacher<>(MessageDataType.TRANSACTIONS);
    private static DataCacher<BlockHashResponse> blockHashesCacher = new
DataCacher<>(MessageDataType.HASHES);
    private static DataCacher<CompleteParam> taskCacher = new
DataCacher<>(MessageDataType.BLOCKS);
    private static DataCacher<NulsDigestData> reactCacher = new
DataCacher<>(MessageDataType.REQUEST);
//    private static DataCacher<Transaction> txCacher = new
DataCacher<>(MessageDataType.TRANSACTION);
//    private static DataCacher<SmallBlock> smallBlockCacher = new
DataCacher<>(MessageDataType.SMALL_BLOCK);

//    public static CompletableFuture<Transaction> addGetTxRequest(NulsDigestData txHash) {
//        return txCacher.addFuture(txHash);
//    }

    public static CompletableFuture<Block> addGetBlockByHeightRequest(NulsDigestData
requestHash) {
        return blockByHeightCacher.addFuture(requestHash);
    }

    public static CompletableFuture<Block> addGetBlockByHashRequest(NulsDigestData
requestHash) {
        return blockByHashCacher.addFuture(requestHash);
    }

    public static void receiveBlock(Block block) {
        NulsDigestData hash =
NulsDigestData.calcDigestData(SerializeUtils.uint64ToByteArray(block.getHeader().getHeight()));
        boolean result = blockByHeightCacher.callback(hash, block, false);
        if (!result) {
            blockByHashCacher.callback(block.getHeader().getHash(), block);
        }
    }

    public static CompletableFuture<BlockHashResponse>
addGetBlockHashesRequest(NulsDigestData requestHash) {
        return blockHashesCacher.addFuture(requestHash);
    }

    public static void receiveHashes(BlockHashResponse hashes) {
        blockHashesCacher.callback(hashes.getRequestMessageHash(), hashes);
```

```
    }

    public static Future<CompleteParam> addTaskRequest(NulsDigestData hash) {
        return taskCacher.addFuture(hash);
    }

    public static void notFound(NotFound data) {
        if (data.getType() == MessageDataType.BLOCK) {
            blockByHeightCacher.notFound(data.getHash());
            blockByHashCacher.notFound(data.getHash());
        } else if (data.getType() == MessageDataType.BLOCKS) {
            taskCacher.notFound(data.getHash());
        } else if (data.getType() == MessageDataType.HASHES) {
            blockHashesCacher.notFound(data.getHash());
        }
//        else if (data.getType() == MessageDataType.TRANSACTIONS) {
//            txGroupCacher.notFound(data.getHash());
//        } else if (data.getType() == MessageDataType.TRANSACTION) {
//            txCacher.notFound(data.getHash());
//        } else if (data.getType() == MessageDataType.SMALL_BLOCK) {
//            smallBlockCacher.notFound(data.getHash());
//        }
    }

    public static void taskComplete(CompleteParam param) {
        taskCacher.callback(param.getRequestHash(), param);
    }

    public static Future<NulsDigestData> addRequest(NulsDigestData requesetId) {
        return reactCacher.addFuture(requesetId);

    }

    public static void requestReact(NulsDigestData requesetId) {
        reactCacher.callback(requesetId, requesetId);
    }

    public static void removeBlockByHeightFuture(NulsDigestData hash) {
        blockByHeightCacher.removeFuture(hash);
    }

    public static void removeBlockByHashFuture(NulsDigestData hash) {
```

```
        blockByHashCacher.removeFuture(hash);
    }

    public static void removeHashesFuture(NulsDigestData hash) {
        blockHashesCacher.removeFuture(hash);
    }

    public static void removeTaskFuture(NulsDigestData hash) {
        taskCacher.removeFuture(hash);
    }

    public static void removeRequest(NulsDigestData requesetId) {
        reactCacher.removeFuture(requesetId);
    }

}

90:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\cache\TransactionDuplicateRemoval.java
 */

package io.nuls.protocol.base.cache;

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.protocol.base.utils.filter.InventoryFilter;

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

/**
 *
 *
 * @author: Niels Wang
 * @date: 2018/7/8
 */
public class TransactionDuplicateRemoval {

    private static InventoryFilter FILTER = new InventoryFilter( 1000000);

    public static boolean mightContain(NulsDigestData hash) {
        return FILTER.contains(hash.getDigestBytes());
```

```
    }

    public static void insert(NulsDigestData hash) {
        FILTER.insert(hash.getDigestBytes());
    }
}

91:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\constant\DownloadStatus.java
 */

package io.nuls.protocol.base.constant;

/**
 * Created by ln on 2018/4/8.
 */
public enum DownloadStatus {

    WAIT,

    READY,

    DOWNLOADING,

    SUCCESS,

    FAILED,

    STOPPED;

}

92:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\download\entity\DownloadRound.java
 */
package io.nuls.protocol.base.download.entity;

import java.util.List;

/**
 * @author Niels
 */
```

```
public class DownloadRound {

    private long start;
    private long end;
    private List<String> nodeIdList;

    public long getStart() {
        return start;
    }

    public void setStart(long start) {
        this.start = start;
    }

    public long getEnd() {
        return end;
    }

    public void setEnd(long end) {
        this.end = end;
    }

    public void setNodeIdList(List<String> nodeIdList) {
        this.nodeIdList = nodeIdList;
    }

    public List<String> getNodeIdList() {
        return nodeIdList;
    }
}

93:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\download\entity\NetworkNewestBlockInfos.java
 */

package io.nuls.protocol.base.download.entity;

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

import java.util.List;
```

```
/**
 * Created by ln on 2018/4/8.
 */
public class NetworkNewestBlockInfos {

    private long netBestHeight;
    private NulsDigestData netBestHash;
    private long localBestHeight;
    private String localBestHash;

    private List<Node> nodes;

    public NetworkNewestBlockInfos() {
    }

    public NetworkNewestBlockInfos(long netBestHeight, NulsDigestData netBestHash, List<Node>
nodes) {
        this.netBestHeight = netBestHeight;
        this.netBestHash = netBestHash;
        this.nodes = nodes;
    }

    public void setNetBestHeight(long netBestHeight) {
        this.netBestHeight = netBestHeight;
    }

    public void setNetBestHash(NulsDigestData netBestHash) {
        this.netBestHash = netBestHash;
    }

    public void setLocalBestHeight(long localBestHeight) {
        this.localBestHeight = localBestHeight;
    }

    public void setLocalBestHash(String localBestHash) {
        this.localBestHash = localBestHash;
    }

    public void setNodes(List<Node> nodes) {
        this.nodes = nodes;
    }
```

```
    public long getNetBestHeight() {
        return netBestHeight;
    }

    public NulsDigestData getNetBestHash() {
        return netBestHash;
    }

    public long getLocalBestHeight() {
        return localBestHeight;
    }

    public String getLocalBestHash() {
        return localBestHash;
    }

    public List<Node> getNodes() {
        return nodes;
    }

    @Override
    public String toString() {
        return "NetworkNewestBlockInfos{" +
                "netBestHeight=" + netBestHeight +
                ", netBestHash='" + netBestHash + '\'' +
                ", localBestHeight=" + localBestHeight +
                ", localBestHash='" + localBestHash + '\'' +
                ", nodes=" + nodes +
                '}';
    }
}

94:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\download\entity\NodeDownloadingStatus.java
 */
package io.nuls.protocol.base.download.entity;

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

/**
 * @author Niels
```

```
 */
public class NodeDownloadingStatus {

    private String nodeId;
    private Set<Long> downloadingSet = new HashSet<>();
    private Set<Long> downloadedSet = new HashSet<>();
    private long updateTime;

    public String getNodeId() {
        return nodeId;
    }

    public void setNodeId(String nodeId) {
        this.nodeId = nodeId;
    }


    public long getUpdateTime() {
        return updateTime;
    }

    public void setUpdateTime(long updateTime) {
        this.updateTime = updateTime;
    }

    public boolean containsHeight(long height) {
        return downloadingSet.contains(height);
    }

    public synchronized void downloaded(long height) {
        downloadedSet.add(height);
    }

    public boolean finished() {
        return downloadedSet.size() == downloadingSet.size();
    }

    public void setDownloadingSet(long start, long end) {
        for (long i = start; i <= end; i++) {
            downloadingSet.add(i);
        }
    }
```

```
}

95:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\download\entity\ResultMessage.java
 */

package io.nuls.protocol.base.download.entity;

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

import java.util.List;

/**
 * Created by ln on 2018/4/8.
 */
public class ResultMessage {
    private NulsDigestData startHash;
    private NulsDigestData endHash;
    private long startHeight;
    private int size;
    private Node node;

    private List<Block> blockList;

    public ResultMessage(NulsDigestData startHash, NulsDigestData endHash, long startHeight,
int size, Node node, List<Block> blockList) {
        this.startHash = startHash;
        this.endHash = endHash;
        this.startHeight = startHeight;
        this.size = size;
        this.node = node;
        this.blockList = blockList;
    }

    public NulsDigestData getStartHash() {
        return startHash;
    }

    public NulsDigestData getEndHash() {
        return endHash;
```

```
    }

    public long getStartHeight() {
        return startHeight;
    }

    public int getSize() {
        return size;
    }

    public Node getNode() {
        return node;
    }

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

    public void setStartHash(NulsDigestData startHash) {
        this.startHash = startHash;
    }

    public void setEndHash(NulsDigestData endHash) {
        this.endHash = endHash;
    }

    public void setStartHeight(long startHeight) {
        this.startHeight = startHeight;
    }

    public void setSize(int size) {
        this.size = size;
    }

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

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

96:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\download\processor\DownloadProcessor.java
 */

```
package io.nuls.protocol.base.download.processor;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.constant.NetworkErrorCode;
import io.nuls.network.model.Node;
import io.nuls.network.service.NetworkService;
import io.nuls.protocol.base.constant.DownloadStatus;
import io.nuls.protocol.base.download.entity.NetworkNewestBlockInfos;
import io.nuls.protocol.base.download.thread.DownloadThreadManager;
import io.nuls.protocol.base.download.utils.DownloadDataStorage;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.service.BlockService;

import java.util.*;
import java.util.concurrent.*;

/**
 * @author ln
 */
public class DownloadProcessor extends Thread {

    private static DownloadProcessor INSTANCE = new DownloadProcessor();

    private static long FAIL_RETRY_TIME = 30 * 1000L;
    private long failedTime;

    // pierre test comment out
    private DownloadStatus downloadStatus = DownloadStatus.WAIT;
```

```
    private ScheduledThreadPoolExecutor threadPool;

    private NetworkService networkService;
    private BlockService blockService;

    private DownloadProcessor() {
    }

    public static DownloadProcessor getInstance() {
        return INSTANCE;
    }

    @Override
    public void run() {
        networkService = NulsContext.getServiceBean(NetworkService.class);
        blockService = NulsContext.getServiceBean(BlockService.class);
        boolean isContinue = checkNetworkAndStatus();

        if (!isContinue) {
            return;
        }

        //
        //There is no change in the inspection network. Usually, the number of nodes that trigger
synchronization is limited.
        // To ensure the accuracy and security of data, try to wait for more nodes to start
synchronization.
//        waitNetworkNotChange();
        downloadStatus = DownloadStatus.READY;
        try {
            doSynchronize();
        } catch (NulsRuntimeException e) {
            Log.warn(e.getMessage());
        }
    }

    /**
     *
     * block synchronization process
     */
    private void doSynchronize() {
```

```
        if (downloadStatus != DownloadStatus.READY) {
            return;
        }

        downloadStatus = DownloadStatus.DOWNLOADING;

        //hash
        //Finding the highest block hash consistent with most nodes in the network
        NetworkNewestBlockInfos newestInfos = getNetworkNewestBlockInfos();

        if (newestInfos.getNodes().size() < ProtocolConstant.ALIVE_MIN_NODE_COUNT) {
            downloadStatus = DownloadStatus.WAIT;
            return;
        }
        NulsContext.getInstance().setNetBestBlockHeight(newestInfos.getNetBestHeight());
        BlockingQueue<Block> blockQueue = new LinkedBlockingQueue<>();

        DownloadThreadManager downloadThreadManager = new
DownloadThreadManager(newestInfos, blockQueue);

        FutureTask<Boolean> threadManagerFuture = new
FutureTask<>(downloadThreadManager);

        TaskManager.createAndRunThread(ProtocolConstant.MODULE_ID_PROTOCOL,
"download-thread-manager",
                new Thread(threadManagerFuture));

        DownloadDataStorage downloadDataStorage = new DownloadDataStorage(blockQueue);

        FutureTask<Boolean> dataStorageFuture = new FutureTask<>(downloadDataStorage);

        TaskManager.createAndRunThread(ProtocolConstant.MODULE_ID_PROTOCOL,
"download-data-storeage",
                new Thread(dataStorageFuture));

        try {
            Boolean downResult = threadManagerFuture.get();

            blockQueue.offer(new Block());

            Boolean storageResult = dataStorageFuture.get();
```

```
            boolean success = downResult != null && downResult.booleanValue() && storageResult != null && storageResult.booleanValue();

            if (success && checkIsNewest(newestInfos)) {
                downloadStatus = DownloadStatus.SUCCESS;
            } else if (downloadStatus != DownloadStatus.WAIT) {
                downloadStatus = DownloadStatus.FAILED;
            }
        } catch (Exception e) {
            Log.error(e);
            downloadStatus = DownloadStatus.FAILED;
        }
    }

    private boolean checkIsNewest(NetworkNewestBlockInfos downloadInfos) {

        long downloadBestHeight = downloadInfos.getNetBestHeight();
        long time = TimeService.currentTimeMillis();
        long timeout = 60 * 1000L;
        long localBestHeight = 0L;

        while (true) {
            if (TimeService.currentTimeMillis() - time > timeout) {
                break;
            }

            long bestHeight = blockService.getBestBlock().getData().getHeader().getHeight();
            if (bestHeight >= downloadBestHeight) {
                break;
            } else if (bestHeight != localBestHeight) {
                localBestHeight = bestHeight;
                time = TimeService.currentTimeMillis();
            }
            try {
                Thread.sleep(100L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        NetworkNewestBlockInfos newestInfos = getNetworkNewestBlockInfos();
        if (newestInfos.getNetBestHeight() >
```

```
blockService.getBestBlock().getData().getHeader().getHeight()) {
            downloadStatus = DownloadStatus.WAIT;
            return false;
        }
        return true;
    }

    /**
     *
     * Get latest block information of peer nodes in the network
     *
     * @return NetworkNewestBlockInfos
     */
    private NetworkNewestBlockInfos getNetworkNewestBlockInfos() {

        NetworkNewestBlockInfos infos = getNetworkNewestBlock();

        return infos;
    }

    public NetworkNewestBlockInfos getNetworkNewestBlock() {

        Collection<Node> nodeList = networkService.getAvailableNodes();

        Map<NulsDigestData, Integer> statisticsMaps = new HashMap<>();
        Map<NulsDigestData, List<Node>> nodeMaps = new HashMap<>();

//        System.out.println("--------------start download--------------------");
        for (Node node : nodeList) {
//            System.out.println(node.getId() + " : " + node.getBestBlockHeight() + " : " +
node.getBestBlockHash());
            NulsDigestData hash = node.getBestBlockHash();

            Integer statistics = statisticsMaps.get(hash);

            if (statistics == null) {
                statisticsMaps.put(hash, 0);
            }

            statisticsMaps.put(hash, statisticsMaps.get(hash) + 1);

            List<Node> nodes = nodeMaps.get(hash);
```

```
            if (nodes == null) {
                nodes = new ArrayList<>();
                nodeMaps.put(hash, nodes);
            }
            nodes.add(node);
        }

        //max number
        int max = 0;
        long bestHeight = 0;
        NulsDigestData bestHash = null;
        List<Node> nodes = null;

        for (Map.Entry<NulsDigestData, Integer> entry : statisticsMaps.entrySet()) {
            int count = entry.getValue();
            NulsDigestData hash = entry.getKey();

            List<Node> tempNodes = nodeMaps.get(hash);
            long height = tempNodes.get(0).getBestBlockHeight();

            if (count > max || (count == max && bestHeight < height)) {
                max = count;
                bestHash = hash;
                bestHeight = height;
                nodes = tempNodes;
            }
        }

        if (nodes == null || nodes.size() == 0) {
            throw new NulsRuntimeException(NetworkErrorCode.NET_NODE_NOT_FOUND);
        }
        return new NetworkNewestBlockInfos(bestHeight, bestHash, nodes);
    }

    private void waitNetworkNotChange() throws NulsRuntimeException {
        //10
        //Wait for no change in the node within 10 seconds (usually growth), then start
synchronization

        int nodeSize = networkService.getAvailableNodes().size();

        long now = TimeService.currentTimeMillis();
```

```
        long timeout = 10000L;
        while (true) {
            int newNodeSize = networkService.getAvailableNodes().size();
            if (newNodeSize > nodeSize) {
                now = TimeService.currentTimeMillis();
                nodeSize = newNodeSize;
            }

            if (TimeService.currentTimeMillis() - now >= timeout) {
                break;
            }
            try {
                Thread.sleep(500L);
            } catch (InterruptedException e) {
                Log.error(e);
            }
        }

        //check node size again
        nodeSize = networkService.getAvailableNodes().size();
        if (nodeSize < ProtocolConstant.ALIVE_MIN_NODE_COUNT) {
            downloadStatus = DownloadStatus.WAIT;
            return;
        }
        downloadStatus = DownloadStatus.READY;
    }

    private boolean checkNetworkAndStatus() {
        //
        //Monitor the network, if it is dropped, reconnect after reconnecting
        Collection<Node> nodes = networkService.getAvailableNodes();
        if (nodes == null || nodes.size() == 0 || nodes.size() <
ProtocolConstant.ALIVE_MIN_NODE_COUNT) {
            return false;
        }

        //if failed , retry
        if (downloadStatus == DownloadStatus.FAILED && TimeService.currentTimeMillis() >
failedTime + FAIL_RETRY_TIME) {
            downloadStatus = DownloadStatus.WAIT;
            return false;
        }
```

```
        if (downloadStatus != DownloadStatus.WAIT) {
            return false;
        }
        return true;
    }

    public boolean startup() {
        threadPool = TaskManager.createScheduledThreadPool(1,
                new NulsThreadFactory(ProtocolConstant.MODULE_ID_PROTOCOL, "data-
synchronize"));
        threadPool.scheduleAtFixedRate(INSTANCE, 0, 1, TimeUnit.SECONDS);
        return true;
    }

    public boolean shutdown() {
        threadPool.shutdownNow();
        downloadStatus = DownloadStatus.STOPPED;
        return true;
    }

    public void setDownloadStatus(DownloadStatus downloadStatus) {
        this.downloadStatus = downloadStatus;
    }

    public DownloadStatus getDownloadStatus() {
        return downloadStatus;
    }
}

97:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\download\thread\DownloadThread.java
 */

package io.nuls.protocol.base.download.thread;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.download.entity.ResultMessage;
import io.nuls.protocol.base.download.utils.DownloadUtils;
```

```
import java.util.List;
import java.util.concurrent.Callable;

/**
 * Created by ln on 2018/4/8.
 */
public class DownloadThread implements Callable<ResultMessage> {

    private NulsDigestData startHash;
    private NulsDigestData endHash;
    private long startHeight;
    private int size;
    private Node node;

    public DownloadThread(NulsDigestData startHash, NulsDigestData endHash, long startHeight,
int size, Node node) {
        this.startHash = startHash;
        this.endHash = endHash;
        this.startHeight = startHeight;
        this.size = size;
        this.node = node;
    }

    @Override
    public ResultMessage call() throws Exception {
        List<Block> blockList = null;
        try {
//            Log.info("download thread : " + Thread.currentThread().getName() + " ,  startHeight : " +
startHeight + ", size : " + size + " , from node : " + node.getId() + " , startHash : " + startHash + " ,
endHash : " + endHash);
            blockList = DownloadUtils.getBlocks(node, startHeight, startHeight + size - 1);
//            Log.info("download complete thread : " + Thread.currentThread().getName() + " ,
startHeight : " + startHeight + ", size : " + size + " , from node : " + node.getId() + " , get data size :
" + (blockList == null ? 0 : blockList.size()));
        } catch (Exception e) {
            Log.error(e.getMessage());
        }
        return new ResultMessage(startHash, endHash, startHeight, size, node, blockList);
    }
}
```

98:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-

base\src\main\java\io\nuls\protocol\base\download\thread\DownloadThreadManager.java
 */

package io.nuls.protocol.base.download.thread;

import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.calc.DoubleUtils;
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.NulsDigestData;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.network.model.Node;
import io.nuls.network.service.NetworkService;
import io.nuls.protocol.base.download.entity.NetworkNewestBlockInfos;
import io.nuls.protocol.base.download.entity.ResultMessage;
import io.nuls.protocol.base.download.utils.DownloadUtils;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.service.BlockService;

import java.util.ArrayList;
import java.util.List;
import java.util.Queue;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;
import java.util.concurrent.ThreadPoolExecutor;

/**
 * @author ln
 */
public class DownloadThreadManager implements Callable<Boolean> {

    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);
    private NetworkService networkService = NulsContext.getServiceBean(NetworkService.class);
    private ConsensusService consensusService =
NulsContext.getServiceBean(ConsensusService.class);

```
    private NulsThreadFactory factory = new
NulsThreadFactory(ProtocolConstant.MODULE_ID_PROTOCOL, "download");

    private NetworkNewestBlockInfos newestInfos;
    private Queue<Block> blockQueue;
    private String queueName;

    private int maxDowncount = 10;

    public DownloadThreadManager(NetworkNewestBlockInfos newestInfos, Queue<Block>
blockQueue) {
        this.newestInfos = newestInfos;
        this.blockQueue = blockQueue;
        this.queueName = queueName;
    }

    @Override
    public Boolean call() throws Exception {
        try {
            boolean isContinue = checkFirstBlock();

            if (!isContinue) {
                return true;
            }
        } catch (NulsRuntimeException e) {
            return false;
        }

        List<Node> nodes = newestInfos.getNodes();
        NulsDigestData netBestHash = newestInfos.getNetBestHash();
        long netBestHeight = newestInfos.getNetBestHeight();
        Block localBestBlock = blockService.getBestBlock().getData();
        NulsDigestData localBestHash = localBestBlock.getHeader().getHash();
        long localBestHeight = localBestBlock.getHeader().getHeight();

        ThreadPoolExecutor executor = TaskManager.createThreadPool(nodes.size(), 0,
                new NulsThreadFactory(ProtocolConstant.MODULE_ID_PROTOCOL, "download-
thread"));

        List<FutureTask<ResultMessage>> futures = new ArrayList<>();

        long totalCount = netBestHeight - localBestHeight;
```

```
        long laveCount = totalCount;

        long downCount = (long) Math.ceil((double) totalCount / (maxDowncount * nodes.size()));

        for (long i = 0; i < downCount; i++) {

            long startHeight = (localBestHeight + 1) + i * maxDowncount * nodes.size();

            for (int j = nodes.size() - 1; j >= 0; j--) {
                Node node = nodes.get(j);
                if (!node.isHandShake()) {
                    nodes.remove(j);
                }
            }
            for (int j = 0; j < nodes.size(); j++) {
                long start = startHeight + j * maxDowncount;
                int size = maxDowncount;

                boolean isEnd = false;
                if (start + size >= netBestHeight) {
                    size = (int) (netBestHeight - start) + 1;
                    isEnd = true;
                }

                DownloadThread downloadThread = new DownloadThread(localBestHash,
netBestHash, start, size, nodes.get(j));

                FutureTask<ResultMessage> downloadThreadFuture = new
FutureTask<>(downloadThread);

                executor.execute(factory.newThread(downloadThreadFuture));

                futures.add(downloadThreadFuture);

                if (isEnd) {
                    break;
                }
            }
            for (FutureTask<ResultMessage> task : futures) {
                ResultMessage result = null;
                try {
```

```
                result = task.get();
            } catch (Exception e) {
                Log.error(e);
            }
            List<Block> blockList = null;

            if (result == null || (blockList = result.getBlockList()) == null || blockList.size() == 0) {
                blockList = retryDownload(executor, result);
            }

            if (blockList == null) {
                executor.shutdown();
                resetNetwork("attempts to download blocks from all available nodes failed");
                return true;
            }

            for (Block block : blockList) {
                blockQueue.offer(block);
            }
        }
        futures.clear();
    }

    executor.shutdown();

    return true;
}

private List<Block> retryDownload(ThreadPoolExecutor executor, ResultMessage result) throws
InterruptedException, ExecutionException {

    //try download to other nodes
    List<Node> otherNodes = new ArrayList<>();

    Node defultNode = result.getNode();

    for (Node node : newestInfos.getNodes()) {
        if (!node.getId().equals(defultNode.getId())) {
            otherNodes.add(node);
        }
    }
```

```
        for (Node node : otherNodes) {
            result.setNode(node);
            List<Block> blockList = downloadBlockFromNode(executor, result);
            if (blockList != null && blockList.size() > 0) {
                return blockList;
            }
        }

        //if fail , down again
        result.setNode(defultNode);

        return downloadBlockFromNode(executor, result);
    }

    private List<Block> downloadBlockFromNode(ThreadPoolExecutor executor, ResultMessage result) throws ExecutionException, InterruptedException {
        DownloadThread downloadThread = new DownloadThread(result.getStartHash(), result.getEndHash(), result.getStartHeight(), result.getSize(), result.getNode());

        FutureTask<ResultMessage> downloadThreadFuture = new FutureTask<ResultMessage>(downloadThread);
        executor.execute(new Thread(downloadThreadFuture));

        List<Block> blockList = null;
        try {
            blockList = downloadThreadFuture.get().getBlockList();
        } catch (Exception e) {
            Log.error(e);
        }
        return blockList;
    }

    private boolean checkFirstBlock() throws NulsException {

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

        if (localBestBlock.getHeader().getHeight() == 0 || (newestInfos.getNetBestHeight() == localBestBlock.getHeader().getHeight() &&
                newestInfos.getNetBestHash().equals(localBestBlock.getHeader().getHash()))) {
            return true;
        }
```

```
        if (newestInfos.getNetBestHeight() < localBestBlock.getHeader().getHeight()) {
            BlockHeader header =
blockService.getBlockHeader(newestInfos.getNetBestHash()).getData();

            if (null == header && networkService.getAvailableNodes().size() >=
networkService.getNetworkParam().getMaxOutCount() &&
DoubleUtils.div(newestInfos.getNodes().size(), networkService.getAvailableNodes().size(), 2) >=
0.8d) {
                for (long i = localBestBlock.getHeader().getHeight(); i <=
newestInfos.getNetBestHeight(); i--) {
                    consensusService.rollbackBlock(localBestBlock);
                    localBestBlock = blockService.getBestBlock().getData();
                }
            } else if (null == header) {
                resetNetwork("The local block is higher than the network block, the number of
connected nodes is not enough to allow the local rollbackTx, so reset");
                return false;
            }
        } else {
            //check need rollbackTx
            checkRollback(localBestBlock, 0);
        }
        return true;
    }

    private void checkRollback(Block localBestBlock, int rollbackCount) throws NulsException {

        if (rollbackCount >= 10) {
//            resetNetwork("number of rollbackTx blocks greater than 10 during download");
            return;
        }

        List<Node> nodes = newestInfos.getNodes();

        for (Node node : nodes) {
            Block remoteBlock =
DownloadUtils.getBlockByHash(localBestBlock.getHeader().getHash(), node);
            if (remoteBlock != null && remoteBlock.getHeader().getHeight() ==
localBestBlock.getHeader().getHeight()) {
                return;
            }
        }
```

```
        if (newestInfos.getNodes().size() > 0) {
            consensusService.rollbackBlock(localBestBlock);
        } else {
//            resetNetwork("the number of available nodes is insufficient for rollbackTx blocks");
            return;
        }

        localBestBlock = blockService.getBestBlock().getData();

        checkRollback(localBestBlock, rollbackCount + 1);
    }

    private void resetNetwork(String reason) {
        NulsContext.getServiceBean(NetworkService.class).reset();
        throw new NulsRuntimeException(KernelErrorCode.FAILED);
    }

}

99:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\download\utils\DownloadDataStorage.java
 */

package io.nuls.protocol.base.download.utils;

import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Block;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.Callable;

/**
 * Created by ln on 2018/4/8.
 */
public class DownloadDataStorage implements Callable<Boolean> {

    private BlockingQueue<Block> blockQueue;
    private String queueName;
    private boolean running = true;
```

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

    public DownloadDataStorage(BlockingQueue<Block> blockQueue) {
        this.blockQueue = blockQueue;
        this.queueName = queueName;
    }

    @Override
    public Boolean call() throws Exception {
        try {
            Block block;
            while ((block = blockQueue.take()) != null) {
                if (block.getHeader() == null) {
                    break;
                }
                consensusService.addBlock(block);
            }
            return true;
        } catch (InterruptedException e) {
            Log.error(e);
            return false;
        }
    }

}

100:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\download\utils\DownloadUtils.java
 */

package io.nuls.protocol.base.download.utils;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
```

```
import io.nuls.protocol.base.cache.ProtocolCacheHandler;
import io.nuls.protocol.message.*;
import io.nuls.protocol.model.BlockHashResponse;
import io.nuls.protocol.model.CompleteParam;
import io.nuls.protocol.model.GetTxGroupParam;
import io.nuls.protocol.model.TxGroup;

import java.io.IOException;
import java.util.*;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

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

    private static MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);

    public static Block getBlockByHash(NulsDigestData hash, Node node) {
        if (hash == null || node == null) {
            return null;
        }
        GetBlockMessage message = new GetBlockMessage(hash);
        Future<Block> future = ProtocolCacheHandler.addGetBlockByHashRequest(hash);
        Future<NulsDigestData> reactFuture = ProtocolCacheHandler.addRequest(hash);
        Result result = messageBusService.sendToNode(message, node, false);
//        Log.error("start request:"+new Date().toLocaleString()+" ::: "+hash);
        if (!result.isSuccess()) {
            ProtocolCacheHandler.removeBlockByHashFuture(hash);
            ProtocolCacheHandler.removeRequest(hash);
            return null;
        }
        try {
            reactFuture.get(1L, TimeUnit.SECONDS);
            Block block = future.get(30L, TimeUnit.SECONDS);
            return block;
        } catch (Exception e) {
            Log.error(node.getId(), e);
            return null;
        } finally {
```

```
            ProtocolCacheHandler.removeBlockByHashFuture(hash);
            ProtocolCacheHandler.removeRequest(hash);
        }
    }

    public static List<Block> getBlocks(Node node, long startHeight, long endHeight) throws
Exception {

        Log.info("getBlocks:" + startHeight + "->" + endHeight + " ,from:" + node.getId());
        List<Block> resultList = new ArrayList<>();

        if (node == null || startHeight < 0L || startHeight > endHeight) {
            return resultList;
        }

//        Log.info("download block " + startHeight + " , " + endHeight + " from : " + node.getId());

        GetBlocksByHeightMessage message = new GetBlocksByHeightMessage(startHeight,
endHeight);

        NulsDigestData requestHash = null;
        try {
            requestHash = NulsDigestData.calcDigestData(message.getMsgBody().serialize());
        } catch (Exception e) {
            Log.error(e);
        }

        Future<CompleteParam> taskFuture =
ProtocolCacheHandler.addTaskRequest(requestHash);
        Future<NulsDigestData> reactFuture = ProtocolCacheHandler.addRequest(requestHash);

        List<Map<NulsDigestData, Future<Block>>> blockFutures = new ArrayList<>();
        for (long i = startHeight; i <= endHeight; i++) {
            NulsDigestData hash =
NulsDigestData.calcDigestData(SerializeUtils.uint64ToByteArray(i));
            Future<Block> blockFuture = ProtocolCacheHandler.addGetBlockByHeightRequest(hash);

            Map<NulsDigestData, Future<Block>> blockFutureMap = new HashMap<>();
            blockFutureMap.put(hash, blockFuture);
            blockFutures.add(blockFutureMap);
        }
```

```
    Result result = messageBusService.sendToNode(message, node, false);
    Log.info("sended.....");
    if (!result.isSuccess()) {
        ProtocolCacheHandler.removeTaskFuture(message.getHash());
        ProtocolCacheHandler.removeRequest(requestHash);

        for (Map<NulsDigestData, Future<Block>> blockFutureMap : blockFutures) {
            for (Map.Entry<NulsDigestData, Future<Block>> entry : blockFutureMap.entrySet()) {
                ProtocolCacheHandler.removeBlockByHeightFuture(entry.getKey());
            }
        }
        return resultList;
    }

    try {
        reactFuture.get(1L, TimeUnit.SECONDS);
        CompleteParam taskResult = taskFuture.get(60L, TimeUnit.SECONDS);
        if (taskResult.isSuccess()) {
            for (Map<NulsDigestData, Future<Block>> blockFutureMap : blockFutures) {
                for (Map.Entry<NulsDigestData, Future<Block>> entry : blockFutureMap.entrySet()) {
                    Block block = entry.getValue().get(30L, TimeUnit.SECONDS);
                    resultList.add(block);
                }
            }
        }
    } catch (Exception e) {
        Log.error(node.getId() + ",start:" + startHeight + " , endHeight:" + endHeight);
        Log.error(e.getMessage());
        return new ArrayList<>();
    } finally {
        ProtocolCacheHandler.removeTaskFuture(requestHash);
        ProtocolCacheHandler.removeRequest(requestHash);

        for (Map<NulsDigestData, Future<Block>> blockFutureMap : blockFutures) {
            for (Map.Entry<NulsDigestData, Future<Block>> entry : blockFutureMap.entrySet()) {
                ProtocolCacheHandler.removeBlockByHeightFuture(entry.getKey());
            }
        }
    }
    return resultList;
}
```

```
    public static List<NulsDigestData> getBlocksHash(Node node, long startHeight, long
endHeight) {

        if (node == null || startHeight < 0L || endHeight < 0L || startHeight > endHeight) {
            return new ArrayList<>();
        }

        if (endHeight - startHeight >= 10000) {
            Log.warn("get block hash more the 10000");
            return new ArrayList<>();
        }

        GetBlocksHashMessage message = new GetBlocksHashMessage(startHeight, endHeight);
        NulsDigestData requestHash = null;
        try {
            requestHash = NulsDigestData.calcDigestData(message.getMsgBody().serialize());
        } catch (IOException e) {
            e.printStackTrace();
        }
        Future<BlockHashResponse> future =
ProtocolCacheHandler.addGetBlockHashesRequest(requestHash);
        Result hashesResult = messageBusService.sendToNode(message, node, false);
        if (!hashesResult.isSuccess()) {
            ProtocolCacheHandler.removeHashesFuture(requestHash);
            return new ArrayList<>();
        }

        long size = (endHeight - startHeight + 1);

        BlockHashResponse response = null;
        try {
            response = future.get(20L, TimeUnit.SECONDS);
        } catch (Exception e) {
            Log.error(node.getId() + ",start:" + startHeight + " , size:" + size);
            Log.error(e);
        } finally {
            ProtocolCacheHandler.removeHashesFuture(requestHash);
        }

        if (null == response || response.getHashList() == null || response.getHashList().size() != size)
{
            Log.warn("get blocks hashList({}-{}) failed:" + node.getId(), startHeight, size);
```

```
            return new ArrayList<>();
        }
        return response.getHashList();
    }
}
```

101:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\handler\BlockMessageHandler.java

```
 */
package io.nuls.protocol.base.handler;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.model.Block;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.cache.ProtocolCacheHandler;
import io.nuls.protocol.message.BlockMessage;

/**
 * @author facjas
 */
public class BlockMessageHandler extends AbstractMessageHandler<BlockMessage> {

    @Override
    public void onMessage(BlockMessage event, Node fromNode) {
        Block block = event.getMsgBody();
        if (null == block) {
            Log.warn("recieved a null blockEvent form " + fromNode.getId());
            return;
        }
        ProtocolCacheHandler.receiveBlock(block);
    }
}
```

102:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\handler\BlocksHashHandler.java

```
 */
package io.nuls.protocol.base.handler;

import io.nuls.kernel.exception.NulsException;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
```

```
import io.nuls.protocol.base.cache.ProtocolCacheHandler;
import io.nuls.protocol.message.BlocksHashMessage;

/**
 * @author Niels
 */
public class BlocksHashHandler extends AbstractMessageHandler<BlocksHashMessage> {

    @Override
    public void onMessage(BlocksHashMessage message, Node fromNode) throws NulsException
{
        ProtocolCacheHandler.receiveHashes(message.getMsgBody());
    }

}

103:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\CompleteHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.cache.ProtocolCacheHandler;
import io.nuls.protocol.message.CompleteMessage;
import io.nuls.protocol.model.CompleteParam;

/**
 * @author ln
 */
public class CompleteHandler extends AbstractMessageHandler<CompleteMessage> {

    @Override
    public void onMessage(CompleteMessage message, Node fromNode) {

        if(message == null || message.getMsgBody() == null || fromNode == null) {
            return;
        }

        CompleteParam param = message.getMsgBody();

        ProtocolCacheHandler.taskComplete(param);
```

```
    }
}

104:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\ForwardSmallBlockHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.utils.SmallBlockDuplicateRemoval;
import io.nuls.protocol.message.ForwardSmallBlockMessage;
import io.nuls.protocol.message.GetSmallBlockMessage;

/**
 * @author facjas
 */
public class ForwardSmallBlockHandler extends
AbstractMessageHandler<ForwardSmallBlockMessage> {

    private MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);

    @Override
    public void onMessage(ForwardSmallBlockMessage message, Node fromNode) {
        if (message == null || fromNode == null || !fromNode.isHandShake() || null ==
message.getMsgBody()) {
            return;
        }
        NulsDigestData hash = message.getMsgBody();
        if (!SmallBlockDuplicateRemoval.needDownloadSmallBlock(hash)) {
            return;
        }
        GetSmallBlockMessage getSmallBlockMessage = new GetSmallBlockMessage();
        getSmallBlockMessage.setMsgBody(hash);
        Result result = messageBusService.sendToNode(getSmallBlockMessage, fromNode, true);
        if (result.isFailed()) {
            SmallBlockDuplicateRemoval.removeForward(hash);
```

```
            return;
        }
    }

}

105:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\ForwardTxMessageHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.cache.TransactionDuplicateRemoval;
import io.nuls.protocol.cache.TemporaryCacheManager;
import io.nuls.protocol.message.ForwardTxMessage;
import io.nuls.protocol.message.GetTxMessage;

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

/**
 * @author facjas
 */
public class ForwardTxMessageHandler extends AbstractMessageHandler<ForwardTxMessage>
{

    @Override
    public void onMessage(ForwardTxMessage message, Node fromNode) {
        if (message == null || fromNode == null || !fromNode.isHandShake() || null ==
message.getMsgBody()) {
            return;
        }
        NulsDigestData hash = message.getMsgBody();
        boolean consains = TransactionDuplicateRemoval.mightContain(hash);
        if (consains) {
            return;
        }
        TransactionDuplicateRemoval.insert(hash);
        GetTxMessage getTxMessage = new GetTxMessage();
```

```
        getTxMessage.setMsgBody(hash);
        Result result = messageBusService.sendToNode(getTxMessage, fromNode, true);
        if (result.isFailed()) {
            return;
        }
    }

}
```

106:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\handler\GetBlockHandler.java

```
 */
package io.nuls.protocol.base.handler;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.constant.MessageDataType;
import io.nuls.protocol.message.BlockMessage;
import io.nuls.protocol.message.GetBlockMessage;
import io.nuls.protocol.message.NotFoundMessage;
import io.nuls.protocol.message.ReactMessage;
import io.nuls.protocol.model.NotFound;
import io.nuls.protocol.service.BlockService;

/**
 * @author facjas
 */
public class GetBlockHandler extends AbstractMessageHandler<GetBlockMessage> {

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

    @Override
    public void onMessage(GetBlockMessage message, Node fromNode) {
```

```
        if(message == null || message.getMsgBody() == null || fromNode == null) {
            return;
        }

        NulsDigestData blockHash = message.getBlockHash();

        // react request
        messageBusService.sendToNode(new ReactMessage(blockHash), fromNode, true);

        Block block= null;
        Result<Block> result = blockService.getBlock(blockHash);
        if (result.isFailed() || (block = result.getData()) == null) {
            sendNotFound(blockHash, fromNode);
            return;
        }
        sendBlock(block, fromNode);
    }

    private void sendNotFound(NulsDigestData hash, Node node) {
        NotFoundMessage message = new NotFoundMessage();
        NotFound data = new NotFound(MessageDataType.BLOCK, hash);
        message.setMsgBody(data);
        Result result = this.messageBusService.sendToNode(message, node, true);
        if (result.isFailed()) {
            Log.warn("send BLOCK NotFound failed:" + node.getId() + ", hash:" + hash);
        }
    }

    private void sendBlock(Block block, Node fromNode) {
        BlockMessage blockMessage = new BlockMessage();
        blockMessage.setMsgBody(block);
        Result result = this.messageBusService.sendToNode(blockMessage, fromNode, true);
        if (result.isFailed()) {
            Log.warn("send block failed:" + fromNode.getId() + ",height:" +
block.getHeader().getHeight());
        }
    }
}

107:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\GetBlocksByHashHandler.java
 */
```

```
package io.nuls.protocol.base.handler;

import io.nuls.core.tools.log.Log;
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.model.Result;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.constant.MessageDataType;
import io.nuls.protocol.message.*;
import io.nuls.protocol.model.CompleteParam;
import io.nuls.protocol.model.GetBlocksByHashParam;
import io.nuls.protocol.model.NotFound;
import io.nuls.protocol.service.BlockService;

import java.io.IOException;

/**
 * @author facjas
 */
public class GetBlocksByHashHandler extends
AbstractMessageHandler<GetBlocksByHashMessage> {

    private static final int MAX_SIZE = 1000;
    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);
    private MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);

    @Override
    public void onMessage(GetBlocksByHashMessage message, Node fromNode) {

        if(message == null || message.getMsgBody() == null || fromNode == null) {
            return;
        }

        GetBlocksByHashParam param = message.getMsgBody();
        if(param.getStartHash() == null || param.getEndHash() == null) {
            return;
        }
```

```
        NulsDigestData requestHash = null;
        try {
            requestHash = NulsDigestData.calcDigestData(param.serialize());
        } catch (IOException e) {
            e.printStackTrace();
        }

        // react request
        messageBusService.sendToNode(new ReactMessage(requestHash), fromNode, true);

        BlockHeader startBlockHeader =
blockService.getBlockHeader(param.getStartHash()).getData();
        if(startBlockHeader == null) {
            sendNotFound(requestHash, fromNode);
            return;
        }
        Block endBlock = blockService.getBlock(param.getEndHash()).getData();
        if(endBlock == null) {
            sendNotFound(requestHash, fromNode);
            return;
        }
        if(endBlock.getHeader().getHeight() - startBlockHeader.getHeight() >= MAX_SIZE) {
            return;
        }

        Block block = endBlock;
        while(true) {
            sendBlock(block, fromNode);
            if(block.getHeader().getHash().equals(startBlockHeader.getHash())) {
                break;
            }
            Result<Block> result = blockService.getBlock(block.getHeader().getPreHash());
            if (result.isFailed() || (block = result.getData()) == null) {
                sendNotFound(requestHash, fromNode);
                return;
            }
        }

        CompleteMessage completeMessage = new CompleteMessage();
        completeMessage.setMsgBody(new CompleteParam(requestHash, true));
        messageBusService.sendToNode(completeMessage, fromNode, true);
```

```
    }

    private void sendNotFound(NulsDigestData hash, Node node) {
        NotFoundMessage message = new NotFoundMessage();
        NotFound data = new NotFound(MessageDataType.BLOCKS, hash);
        message.setMsgBody(data);
        Result result = this.messageBusService.sendToNode(message, node, true);
        if (result.isFailed()) {
            Log.warn("send BLOCK NotFound failed:" + node.getId() + ", hash:" + hash);
        }
    }

    private void sendBlock(Block block, Node fromNode) {
        BlockMessage blockMessage = new BlockMessage();
        blockMessage.setMsgBody(block);
        Result result = this.messageBusService.sendToNode(blockMessage, fromNode, true);
        if (result.isFailed()) {
            Log.warn("send block failed:" + fromNode.getId() + ",height:" +
block.getHeader().getHeight());
        }
    }
}

108:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\GetBlocksByHeightHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.core.tools.log.Log;
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.model.Result;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.constant.MessageDataType;
import io.nuls.protocol.message.*;
import io.nuls.protocol.model.CompleteParam;
import io.nuls.protocol.model.GetBlocksByHeightParam;
import io.nuls.protocol.model.NotFound;
```

```
import io.nuls.protocol.service.BlockService;

import java.io.IOException;

/**
 * @author facjas
 */
public class GetBlocksByHeightHandler extends
AbstractMessageHandler<GetBlocksByHeightMessage> {

    private static final int MAX_SIZE = 1000;
    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);
    private MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);

    @Override
    public void onMessage(GetBlocksByHeightMessage message, Node fromNode) {

        if(message == null || message.getMsgBody() == null || fromNode == null) {
            return;
        }

        GetBlocksByHeightParam param = message.getMsgBody();
        if(param.getStartHeight() < 0L || param.getStartHeight() > param.getEndHeight()) {
            return;
        }
        if( param.getEndHeight() - param.getStartHeight() >= MAX_SIZE) {
            return;
        }

        NulsDigestData requestHash = null;
        try {
            requestHash = NulsDigestData.calcDigestData(param.serialize());
        } catch (IOException e) {
            e.printStackTrace();
        }

        // react request
        messageBusService.sendToNode(new ReactMessage(requestHash), fromNode, true);

        BlockHeader startBlockHeader =
blockService.getBlockHeader(param.getStartHeight()).getData();
```

```
    if(startBlockHeader == null) {
        sendNotFound(requestHash, fromNode);
        return;
    }
    Block endBlock = blockService.getBlock(param.getEndHeight()).getData();
    if(endBlock == null) {
        sendNotFound(requestHash, fromNode);
        return;
    }

    Block block = endBlock;
    while(true) {
        sendBlock(block, fromNode);
        if(block.getHeader().getHash().equals(startBlockHeader.getHash())) {
            break;
        }
        Result<Block> result = blockService.getBlock(block.getHeader().getPreHash());
        if (result.isFailed() || (block = result.getData()) == null) {
            sendNotFound(requestHash, fromNode);
            return;
        }
    }

    CompleteMessage completeMessage = new CompleteMessage();
    completeMessage.setMsgBody(new CompleteParam(requestHash, true));
    messageBusService.sendToNode(completeMessage, fromNode, true);
}

private void sendNotFound(NulsDigestData hash, Node node) {
    NotFoundMessage message = new NotFoundMessage();
    NotFound data = new NotFound(MessageDataType.BLOCKS, hash);
    message.setMsgBody(data);
    Result result = this.messageBusService.sendToNode(message, node, true);
    if (result.isFailed()) {
        Log.warn("send BLOCK NotFound failed:" + node.getId() + ", hash:" + hash);
    }
}

private void sendBlock(Block block, Node fromNode) {
    BlockMessage blockMessage = new BlockMessage();
    blockMessage.setMsgBody(block);
    Result result = this.messageBusService.sendToNode(blockMessage, fromNode, true);
```

```
        if (result.isFailed()) {
            Log.warn("send block failed:" + fromNode.getId() + ",height:" +
block.getHeader().getHeight());
        }
    }
}

109:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\GetBlocksHashHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.core.tools.log.BlockLog;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.constant.MessageDataType;
import io.nuls.protocol.message.BlocksHashMessage;
import io.nuls.protocol.message.GetBlocksHashMessage;
import io.nuls.protocol.message.NotFoundMessage;
import io.nuls.protocol.model.BlockHashResponse;
import io.nuls.protocol.model.GetBlocksHashParam;
import io.nuls.protocol.model.NotFound;
import io.nuls.protocol.service.BlockService;

/**
 * @author Niels
 */
public class GetBlocksHashHandler extends AbstractMessageHandler<GetBlocksHashMessage>
{

    private static final int MAX_SIZE = 10000;

    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);
    private MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);
```

```
@Override
public void onMessage(GetBlocksHashMessage message, Node fromNode) {
    GetBlocksHashParam param = message.getMsgBody();
    if (param.getEndHeight() - param.getStartHeight() >= MAX_SIZE) {
        return;
    }
    NulsDigestData requestHash = message.getHash();

    BlockHeader endHeader = blockService.getBlockHeader(param.getEndHeight()).getData();
    if (null == endHeader) {
        sendNotFound(fromNode, requestHash);
        return;
    }
    BlockHashResponse response = new BlockHashResponse();

    response.setRequestMessageHash(requestHash);
    BlockHeader header = endHeader;
    while (header.getHeight() >= param.getStartHeight()) {
        response.putFront(header.getHash());
        header = blockService.getBlockHeader(header.getPreHash()).getData();
        if(header == null) {
            break;
        }
    }
    sendResponse(response, fromNode);
}

private void sendNotFound(Node node, NulsDigestData hash) {
    NotFoundMessage event = new NotFoundMessage();
    NotFound data = new NotFound(MessageDataType.HASHES, hash);
    event.setMsgBody(data);
    Result result = this.messageBusService.sendToNode(event, node, true);
    if (result.isFailed()) {
        Log.warn("send not found failed:" + node.getId() + ", hash:" + hash);
    }
}

private void sendResponse(BlockHashResponse response, Node fromNode) {
    BlocksHashMessage event = new BlocksHashMessage();
    event.setMsgBody(response);
    Result result = messageBusService.sendToNode(event, fromNode, true);
    if (result.isFailed()) {
```

```
            BlockLog.debug("send block hashes to " + fromNode.getId() + " failed!");
        }
    }
}

110:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\GetSmallBlockHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.cache.TemporaryCacheManager;
import io.nuls.protocol.message.*;
import io.nuls.protocol.model.SmallBlock;

/**
 * @author facjas
 */
public class GetSmallBlockHandler extends AbstractMessageHandler<GetSmallBlockMessage> {

    private MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);
    private TemporaryCacheManager cacheManager = TemporaryCacheManager.getInstance();

    @Override
    public void onMessage(GetSmallBlockMessage message, Node fromNode) {
        if (message == null || fromNode == null || null == message.getMsgBody()) {
            return;
        }
        NulsDigestData blockHash = message.getMsgBody();
        SmallBlock smallBlock = cacheManager.getSmallBlockByHash(blockHash);
        if (null == smallBlock) {
            return;
        }
        SmallBlockMessage smallBlockMessage = new SmallBlockMessage();
        smallBlockMessage.setMsgBody(smallBlock);
        messageBusService.sendToNode(smallBlockMessage, fromNode, true);
```

```
    }
}

111:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\GetTxGroupHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.constant.MessageDataType;
import io.nuls.protocol.message.GetTxGroupRequest;
import io.nuls.protocol.message.NotFoundMessage;
import io.nuls.protocol.message.TxGroupMessage;
import io.nuls.protocol.model.GetTxGroupParam;
import io.nuls.protocol.model.NotFound;
import io.nuls.protocol.model.TxGroup;
import io.nuls.protocol.service.TransactionService;

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

/**
 * @author facjas
 */
public class GetTxGroupHandler extends AbstractMessageHandler<GetTxGroupRequest> {

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

    @Override
    public void onMessage(GetTxGroupRequest message, Node fromNode) {

        if(message == null || fromNode == null) {
            return;
        }
```

```
        GetTxGroupParam getTxGroupParam = message.getMsgBody();
        if (getTxGroupParam == null || getTxGroupParam.getTxHashList() == null ||
getTxGroupParam.getTxHashList().size() > 10000) {
            return;
        }
        NulsDigestData requestHash = null;
        try {
            requestHash = NulsDigestData.calcDigestData(getTxGroupParam.serialize());
        } catch (IOException e) {
            Log.error(e);
            return;
        }

        TxGroupMessage txGroupMessage = new TxGroupMessage();
        TxGroup txGroup = new TxGroup();
        List<Transaction> txList = new ArrayList<>();

        for (NulsDigestData hash : getTxGroupParam.getTxHashList()) {
            Transaction tx = transactionService.getTx(hash);
            if (tx != null) {
                txList.add(tx);
            } else {
                Log.error("GetTxGroupHandler NULL
TX=======================================hash: " + hash.getDigestHex());
                return;
            }
        }
        if (txList.isEmpty()) {
            Log.error("ASK:{}, {}", fromNode, getTxGroupParam.getTxHashList().get(0));
            return;
        }

        txGroup.setTxList(txList);
        txGroup.setRequestHash(requestHash);

        txGroupMessage.setMsgBody(txGroup);
        messageBusService.sendToNode(txGroupMessage, fromNode, true);
    }

}
```

```
112:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\handler\GetTxMessageHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.message.GetTxMessage;
import io.nuls.protocol.message.TransactionMessage;
import io.nuls.protocol.service.TransactionService;

/**
 * @author facjas
 */
public class GetTxMessageHandler extends AbstractMessageHandler<GetTxMessage> {

    private MessageBusService messageBusService = NulsContext.getServiceBean(MessageBusService.class);
    private TransactionService transactionService = NulsContext.getServiceBean(TransactionService.class);

    @Override
    public void onMessage(GetTxMessage message, Node fromNode) {
        if (message == null || fromNode == null || null == message.getMsgBody()) {
            return;
        }
        Transaction tx = transactionService.getTx(message.getMsgBody());
        if (null == tx) {
            return;
        }

        TransactionMessage txMessage = new TransactionMessage();
        txMessage.setMsgBody(tx);
        Result result = messageBusService.sendToNode(txMessage, fromNode, true);
        if (!result.isSuccess()) {
            Log.error("send error to node : " + fromNode.getId());
        }
```

```
    }

}
```

113:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\handler\NotFoundHander.java

```
 */

package io.nuls.protocol.base.handler;

import io.nuls.kernel.exception.NulsException;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.cache.ProtocolCacheHandler;
import io.nuls.protocol.message.NotFoundMessage;

/**
 * @author: Niels Wang
 */
public class NotFoundHander extends AbstractMessageHandler<NotFoundMessage> {
    /**
     * @param node the node who send this event!
     */
    @Override
    public void onMessage(NotFoundMessage event, Node node) throws NulsException {
        ProtocolCacheHandler.notFound(event.getMsgBody());
    }
}
```

114:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\handler\ReactMessageHandler.java

```
 *
 */

package io.nuls.protocol.base.handler;

import io.nuls.kernel.exception.NulsException;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.cache.ProtocolCacheHandler;
import io.nuls.protocol.message.ReactMessage;
import io.nuls.protocol.model.ReactParam;
```

```
/**
 * @author ln
 */
public class ReactMessageHandler extends AbstractMessageHandler<ReactMessage> {

    @Override
    public void onMessage(ReactMessage message, Node fromNode) throws NulsException {
        if(message == null || message.getMsgBody() == null || fromNode == null) {
            return;
        }
        ReactParam param = message.getMsgBody();
        if(param.getRequestId() == null) {
            return;
        }
        ProtocolCacheHandler.requestReact(param.getRequestId());
    }
}

115:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\SmallBlockHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.BlockLog;
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.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.utils.SmallBlockDuplicateRemoval;
import io.nuls.protocol.base.utils.AssemblyBlockUtil;
import io.nuls.protocol.cache.TemporaryCacheManager;
import io.nuls.protocol.message.GetTxGroupRequest;
import io.nuls.protocol.message.SmallBlockMessage;
```

```
import io.nuls.protocol.model.GetTxGroupParam;
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.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author facjas
 */
public class SmallBlockHandler extends AbstractMessageHandler<SmallBlockMessage> {

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

    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);
    private TemporaryCacheManager temporaryCacheManager =
TemporaryCacheManager.getInstance();
    private TransactionService transactionService =
NulsContext.getServiceBean(TransactionService.class);

    @Override
    public void onMessage(SmallBlockMessage event, Node fromNode) {
        SmallBlock smallBlock = event.getMsgBody();
        if (null == smallBlock) {
            Log.warn("recieved a null smallBlock!");
            return;
        }

        BlockHeader header = smallBlock.getHeader();
        //
        if (header.getTime() > (TimeService.currentTimeMillis() +
ProtocolConstant.BLOCK_TIME_INTERVAL_SECOND * 1000)) {
            return;
        }

        if (!SmallBlockDuplicateRemoval.needProcess(header.getHash())) {
            return;
```

```
        }

        BlockHeader theBlockHeader = blockService.getBlockHeader(header.getHash()).getData();
        if (null != theBlockHeader) {
            return;
        }

        ValidateResult result = header.verify();
        boolean isOrphan = result.getErrorCode() == TransactionErrorCode.ORPHAN_TX ||
result.getErrorCode() == TransactionErrorCode.ORPHAN_BLOCK;
        BlockLog.debug("recieve new block from(" + fromNode.getId() + "), tx count : " +
header.getTxCount() + " , tx pool count : " + consensusService.getMemoryTxs().size() + " , header
height:" + header.getHeight() + ", preHash:" + header.getPreHash() + " , hash:" +
header.getHash() + ", addressHex:" +
AddressTool.getStringAddressByBytes(header.getPackingAddress()) +
                "\n and verify block result: " + result.isSuccess() + " , verify message : " + result.getMsg()
+ " , isOrphan : " + isOrphan);

        if (result.isFailed() && !isOrphan) {
            BlockLog.debug("discard a SmallBlock:" + smallBlock.getHeader().getHash() + ", from:" +
fromNode.getId() + " ,reason:" + result.getMsg());
            return;
        }
        Map<NulsDigestData, Transaction> txMap = new HashMap<>();
        for (Transaction tx : smallBlock.getSubTxList()) {
            txMap.put(tx.getHash(), tx);
        }
        List<NulsDigestData> needHashList = new ArrayList<>();
        for (NulsDigestData hash : smallBlock.getTxHashList()) {
            Transaction tx = txMap.get(hash);
            if (null == tx) {
                tx = transactionService.getTx(hash);
                if (tx != null) {
                    smallBlock.getSubTxList().add(tx);
                    txMap.put(hash, tx);
                }
            }
            if (null == tx) {
                needHashList.add(hash);
            }
        }
        if (!needHashList.isEmpty()) {
```

```
        Log.info("block height : " + header.getHeight() + ", tx count : " + header.getTxCount() + " ,
get group tx of " + needHashList.size());
        GetTxGroupRequest request = new GetTxGroupRequest();
        GetTxGroupParam param = new GetTxGroupParam();
        param.setTxHashList(needHashList);
        request.setMsgBody(param);
        Result sendResult = this.messageBusService.sendToNode(request, fromNode, true);
        if (sendResult.isFailed()) {
            Log.warn("get tx group failed,height:" + header.getHeight());
        } else {
            NulsDigestData requestHash = null;
            try {
                requestHash = NulsDigestData.calcDigestData(request.getMsgBody().serialize());
            } catch (IOException e) {
                Log.error(e);
                return;
            }
            temporaryCacheManager.cacheSmallBlockWithRequest(requestHash, smallBlock);
        }
        return;
    }

    Block block = AssemblyBlockUtil.assemblyBlock(header, txMap,
smallBlock.getTxHashList());
    consensusService.newBlock(block, fromNode);
  }
}

116:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\TransactionMessageHandler.java
 */
package io.nuls.protocol.base.handler;

import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.cache.TransactionDuplicateRemoval;
import io.nuls.protocol.message.TransactionMessage;
import io.nuls.protocol.service.TransactionService;
```

```
import java.util.HashSet;
import java.util.Set;

/**
 * @author Niels
 */
public class TransactionMessageHandler extends
AbstractMessageHandler<TransactionMessage> {

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

    @Override
    public void onMessage(TransactionMessage message, Node fromNode) {

        Transaction tx = message.getMsgBody();
        if (null == tx) {
            return;
        }
        if (tx.isSystemTx()) {
            return;
        }
        NulsDigestData hash = tx.getHash();
        TransactionDuplicateRemoval.insert(hash);
        transactionService.newTx(tx);
    }

}

117:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\handler\TxGroupHandler.java
 */
package io.nuls.protocol.base.handler;

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.model.*;
import io.nuls.message.bus.handler.AbstractMessageHandler;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.cache.ProtocolCacheHandler;
```

```
import io.nuls.protocol.base.utils.AssemblyBlockUtil;
import io.nuls.protocol.cache.TemporaryCacheManager;
import io.nuls.protocol.message.GetTxGroupRequest;
import io.nuls.protocol.message.TxGroupMessage;
import io.nuls.protocol.model.GetTxGroupParam;
import io.nuls.protocol.model.SmallBlock;
import io.nuls.protocol.model.TxGroup;

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

/**
 * @author facjas
 */
public class TxGroupHandler extends AbstractMessageHandler<TxGroupMessage> {

    private TemporaryCacheManager temporaryCacheManager =
TemporaryCacheManager.getInstance();

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

    @Override
    public void onMessage(TxGroupMessage message, Node fromNode) throws NulsException {
        TxGroup txGroup = message.getMsgBody();
        if (null == txGroup) {
            Log.warn("recieved a null txGroup form " + fromNode.getId());
            return;
        }

        SmallBlock smallBlock =
temporaryCacheManager.getSmallBlockByRequest(txGroup.getRequestHash());
        if (null == smallBlock) {
            return;
        }
        BlockHeader header = smallBlock.getHeader();
        Map<NulsDigestData, Transaction> txMap = new HashMap<>();
        for (Transaction tx : smallBlock.getSubTxList()) {
            txMap.put(tx.getHash(), tx);
```

```
        }
        for(Transaction tx :txGroup.getTxList()) {
            txMap.put(tx.getHash(), tx);
        }
        for (NulsDigestData hash : smallBlock.getTxHashList()) {
            Transaction tx = txMap.get(hash);
            if (null == tx) {
                tx = temporaryCacheManager.getTx(hash);
            }
            if (tx != null) {
                smallBlock.getSubTxList().add(tx);
                txMap.put(hash, tx);
            }
        }

        Block block = AssemblyBlockUtil.assemblyBlock(header, txMap,
smallBlock.getTxHashList());
        consensusService.newBlock(block, fromNode);

    }
}
```

118:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\module\BaseProtocolsModuleBootstrap.java

```
 */
package io.nuls.protocol.base.module;


import io.nuls.consensus.constant.ConsensusConstant;
import io.nuls.core.tools.log.Log;
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.thread.manager.TaskManager;
import io.nuls.kernel.utils.TransactionManager;
import io.nuls.message.bus.constant.MessageBusConstant;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.constant.NetworkConstant;
import io.nuls.protocol.base.handler.*;
import io.nuls.protocol.base.service.DownloadServiceImpl;
import io.nuls.protocol.message.*;
```

```
import io.nuls.protocol.model.tx.CoinBaseTransaction;
import io.nuls.protocol.model.tx.DataTransaction;
import io.nuls.protocol.model.tx.TransferTransaction;
import io.nuls.protocol.module.AbstractProtocolModule;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.DownloadService;


/**
 * @author Niels
 */
public class BaseProtocolsModuleBootstrap extends AbstractProtocolModule {

    @Override
    public void init() {
        TransactionManager.putTx(CoinBaseTransaction.class, null);
        TransactionManager.putTx(TransferTransaction.class, null);
        TransactionManager.putTx(DataTransaction.class, null);
    }

    @Override
    public void start() {
        this.waitForDependencyRunning(MessageBusConstant.MODULE_ID_MESSAGE_BUS);
        this.waitForDependencyInited(ConsensusConstant.MODULE_ID_CONSENSUS,
NetworkConstant.NETWORK_MODULE_ID);
        BlockService blockService = NulsContext.getServiceBean(BlockService.class);

        Block block0 = blockService.getGengsisBlock().getData();
        Block genesisBlock = NulsContext.getInstance().getGenesisBlock();
        if (null == block0) {
            try {
                blockService.saveBlock(genesisBlock);
            } catch (NulsException e) {
                Log.error(e);
                throw new NulsRuntimeException(e);
            }
        }
        Block block = blockService.getBestBlock().getData();
        while (null != block && block.verify().isFailed()) {
            try {
                blockService.rollbackBlock(block);
            } catch (NulsException e) {
```

```
                Log.error(e);
            }
            block = blockService.getBlock(block.getHeader().getPreHash()).getData();
        }
        if (null != block) {
            NulsContext.getInstance().setBestBlock(block);
            this.initHandlers();
            ((DownloadServiceImpl) NulsContext.getServiceBean(DownloadService.class)).start();
        } else {
            start();
        }

    }

    private void initHandlers() {
        MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);
        messageBusService.subscribeMessage(BlockMessage.class, new BlockMessageHandler());
        messageBusService.subscribeMessage(BlocksHashMessage.class, new
BlocksHashHandler());
        messageBusService.subscribeMessage(GetBlocksHashMessage.class, new
GetBlocksHashHandler());
        messageBusService.subscribeMessage(NotFoundMessage.class, new NotFoundHander());
        messageBusService.subscribeMessage(GetBlockMessage.class, new GetBlockHandler());
        messageBusService.subscribeMessage(GetBlocksByHashMessage.class, new
GetBlocksByHashHandler());
        messageBusService.subscribeMessage(GetBlocksByHeightMessage.class, new
GetBlocksByHeightHandler());
        messageBusService.subscribeMessage(GetTxGroupRequest.class, new
GetTxGroupHandler());
        messageBusService.subscribeMessage(TxGroupMessage.class, new TxGroupHandler());
        messageBusService.subscribeMessage(TransactionMessage.class, new
TransactionMessageHandler());
        messageBusService.subscribeMessage(SmallBlockMessage.class, new
SmallBlockHandler());
        messageBusService.subscribeMessage(CompleteMessage.class, new CompleteHandler());
        messageBusService.subscribeMessage(ReactMessage.class, new
ReactMessageHandler());

//        TaskManager.createAndRunThread(ProtocolConstant.MODULE_ID_PROTOCOL, "Tx-
Download", TransactionDownloadProcessor.getInstance());
        messageBusService.subscribeMessage(GetTxMessage.class, new
```

```
GetTxMessageHandler());
//
//        TaskManager.createAndRunThread(ProtocolConstant.MODULE_ID_PROTOCOL,
"SmallBlock-Download", SmallBlockDownloadProcessor.getInstance());
        messageBusService.subscribeMessage(GetSmallBlockMessage.class, new
GetSmallBlockHandler());
        messageBusService.subscribeMessage(ForwardSmallBlockMessage.class, new
ForwardSmallBlockHandler());
        messageBusService.subscribeMessage(ForwardTxMessage.class, new
ForwardTxMessageHandler());
    }

    @Override
    public void shutdown() {
        ((DownloadServiceImpl) NulsContext.getServiceBean(DownloadService.class)).stop();
        TaskManager.shutdownByModuleId(this.getModuleId());
    }

    @Override
    public void destroy() {
    }

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

}

119:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\service\BlockServiceImpl.java
 */

package io.nuls.protocol.base.service;

import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTransfer;
import io.nuls.contract.entity.tx.CallContractTransaction;
import io.nuls.contract.entity.tx.ContractTransferTransaction;
import io.nuls.contract.service.ContractService;
import io.nuls.core.tools.log.Log;
```

```
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.*;
import io.nuls.ledger.service.LedgerService;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.utils.PoConvertUtil;
import io.nuls.protocol.constant.ProtocolErroeCode;
import io.nuls.protocol.message.ForwardSmallBlockMessage;
import io.nuls.protocol.message.SmallBlockMessage;
import io.nuls.protocol.model.SmallBlock;
import io.nuls.protocol.service.BlockService;
import io.nuls.protocol.service.TransactionService;
import io.nuls.protocol.storage.po.BlockHeaderPo;
import io.nuls.protocol.storage.service.BlockHeaderStorageService;

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

/**
 *
 * Block processing service classes.
 *
 * @author: Niels Wang
 */
@Service("blockService")
public class BlockServiceImpl implements BlockService {

    /**
     *
     * Storage utility class
     */
    @Autowired
    private BlockHeaderStorageService blockHeaderStorageService;

    @Autowired
    private LedgerService ledgerService;

    @Autowired
    private TransactionService transactionService;
```

```
@Autowired
private MessageBusService messageBusService;
@Autowired
private AccountLedgerService accountLedgerService;

@Autowired
private ContractService contractService;

/**
 *
 * Get the creation block (from storage)
 */
@Override
public Result<Block> getGengsisBlock() {
    BlockHeaderPo headerPo = blockHeaderStorageService.getBlockHeaderPo(0);
    if (null == headerPo) {
        return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
    }
    Block block = getBlock(headerPo);
    return Result.getSuccess().setData(block);
}

/**
 *
 * Get the highest block (from storage)
 */
@Override
public Result<Block> getBestBlock() {
    BlockHeaderPo headerPo = blockHeaderStorageService.getBestBlockHeaderPo();
    if (null == headerPo) {
        return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
    }
    Block block = getBlock(headerPo);
    return Result.getSuccess().setData(block);
}

/**
 * po
 * Assemble the complete block according to block head Po.
 *
 * @param headerPo /block header po
```

```
     * @return /the complete block
     */
    private Block getBlock(BlockHeaderPo headerPo) {
        return getBlock(headerPo, false);
    }

    private Block getBlock(BlockHeaderPo headerPo, boolean isNeedContractTransfer) {
        List<Transaction> txList = new ArrayList<>();
        for (NulsDigestData hash : headerPo.getTxHashList()) {
            Transaction tx = ledgerService.getTx(hash);
            txList.add(tx);
            if(isNeedContractTransfer) {
                //pierre add ()
                contractTransfer(tx, txList);
            }
        }
        Block block = new Block();
        BlockHeader blockHeader = PoConvertUtil.fromBlockHeaderPo(headerPo);
        if(isNeedContractTransfer) {
            blockHeader.setTxCount(txList.size());
        }
        block.setHeader(blockHeader);
        block.setTxs(txList);
        return block;
    }

    private void contractTransfer(Transaction tx, List<Transaction> txList) {
        if(tx instanceof CallContractTransaction) {
            CallContractTransaction callTx = (CallContractTransaction) tx;
            ContractResult contractResult = callTx.getContractResult();
            if(contractResult != null) {
                List<ContractTransfer> transfers = contractResult.getTransfers();
                // ()
                if(transfers != null && transfers.size() > 0) {
                    for(ContractTransfer transfer : transfers) {
                        Transaction contractTx = ledgerService.getTx(transfer.getHash());
                        if(contractTx != null) {
                            txList.add(contractTx);
                        }
                    }
                }
            }
```

```
    }
}

/**
 *
 * Get the highest block header (from storage)
 */
@Override
public Result<BlockHeader> getBestBlockHeader() {
    BlockHeaderPo headerPo = blockHeaderStorageService.getBestBlockHeaderPo();
    if (null == headerPo) {
        return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
    }
    return Result.getSuccess().setData(PoConvertUtil.fromBlockHeaderPo(headerPo));
}

/**
 *
 * Get the block head (from storage) according to the block height
 *
 * @param height /block height
 * @return
 */
@Override
public Result<BlockHeader> getBlockHeader(long height) {
    BlockHeaderPo headerPo = blockHeaderStorageService.getBlockHeaderPo(height);
    if (null == headerPo) {
        return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
    }
    return Result.getSuccess().setData(PoConvertUtil.fromBlockHeaderPo(headerPo));
}

/**
 *
 * Get the block head (from storage) according to the block hash
 *
 * @param hash /block hash
 * @return /block header
 */
@Override
public Result<BlockHeader> getBlockHeader(NulsDigestData hash) {
    BlockHeaderPo headerPo = blockHeaderStorageService.getBlockHeaderPo(hash);
```

```
        if (null == headerPo) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        return Result.getSuccess().setData(PoConvertUtil.fromBlockHeaderPo(headerPo));
    }

    /**
     *
     * Get the block (from storage) according to the block hash
     *
     * @param hash /block hash
     * @return /block
     */
    @Override
    public Result<Block> getBlock(NulsDigestData hash) {
        BlockHeaderPo headerPo = blockHeaderStorageService.getBlockHeaderPo(hash);
        if (null == headerPo) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        Block block = getBlock(headerPo);
        return Result.getSuccess().setData(block);
    }

    /**
     *
     * Get the block (from storage) according to the block hash
     *
     * @param hash /block hash
     * @param isNeedContractTransfer ()/If necessary to add the contract transfer (from the
contract) to the block
     * @return /block
     */
    @Override
    public Result<Block> getBlock(NulsDigestData hash, boolean isNeedContractTransfer) {
        BlockHeaderPo headerPo = blockHeaderStorageService.getBlockHeaderPo(hash);
        if (null == headerPo) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        Block block = getBlock(headerPo, isNeedContractTransfer);
        return Result.getSuccess().setData(block);
    }
```

```
    /**
     *
     * Get the block (from storage) according to the block height
     *
     * @param height /block height
     * @return /block
     */
    @Override
    public Result<Block> getBlock(long height) {
        BlockHeaderPo headerPo = blockHeaderStorageService.getBlockHeaderPo(height);
        if (null == headerPo) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        Block block = getBlock(headerPo);
        return Result.getSuccess().setData(block);
    }

    /**
     *
     * Get the block (from storage) according to the block height
     *
     * @param height /block height
     * @param isNeedContractTransfer ()/If necessary to add the contract transfer (from the
contract) to the block
     * @return /block
     */
    @Override
    public Result<Block> getBlock(long height, boolean isNeedContractTransfer) {
        BlockHeaderPo headerPo = blockHeaderStorageService.getBlockHeaderPo(height);
        if (null == headerPo) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        Block block = getBlock(headerPo, isNeedContractTransfer);
        return Result.getSuccess().setData(block);
    }

    /**
     *
     * Save the block to the store.
     *
     * @param block /whole block
     * @return /operating result
```

```
     * @throws NulsException /There may be exceptions to the save block, please handle it carefully after capture.
     */
    @Override
    public Result saveBlock(Block block) throws NulsException {
        if (null == block || block.getHeader() == null || block.getTxs() == null) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        long height = block.getHeader().getHeight();
        List<Transaction> savedList = new ArrayList<>();
        for (Transaction transaction : block.getTxs()) {
            transaction.setBlockHeight(height);
            Result result = transactionService.commitTx(transaction, block.getHeader());
            if (result.isSuccess()) {
                result = ledgerService.saveTx(transaction);
            }
            if (result.isSuccess()) {
                savedList.add(transaction);
            } else {
                this.rollbackTxList(savedList, block.getHeader(), false);
                return result;
            }
        }
        Result result = this.blockHeaderStorageService.saveBlockHeader(PoConvertUtil.toBlockHeaderPo(block));
        if (result.isFailed()) {
            this.rollbackTxList(savedList, block.getHeader(), false);
            return result;
        }
        try {
            accountLedgerService.saveConfirmedTransactionList(block.getTxs());
            //
            contractService.saveConfirmedTransactionList(block.getTxs());
        } catch (Exception e) {
            Log.warn("save local tx failed", e);
        }
        return Result.getSuccess();
    }

    /**
     *
     * When you fail to save the block, you need to roll back the already stored transaction.
```

```
     */
    private boolean rollbackTxList(List<Transaction> savedList, BlockHeader blockHeader, boolean
atomicity) throws NulsException {
        List<Transaction> rollbackedList = new ArrayList<>();
        for (int i = savedList.size() - 1; i >= 0; i--) {
            Transaction tx = savedList.get(i);
            Result result = transactionService.rollbackTx(tx, blockHeader);
            if (atomicity) {
                if (result.isFailed()) {
                    break;
                } else {
                    rollbackedList.add(tx);
                }
            }
        }
        if (atomicity && savedList.size() != rollbackedList.size()) {
            for (int i = rollbackedList.size() - 1; i >= 0; i--) {
                Transaction tx = rollbackedList.get(i);
                transactionService.commitTx(tx, blockHeader);
            }
            return false;
        }
        return true;
    }

    /**
     *
     * roll back the block to the store.
     *
     * @param block /whole block
     * @return /operating result
     * @throws NulsException /There may be exceptions to the roll back block, please handle it
carefully after capture.
     */
    @Override
    public Result rollbackBlock(Block block) throws NulsException {
        if (null == block) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        boolean b = this.rollbackTxList(block.getTxs(), block.getHeader(), true);
        if (!b) {
            return Result.getFailed(KernelErrorCode.DATA_ERROR);
```

```
        }
        BlockHeaderPo po = new BlockHeaderPo();
        po.setHash(block.getHeader().getHash());
        po.setHeight(block.getHeader().getHeight());
        po.setPreHash(block.getHeader().getPreHash());
        Result result = this.blockHeaderStorageService.removeBlockHerader(po);
        if (result.isFailed()) {
            return result;
        }
        try {
            accountLedgerService.rollbackTransactions(block.getTxs());
            //
            contractService.rollbackTransactionList(block.getTxs());
        } catch (Exception e) {
            Log.warn("rollbackTransaction local tx failed", e);
        }
        return result;
    }

    /**
     *
     * Forward block to other peers of the connection, allowing one column (not forward to it)
     *
     * @param blockHash   /the hash of block
     * @param excludeNode /The nodes that need to be excluded are generally due to the block received from the node.
     * @return /forward results
     */
    @Override
    public Result forwardBlock(NulsDigestData blockHash, Node excludeNode) {
        ForwardSmallBlockMessage message = new ForwardSmallBlockMessage();
        message.setMsgBody(blockHash);
        return messageBusService.broadcast(message, excludeNode, true, 100);
    }

    /**
     *
     * The broadcast small block gives the connection to other peers.
     *
     * @param smallBlock /the small block
     * @return /Broadcast the results
     */
```

```
    @Override
    public Result broadcastBlock(SmallBlock smallBlock) {
        SmallBlockMessage message = fillSmallBlockMessage(smallBlock);
        Result<List<String>> result = messageBusService.broadcast(message, null, true, 100);
        return result;
    }

    /**
     *
     * the block is put into the message container and the message container is returned.
     *
     * @param smallBlock
     * @return /Block message container.
     */
    private SmallBlockMessage fillSmallBlockMessage(SmallBlock smallBlock) {
        SmallBlockMessage message = new SmallBlockMessage();
        message.setMsgBody(smallBlock);
        return message;
    }
}

120:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\service\DownloadServiceImpl.java
 */

package io.nuls.protocol.base.service;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.network.model.Node;
import io.nuls.protocol.base.constant.DownloadStatus;
import io.nuls.protocol.base.download.processor.DownloadProcessor;
import io.nuls.protocol.base.download.utils.DownloadUtils;
import io.nuls.protocol.constant.ProtocolErroeCode;
import io.nuls.protocol.model.TxGroup;
import io.nuls.protocol.service.DownloadService;

import java.util.List;
```

```
/**
 * @author ln
 */
@Service
public class DownloadServiceImpl implements DownloadService {

    private DownloadProcessor processor = DownloadProcessor.getInstance();

    /**
     * hash
     * Download a block according from the node to the hash, and the download process is blocked.
     *
     * @param hash /block hash
     * @param node /Specified node
     * @return / block & results
     */
    @Override
    public Result<Block> downloadBlock(NulsDigestData hash, Node node) {
        Block block = null;
        try {
            block = DownloadUtils.getBlockByHash(hash, node);
        } catch (RuntimeException e) {
            return Result.getFailed(KernelErrorCode.SYS_UNKOWN_EXCEPTION);
        }
        if (block == null) {
            return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        }
        return Result.getSuccess().setData(block);
    }

    /**
     *
     * Returns the results of the download.
     */
    @Override
    public Result isDownloadSuccess() {
        if (processor.getDownloadStatus() == DownloadStatus.SUCCESS) {
            return Result.getSuccess();
        }
        return Result.getFailed(KernelErrorCode.FAILED);
    }
```

```
    public boolean start() {
        processor = DownloadProcessor.getInstance();
        return processor.startup();
    }

    public boolean stop() {
        return processor.shutdown();
    }

    @Override
    public Result reset() {
        processor.setDownloadStatus(DownloadStatus.WAIT);
        return Result.getSuccess();
    }

}
```

121:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\service\TransactionServiceImpl.java

```
 */

package io.nuls.protocol.base.service;

import io.nuls.account.service.AccountService;
import io.nuls.consensus.service.ConsensusService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
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.TransactionManager;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.service.LedgerService;
import io.nuls.message.bus.service.MessageBusService;
import io.nuls.network.model.Node;
import io.nuls.protocol.cache.TemporaryCacheManager;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.message.ForwardTxMessage;
import io.nuls.protocol.message.TransactionMessage;
```

```
import io.nuls.protocol.service.TransactionService;

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

/**
 * @author: Niels Wang
 */
@Service
public class TransactionServiceImpl implements TransactionService {

    private TemporaryCacheManager temporaryCacheManager =
TemporaryCacheManager.getInstance();

    @Autowired
    private MessageBusService messageBusService;
    @Autowired
    private LedgerService ledgerService;
    @Autowired
    private ConsensusService consensusService;

    @Autowired
    private AccountService accountService;

    /**
     *
     * Identify the method that is invoked during the transaction and submit the transaction related
business.
     *
     * @param tx            /The transaction of the operation
     * @param secondaryData /Secondary data (available for null)
     * @return /operating results
     */
    @Override
    public Result commitTx(Transaction tx, Object secondaryData) {
        List<TransactionProcessor> processorList =
TransactionManager.getProcessorList(tx.getClass());
        List<TransactionProcessor> commitedProcessorList = new ArrayList<>();
        for (TransactionProcessor processor : processorList) {
            Result result = processor.onCommit(tx, secondaryData);
            if (result.isSuccess()) {
                commitedProcessorList.add(processor);
```

```
            } else {
                for (int i = commitedProcessorList.size() - 1; i >= 0; i--) {
                    TransactionProcessor processor1 = commitedProcessorList.get(i);
                    processor1.onRollback(tx, secondaryData);
                }
                return result;
            }
        }
        return Result.getSuccess();
    }

    /**
     *
     * The method invoked when the transaction is rolled back and the transaction related business is returned.
     *
     * @param tx            /The transaction of the operation
     * @param secondaryData /Secondary data (available for null)
     * @return /operating results
     */
    @Override
    public Result rollbackTx(Transaction tx, Object secondaryData) {
        if (null == tx) {
            return Result.getSuccess();
        }
        List<TransactionProcessor> processorList = TransactionManager.getProcessorList(tx.getClass());
        List<TransactionProcessor> rollbackedList = new ArrayList<>();
        for (TransactionProcessor processor : processorList) {
            Result result = processor.onRollback(tx, secondaryData);
            if (result.isSuccess()) {
                rollbackedList.add(processor);
            } else {
                for (int i = rollbackedList.size() - 1; i >= 0; i--) {
                    TransactionProcessor processor1 = rollbackedList.get(i);
                    processor1.onCommit(tx, secondaryData);
                }
                return result;
            }
        }
        try {
            ledgerService.rollbackTx(tx);
```

```
        } catch (NulsException e) {
            Log.error(e);
            return Result.getFailed(e.getErrorCode());
        }
        return Result.getSuccess();
    }

    /**
     *
     * Forward Transaction to other peers of the connection, allowing one column (not forward to it)
     *
     * @param tx          /the whole transaction
     * @param excludeNode /The nodes that need to be excluded are generally due to the
transaction received from the node.
     * @return /forward results
     */
    @Override
    public Result forwardTx(Transaction tx, Node excludeNode) {
        ForwardTxMessage message = new ForwardTxMessage();
        message.setMsgBody(tx.getHash());
        return messageBusService.broadcast(message, excludeNode, true, 50);
    }

    /**
     *
     * The broadcast transaction gives the connection to other peers.
     *
     * @param tx /the whole transaction
     * @return /Broadcast the results
     */
    @Override
    public Result broadcastTx(Transaction tx) {
        TransactionMessage message = new TransactionMessage();
        message.setMsgBody(tx);
        consensusService.newTx(tx);
        // pierre test comment out
        //return Result.getSuccess();
        return messageBusService.broadcast(message, null, true, 50);
    }

    @Override
    public Result newTx(Transaction tx) {
```

```
        return consensusService.newTx(tx);
    }

    /**
     *
     * <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.
     *
     * @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();
        }
//        ValidateResult result = ledgerService.verifyDoubleSpend(txList);
//        if (result.isFailed()) {
//            return result;
//        }
        List<Transaction> newTxList = new ArrayList<>();
        for (Transaction tx : txList) {
            if (tx.getType() == ProtocolConstant.TX_TYPE_COINBASE || tx.getType() == ProtocolConstant.TX_TYPE_TRANSFER) {
                continue;
            }
            newTxList.add(tx);
        }
        List<TransactionProcessor> processorList = TransactionManager.getAllProcessorList();
        ValidateResult result = ValidateResult.getSuccessResult();
        for (TransactionProcessor processor : processorList) {
            result = processor.conflictDetect(newTxList);
            if (result.isFailed()) {
                break;
            }
        }
        return result;
    }
```

```
    @Override
    public Transaction getTx(NulsDigestData hash) {
        return consensusService.getTx(hash);
    }


}
```

122:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\utils\AssemblyBlockUtil.java

```
 */

package io.nuls.protocol.base.utils;

import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;

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

/**
 * @author: Niels Wang
 */
public class AssemblyBlockUtil {
    public static Block assemblyBlock(BlockHeader header, Map<NulsDigestData, Transaction> txMap, List<NulsDigestData> txHashList) {
        Block block = new Block();
        block.setHeader(header);
        List<Transaction> txs = new ArrayList<>();
        for (NulsDigestData txHash : txHashList) {
            Transaction tx = txMap.get(txHash);
            if (null == tx) {
                throw new NulsRuntimeException(TransactionErrorCode.TX_NOT_EXIST);
            }
            tx.setBlockHeight(header.getHeight());
            txs.add(tx);
```

```
        }
        block.setTxs(txs);
        return block;
    }
}

123:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\utils\BlockRequest.java
 */

package io.nuls.protocol.base.utils;

import io.nuls.kernel.func.TimeService;
import io.nuls.network.model.Node;
import io.nuls.protocol.message.GetBlockMessage;

/**
 * @author: Niels Wang
 */
public class BlockRequest {
    private GetBlockMessage message;
    private Node node;
    private long time;

    public BlockRequest(GetBlockMessage message, Node node) {
        this.message = message;
        this.node = node;
        this.time = TimeService.currentTimeMillis();
    }

    public GetBlockMessage getMessage() {
        return message;
    }

    public Node getNode() {
        return node;
    }

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

```
124:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\utils\BlockSendThread.java
// */
//
//package io.nuls.protocol.base.utils;
//
//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.NulsDigestData;
//import io.nuls.kernel.model.Result;
//import io.nuls.message.bus.service.MessageBusService;
//import io.nuls.network.model.Node;
//import io.nuls.protocol.constant.MessageDataType;
//import io.nuls.protocol.message.BlockMessage;
//import io.nuls.protocol.message.GetBlockMessage;
//import io.nuls.protocol.message.NotFoundMessage;
//import io.nuls.protocol.model.GetBlocksByHashParam;
//import io.nuls.protocol.model.NotFound;
//import io.nuls.protocol.service.BlockService;
//
//import java.util.concurrent.BlockingQueue;
//import java.util.concurrent.LinkedBlockingDeque;
//
///**
// * @author: Niels Wang
// */
//public class BlockSendThread implements Runnable {
//
//    private static final int MAX_SIZE = 1000;
//    private BlockService blockService = NulsContext.getServiceBean(BlockService.class);
//    private MessageBusService messageBusService =
NulsContext.getServiceBean(MessageBusService.class);
//
//    private static BlockingQueue<BlockRequest> queue = new LinkedBlockingDeque<>(1024);
//
//    /**
//     *
//     */
//    @Override
```

```
//    public void run() {
//        while (true) {
//            try {
//                BlockRequest request = queue.take();
//                if ((TimeService.currentTimeMillis() - request.getTime()) >= 30000L) {
//                    continue;
//                }
//                execute(request);
//            } catch (Exception e) {
//                Log.error(e);
//            }
//        }
//    }
//
//    private void execute(BlockRequest request) {
//        GetBlockMessage message = request.getMessage();
//        Node fromNode = request.getNode();
//        GetBlocksByHashParam param = message.getMsgBody();
//        if (param.getSize() > MAX_SIZE) {
//            return;
//        }
//        if (param.getSize() == 1) {
//            Block block = null;
//            Result<Block> result = this.blockService.getBlock(param.getStartHash());
//            if (result.isFailed()) {
//                sendNotFound(param.getStartHash(), fromNode);
//                return;
//            }
//            block = result.getData();
//            sendBlock(block, fromNode);
//            return;
//        }
//        Block chainStartBlock = null;
//        Result<Block> blockResult = this.blockService.getBlock(param.getStartHash());
//        if (blockResult.isFailed()) {
//            sendNotFound(param.getStartHash(), fromNode);
//            return;
//        } else {
//            chainStartBlock = blockResult.getData();
//        }
//        Block chainEndBlock = null;
//        blockResult = this.blockService.getBlock(param.getEndHash());
```

```
//        if (blockResult.isFailed()) {
//            sendNotFound(param.getEndHash(), fromNode);
//            return;
//        } else {
//            chainEndBlock = blockResult.getData();
//        }
//        if (chainEndBlock.getHeader().getHeight() < chainStartBlock.getHeader().getHeight()) {
//            return;
//        }
//        long end = param.getStart() + param.getSize() - 1;
//        if (chainStartBlock.getHeader().getHeight() > param.getStart() ||
chainEndBlock.getHeader().getHeight() < end) {
//            sendNotFound(param.getStartHash(), fromNode);
//            return;
//        }
//
//        Block block = chainEndBlock;
//        while (true) {
//            this.sendBlock(block, fromNode);
//            if (block.getHeader().getHash().equals(chainStartBlock.getHeader().getHash())) {
//                break;
//            }
//            if (block.getHeader().getPreHash().equals(chainStartBlock.getHeader().getHash())) {
//                block = chainStartBlock;
//                continue;
//            }
//            block = blockService.getBlock(block.getHeader().getPreHash()).getData();
//        }
//    }
//
//    private void sendNotFound(NulsDigestData hash, Node node) {
//        NotFoundMessage event = new NotFoundMessage();
//        NotFound data = new NotFound(MessageDataType.BLOCK, hash);
//        event.setMsgBody(data);
//        Result result = this.messageBusService.sendToNode(event, node, true);
//        if (result.isFailed()) {
//            Log.warn("send not found failed:" + node.getId() + ", hash:" + hash);
//        }
//    }
//
//    private void sendBlock(Block block, Node fromNode) {
//        if (null == block) {
```

```
//            Log.warn("there is a null block");
//            return;
//        }
//        BlockMessage blockMessage = new BlockMessage();
//        blockMessage.setMsgBody(block);
//        Result result = this.messageBusService.sendToNode(blockMessage, fromNode, true);
//        if (result.isFailed()) {
//            Log.warn("send block failed:" + fromNode.getId() + ",height:" +
block.getHeader().getHeight());
//        }
//    }
//
//    public static void offer(GetBlockMessage request, Node fromNode) {
//        queue.offer(new BlockRequest(request, fromNode));
//    }
//}

125:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\utils\filter\InventoryFilter.java
 */

package io.nuls.protocol.base.utils.filter;

import com.google.common.hash.BloomFilter;
import com.google.common.hash.Funnels;

import java.util.concurrent.atomic.AtomicInteger;

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

    private final int elements;
    private AtomicInteger size = new AtomicInteger(0);

    private BloomFilter<byte[]> filter;

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

```
    public InventoryFilter(int elements) {
        this.elements = elements;
        filter = BloomFilter.create(Funnels.byteArrayFunnel(), elements, 0.00001);
    }

    public BloomFilter getFilter() {
        return filter;
    }

    public void insert(byte[] object) {
//        lock.lock();
//        try {
        filter.put(object);
        int count = size.incrementAndGet();
        if (count >= elements - 100) {
            this.clear();
        }
//        } finally {
//            lock.unlock();
//        }
    }

    public boolean contains(byte[] object) {
        return filter.mightContain(object);
    }

    public void clear() {
        filter = BloomFilter.create(Funnels.byteArrayFunnel(), elements, 0.00001);
    }
}

126:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\utils\PoConvertUtil.java
 */

package io.nuls.protocol.base.utils;

import io.nuls.contract.util.ContractUtil;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.protocol.storage.po.BlockHeaderPo;
```

```
/**
 * @author: Niels Wang
 */
public class PoConvertUtil {


    public static BlockHeader fromBlockHeaderPo(BlockHeaderPo po) {
        BlockHeader header = new BlockHeader();
        header.setHash(po.getHash());
        header.setHeight(po.getHeight());
        header.setExtend(po.getExtend());
        header.setPreHash(po.getPreHash());
        header.setTime(po.getTime());
        header.setMerkleHash(po.getMerkleHash());
        header.setTxCount(po.getTxCount());
        header.setBlockSignature(po.getScriptSign());
        //pierre add contract stateRoot
        header.setStateRoot(po.getStateRoot());
        return header;
    }


    public static BlockHeaderPo toBlockHeaderPo(Block block) {
        BlockHeaderPo po = new BlockHeaderPo();
        po.setHash(block.getHeader().getHash());
        po.setPreHash(block.getHeader().getPreHash());
        po.setMerkleHash(block.getHeader().getMerkleHash());
        po.setTime(block.getHeader().getTime());
        po.setHeight(block.getHeader().getHeight());
        po.setTxCount(block.getHeader().getTxCount());
        po.setPackingAddress(block.getHeader().getPackingAddress());
        po.setScriptSign(block.getHeader().getBlockSignature());
        po.setExtend(block.getHeader().getExtend());
        po.setTxHashList(block.getTxHashList());
        po.setStateRoot(ContractUtil.getStateRoot(block.getHeader()));
        return po;
    }
}

127:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\utils\xml\XmlLoader.java
 */
```

```
package io.nuls.protocol.base.utils.xml;

import io.nuls.core.tools.cfg.ConfigLoader;
import io.nuls.core.tools.log.Log;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import java.io.IOException;
import java.io.InputStream;

/**
 * @author: Charlie
 * @date: 2018/8/14
 */
public class XmlLoader {

    public static void loadXml(String xmlName, DefaultHandler handler) throws SAXException {
        try {
            SAXParser saxParser = SAXParserFactory.newInstance().newSAXParser();
            InputStream inputStream =
ConfigLoader.class.getClassLoader().getResourceAsStream(xmlName);
            saxParser.parse(inputStream, handler);
        } catch (ParserConfigurationException e) {
            Log.error(e);
        } catch (IOException e) {
            Log.error(e);
        }
    }
}

128:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\validator\TransferValidator.java
 */

package io.nuls.protocol.base.validator;

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.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 io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.tx.TransferTransaction;

import java.util.Set;

/**
 * @author: Niels Wang
 * @date: 2018/7/5
 */
@Component
public class TransferValidator implements NulsDataValidator<TransferTransaction> {

    @Override
    public ValidateResult validate(TransferTransaction tx) throws NulsException {
        Set<String> addressSet = SignatureUtil.getAddressFromTX(tx);
        for (Coin coin : tx.getCoinData().getTo()) {
            byte[] owner = coin.getOwner();
            if (owner.length > 23) {
                owner = coin.getAddress();
            }
            // Keep the change maybe a very small coin
            if (addressSet.contains(AddressTool.getStringAddressByBytes(owner))) {
                // When the receiver sign this tx,Allow it transfer small coin
                continue;
            }

            if (coin.getNa().isLessThan(ProtocolConstant.MININUM_TRANSFER_AMOUNT)) {
                return ValidateResult.getFailedResult(this.getClass().getSimpleName(),
TransactionErrorCode.TOO_SMALL_AMOUNT);
            }
        }
        return ValidateResult.getSuccessResult();
    }
}

129:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
```

base\src\main\java\io\nuls\protocol\base\validator\TxEffectiveValidator.java

```
package io.nuls.protocol.base.validator;

import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Transaction;

import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.base.version.NulsVersionManager;
import io.nuls.protocol.base.version.ProtocolContainer;

/**
 * @author Niels
 */
@Component
public class TxEffectiveValidator implements NulsDataValidator<Transaction> {
    @Override
    public ValidateResult validate(Transaction tx) {
        int txType = tx.getType();
        ProtocolContainer container = NulsVersionManager.getCurrentProtocolContainer();
        if(!container.containsTxType(txType)) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_NOT_EFFECTIVE);
        }
        return ValidateResult.getSuccessResult();
    }
}
```

130:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\version\NulsVersionHandler.java

```
 */
package io.nuls.protocol.base.version;

import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Transaction;
import io.nuls.protocol.base.utils.xml.XmlLoader;
import io.nuls.protocol.message.base.BaseMessage;
import org.xml.sax.Attributes;
```

```
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

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

/**
 * @author: Charlie
 * @date: 2018/8/15
 */
public class NulsVersionHandler extends DefaultHandler {

    /**
     *
     */
    private static final String INCLUD = "include";
    /**
     *
     */
    private static final String PROTOCOL = "protocol";
    /**
     *  class
     */
    private static final String INCLUD_SRC = "src";
    /**
     *
     */
    private static final String PROTOCOL_VERSION = "version";
    /**
     *
     */
    private static final String PROTOCOL_PERCENT = "percent";
    /**
     *
     */
    private static final String PROTOCOL_DELAY = "delay";
    /**
     *
     */
    private static final String PROTOCOL_BLOCK = "block";
    /**
     *
```

```
     */
    private static final String PROTOCOL_EXTEND = "extend";
    /**
     *  Tx
     */
    private static final String PROTOCOL_TX = "tx";
    /**
     *  Tx
     */
    private static final String PROTOCOL_TX_DISCARD = "tx-discard";
    /**
     *  msg
     */
    private static final String PROTOCOL_MSG = "message";
    /**
     *  msg
     */
    private static final String PROTOCOL_MSG_DISCARD = "message-discard";
    /**
     *  id
     */
    private static final String REF = "ref";

    /**
     * ProtocolProtocol
     */
    private ProtocolContainer protocolContainer;

    /**
     * ProtocolversionProtocol
     */
    private Integer extendTS = null;

    private Map<Integer, Class<? extends Transaction>> discardsTx = null;
    private Map<String, Class<? extends BaseMessage>> discardsMsg = null;


    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes)
throws SAXException {
        super.startElement(uri, localName, qName, attributes);
        //
```

```
        if(INCLUD.equals(qName)){
            String xmlName = attributes.getValue(INCLUD_SRC);
            XmlLoader.loadXml(xmlName, new ProtocolVersionHandler());
        }

        //
        if(PROTOCOL.equals(qName)) {
            protocolContainer = new ProtocolContainer();
            String version = attributes.getValue(PROTOCOL_VERSION);
            String percent = attributes.getValue(PROTOCOL_PERCENT);
            String delay = attributes.getValue(PROTOCOL_DELAY);
            if(!StringUtils.isNumeric(version) || !StringUtils.isNumeric(percent)
||!StringUtils.isNumeric(delay)){
                Log.error(KernelErrorCode.CONFIG_ERROR.getMsg());
                throw new SAXException();
            }
            protocolContainer.setVersion(Integer.parseInt(version.trim()));
            protocolContainer.setPercent(Integer.parseInt(percent.trim()));
            protocolContainer.setDelay(Integer.parseInt(delay.trim()));

            String extend = attributes.getValue(PROTOCOL_EXTEND);
            if(StringUtils.isNotBlank(extend) && !StringUtils.isNumeric(percent)){
                Log.error(KernelErrorCode.CONFIG_ERROR.getMsg());
                throw new SAXException();
            }
            extendTS = StringUtils.isBlank(extend) ? null : Integer.parseInt(extend.trim());

            String block = attributes.getValue(PROTOCOL_BLOCK);

            if(StringUtils.isNotBlank(block)) {
                Class blockClass = null;
                try {
                    blockClass = Class.forName(block.trim());
                    protocolContainer.setBlockClass(blockClass);
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                    throw new SAXException();
                }
            }else{
                if(null != extendTS){
                    ProtocolContainer parentPC = NulsVersionManager.getProtocolContainer(extendTS);
                    if(null == parentPC || null == parentPC.getBlockClass()){
```

```
                throw new SAXException();
            }
            protocolContainer.setBlockClass(parentPC.getBlockClass());
        }else{
            Log.error(KernelErrorCode.CONFIG_ERROR.getMsg());
            throw new SAXException();
        }
    }

    discardsTx = new HashMap<>();
    discardsMsg = new HashMap<>();
}

//Tx
if(PROTOCOL_TX_DISCARD.equals(qName)){
    String discard = attributes.getValue(REF);
    if(!NulsVersionManager.containsTxId(discard)){
        throw new SAXException(discard);
    }
    Class txCLass = NulsVersionManager.getTxProtocol(discard.trim());
    Transaction tx = null;
    try {
        tx = (Transaction) txCLass.newInstance();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    }
    discardsTx.put(tx.getType(), txCLass);
}

//Tx
if(PROTOCOL_TX.equals(qName)){
    String txId = attributes.getValue(REF);
    if(!NulsVersionManager.containsTxId(txId)){
        throw new SAXException(txId);
    }
    Class txCLass = NulsVersionManager.getTxProtocol(txId);
    Transaction tx = null;
    try {
        tx = (Transaction) txCLass.newInstance();
    } catch (InstantiationException e) {
```

```
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
        protocolContainer.putTransaction(tx.getType(), txCLass);
    }

    //msg
    if(PROTOCOL_MSG_DISCARD.equals(qName)){
        String msgId = attributes.getValue(REF);
        if(!NulsVersionManager.containsMessageId(msgId)){
            throw new SAXException(msgId);
        }
        Class txCLass = NulsVersionManager.getMessageProtocol(msgId);
        BaseMessage msg = null;
        try {
            msg = (BaseMessage) txCLass.newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
        String moduleId = String.valueOf(msg.getHeader().getModuleId());
        String type = String.valueOf(msg.getHeader().getMsgType());
        String key = moduleId + "-" + type;
        discardsMsg.put(key, txCLass);
    }

    //msg
    if(PROTOCOL_MSG.equals(qName)){
        String msgId = attributes.getValue(REF);
        if(!NulsVersionManager.containsMessageId(msgId)){
            throw new SAXException(msgId);
        }
        Class txCLass = NulsVersionManager.getMessageProtocol(msgId);
        BaseMessage msg = null;
        try {
            msg = (BaseMessage) txCLass.newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
```

```
            }
            String moduleId = String.valueOf(msg.getHeader().getModuleId());
            String type = String.valueOf(msg.getHeader().getMsgType());
            String key = moduleId + "-" + type;
            protocolContainer.putMessage(key, txCLass);
        }
    }

    @Override
    public void endElement(String uri, String localName, String qName) throws SAXException {

        //
        if(PROTOCOL.equals(qName)) {
            if(null != extendTS) {
                ProtocolContainer parentPC = NulsVersionManager.getProtocolContainer(extendTS);
                if(null == parentPC){
                    throw new SAXException();
                }

                Map<Integer, Class<? extends Transaction>> parentTxMap = parentPC.getTxMap();
                for (Map.Entry<Integer, Class<? extends Transaction>> entry : parentTxMap.entrySet())
{
                    //Tx
                    if(!discardsTx.containsKey(entry.getKey())){
                        protocolContainer.putTransaction(entry.getKey(), entry.getValue());
                    }
                }

                Map<String, Class<? extends BaseMessage>> messageMap =
parentPC.getMessageMap();
                for(Map.Entry<String, Class<? extends BaseMessage>> entry :
messageMap.entrySet()){
                    if(!discardsMsg.containsKey(entry.getKey())){
                        protocolContainer.putMessage(entry.getKey(), entry.getValue());
                    }
                }
            }
            //
            if(protocolContainer.getVersion() > NulsContext.CURRENT_PROTOCOL_VERSION) {
                NulsContext.CURRENT_PROTOCOL_VERSION = protocolContainer.getVersion();
            }
            NulsVersionManager.putContainerMap(protocolContainer.getVersion(),
```

```
protocolContainer);

            extendTS = null;
            discardsTx = null;
            discardsMsg = null;
            protocolContainer = null;
        }
    }

}
```

131:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-base\src\main\java\io\nuls\protocol\base\version\NulsVersionManager.java

```
 */
package io.nuls.protocol.base.version;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Transaction;
import io.nuls.protocol.base.utils.xml.XmlLoader;
import io.nuls.protocol.message.base.BaseMessage;
import io.nuls.protocol.storage.po.ProtocolInfoPo;
import io.nuls.protocol.storage.po.ProtocolTempInfoPo;
import io.nuls.protocol.storage.service.VersionManagerStorageService;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class NulsVersionManager {

    /**
     *
     */
    private static Map<String, Class<? extends BaseMessage>> messageProtocolMap = new ConcurrentHashMap<>();
    /**
     *
     */
    private static Map<String, Class<? extends Transaction>> txProtocolMap = new ConcurrentHashMap<>();
```

```
    /**
     *
     */
    private static Map<Integer, ProtocolContainer> containerMap = new ConcurrentHashMap<>();

    private static VersionManagerStorageService versionManagerStorageService;

    private static Map<String, Integer> consensusVersionMap = new ConcurrentHashMap<>();

//    public static void test() {
//        for (Map.Entry<String, Class<? extends Transaction>> entry : txProtocolMap.entrySet()) {
//            System.out.println(entry.getValue());
//        }
//        System.out.println();
//        for (Map.Entry<String, Class<? extends BaseMessage>> entry :
messageProtocolMap.entrySet()) {
//            System.out.println(entry.getValue());
//        }
//        for (Map.Entry<Integer, ProtocolContainer> entry : containerMap.entrySet()) {
//            System.out.println("----- ProtocolContainer -----");
//            try {
//                System.out.println(JSONUtils.obj2PrettyJson(entry));
//            } catch (Exception e) {
//                e.printStackTrace();
//            }
//            System.out.println("----------------------------");
//        }
//    }

    public static void init() throws Exception {
        loadConfig();
        VersionManagerStorageService vmss =
NulsContext.getServiceBean(VersionManagerStorageService.class);
        Integer mainVersion = vmss.getMainVersion();
        if (mainVersion != null) {
            NulsContext.MAIN_NET_VERSION = mainVersion;
        }
        NulsContext.CHANGE_HASH_SERIALIZE_HEIGHT =
vmss.getChangeTxHashBlockHeight();
        ProtocolContainer container = getProtocolContainer(1);
        if (container != null) {
            container.setStatus(ProtocolContainer.VALID);
```

```
        }
    }

    /***
     *
     */
    public static void loadVersion() {
        //
        VersionManagerStorageService vmss =
NulsContext.getServiceBean(VersionManagerStorageService.class);
        Map<String, Integer> versionMap = vmss.getConsensusVersionMap();
        if (versionMap != null) {
            consensusVersionMap = versionMap;
        }
        checkHasLaterVersion();
        //
        for (ProtocolContainer protocolContainer : containerMap.values()) {
            ProtocolInfoPo protocolInfoPo = vmss.getProtocolInfoPo(protocolContainer.getVersion());
            if (protocolContainer.getVersion() == 1) {
                protocolContainer.setStatus(ProtocolContainer.VALID);
                protocolContainer.setEffectiveHeight(0L);
                protocolContainer.setCurrentDelay(0L);
                protocolContainer.setCurrentPercent(100);
                protocolContainer.setRoundIndex(0);
            } else if (protocolInfoPo != null) {
                protocolContainer.setCurrentDelay(protocolInfoPo.getCurrentDelay());
                protocolContainer.setStatus(protocolInfoPo.getStatus());
                protocolContainer.setAddressSet(protocolInfoPo.getAddressSet());
                protocolContainer.setEffectiveHeight(protocolInfoPo.getEffectiveHeight());
                protocolContainer.setCurrentPercent(protocolInfoPo.getCurrentPercent());
                protocolContainer.setRoundIndex(protocolInfoPo.getRoundIndex());
                protocolContainer.setPrePercent(protocolInfoPo.getPrePercent());
            }
            //container
            ProtocolTempInfoPo tempInfoPo =
getVersionManagerStorageService().getProtocolTempInfoPo(protocolContainer.getProtocolKey());
            if (tempInfoPo != null) {
                protocolContainer.setRoundIndex(tempInfoPo.getRoundIndex());
                protocolContainer.setCurrentDelay(tempInfoPo.getCurrentDelay());
                protocolContainer.setAddressSet(tempInfoPo.getAddressSet());
                protocolContainer.setStatus(tempInfoPo.getStatus());
                protocolContainer.setEffectiveHeight(tempInfoPo.getEffectiveHeight());
```

```
            protocolContainer.setCurrentPercent(tempInfoPo.getCurrentPercent());
            protocolContainer.setPrePercent(tempInfoPo.getPrePercent());
            protocolInfoPo = new ProtocolInfoPo(tempInfoPo);
            getVersionManagerStorageService().saveProtocolInfoPo(protocolInfoPo);
getVersionManagerStorageService().removeProtocolTempInfo(tempInfoPo.getProtocolKey());
    }

          //
          if (protocolContainer.getStatus() == ProtocolContainer.VALID) {
             if (NulsContext.MAIN_NET_VERSION < protocolContainer.getVersion()) {
                NulsContext.MAIN_NET_VERSION = protocolContainer.getVersion();
getVersionManagerStorageService().saveMainVersion(NulsContext.MAIN_NET_VERSION);
         //2hash
                if (protocolContainer.getVersion() == 2) {
getVersionManagerStorageService().saveChangeTxHashBlockHeight(protocolContainer.getEffecti
veHeight());
                   NulsContext.CHANGE_HASH_SERIALIZE_HEIGHT =
protocolContainer.getEffectiveHeight();
                }
             }
          }
       }
    }

    /**
     *
     *
     * @throws Exception
     */
    public static void loadConfig() throws Exception {
       XmlLoader.loadXml(NulsConstant.NULS_VERSION_XML, new NulsVersionHandler());
    }

    /**
     *
     */
    private static void checkHasLaterVersion() {
       Map<String, ProtocolTempInfoPo> protocolTempMap =
getVersionManagerStorageService().getProtocolTempMap();
       for (ProtocolTempInfoPo tempInfoPo : protocolTempMap.values()) {
          if (tempInfoPo.getVersion() > NulsContext.CURRENT_PROTOCOL_VERSION) {
             if (tempInfoPo.getStatus() == ProtocolContainer.VALID) {
```

```
                //linux
                //NulsContext.mastUpGrade = true
                if (System.getProperties().getProperty("os.name").toUpperCase().indexOf("LINUX") !=
-1) {
                    Log.error("The version is too low to upgrade");
                    NulsContext.getInstance().exit(1);
                    return;
                } else {
                    NulsContext.mastUpGrade = true;
                    return;
                }
            }
        }
    }
}

/**
 *
 *
 * @return ProtocolContainer
 */
public static ProtocolContainer getCurrentProtocolContainer() {
    return containerMap.get(NulsContext.CURRENT_PROTOCOL_VERSION);
}

/**
 *
 *
 * @return ProtocolContainer
 */
public static ProtocolContainer getMainProtocolContainer() {
    return containerMap.get(NulsContext.MAIN_NET_VERSION);
}

public static Map<Integer, ProtocolContainer> getAllProtocolContainers() {
    return containerMap;
}

/**
 *
 *
 * @param version
```

```
 * @return ProtocolContainer
 */
public static ProtocolContainer getProtocolContainer(int version) {
    return containerMap.get(version);
}

/**
 * idclass
 *
 * @param id id
 * @return Transaction class
 */
public static Class<? extends Transaction> getTxProtocol(String id) {
    return txProtocolMap.get(id);
}

/**
 * idclass
 *
 * @param id id
 * @return Message class
 */
public static Class<? extends BaseMessage> getMessageProtocol(String id) {
    return messageProtocolMap.get(id);
}

/**
 * mapid
 *
 * @param id id
 * @return boolean
 */
public static boolean containsTxId(String id) {
    return txProtocolMap.containsKey(id);
}

/**
 * mapid
 *
 * @param id id
 * @return boolean
 */
```

```
    public static boolean containsMessageId(String id) {
        return messageProtocolMap.containsKey(id);
    }

    /**
     * map
     *
     * @param version
     * @return boolean
     */
    public static boolean containsProtocolVersion(Integer version) {
        return containerMap.containsKey(version);
    }

    /**
     *
     *
     * @param id      id
     * @param txClass class
     * @throws NulsException nuls
     */
    public static void putTxProtocol(String id, Class<? extends Transaction> txClass) {
        if (containsTxId(id) || null == txClass) {
            throw new RuntimeException();
        }
        txProtocolMap.put(id, txClass);
    }

    /**
     *
     *
     * @param id           id
     * @param messageClass
     * @throws NulsException
     */
    public static void putMessageProtocol(String id, Class<? extends BaseMessage>
messageClass) {
        if (containsMessageId(id) || null == messageClass) {
            throw new RuntimeException();
        }
        messageProtocolMap.put(id, messageClass);
    }
```

```
    /**
     *
     *
     * @param key               id
     * @param protocolContainer
     */
    public static void putContainerMap(Integer key, ProtocolContainer protocolContainer) {
        if (null == key || null == protocolContainer) {
            throw new RuntimeException();
        }
        containerMap.put(key, protocolContainer);
    }

    public static Integer getMainVersion() {
        return NulsContext.MAIN_NET_VERSION;
    }

    public static Integer getCurrentVersion() {
        return NulsContext.CURRENT_PROTOCOL_VERSION;
    }

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

    public static Map<String, Integer> getConsensusVersionMap() {
        return consensusVersionMap;
    }

    public static void setConsensusVersionMap(Map<String, Integer> consensusVersionMap) {
        NulsVersionManager.consensusVersionMap = consensusVersionMap;
    }
}

132:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\version\ProtocolContainer.java
 */
```

```
package io.nuls.protocol.base.version;

import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.message.base.BaseMessage;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class ProtocolContainer<T extends BaseNulsData> {
    /**
     *
     */
    private Integer version;
    /**
     *
     */
    private Class<T> blockClass;
    /****/
    private int percent;
    /****/
    private long roundIndex;
    /****/
    private long delay;
    /*** */
    private long currentDelay;
    /**()*/
    private int currentPercent;
    /****/
    private int prePercent;
    /***/
    private Set<String> addressSet;
    /**  */
    private Long effectiveHeight;
    /**
     *
     */
    private int status;
    /**
```

```
 *
 */
public static final int INVALID = 0;
/**
 *
 */
public static final int DELAY_LOCK = 1;
/**
 *
 */
public static final int VALID = 2;


public ProtocolContainer() {
    addressSet = new HashSet<>();
}

/**
 * Transactionmapkey
 */
private Map<Integer, Class<? extends Transaction>> txMap = new HashMap<>();

/**
 * MessagemapkeymoduleId + messageTypekey "3-1"
 */
private Map<String, Class<? extends BaseMessage>> messageMap = new HashMap<>();


public void putTransaction(int type, Class<? extends Transaction> clazz) {
    this.txMap.put(type, clazz);
}

public void putMessage(String key, Class<? extends BaseMessage> clazz) {
    this.messageMap.put(key, clazz);
}

public Map<Integer, Class<? extends Transaction>> getTxMap() {
    return txMap;
}

public Map<String, Class<? extends BaseMessage>> getMessageMap() {
    return messageMap;
```

```
}

public Transaction getTransaction(NulsByteBuffer byteBuffer) {
    return null;
}

public BaseMessage getMessage(NulsByteBuffer byteBuffer) {
    return null;
}

public boolean containsTxType(int type) {
    return txMap.containsKey(type);
}

public boolean containsMessageType(int moduleId, int type) {
    return messageMap.containsKey(moduleId + "-" + type);
}

public Integer getVersion() {
    return version;
}

public void setVersion(Integer version) {
    this.version = version;
}

public long getDelay() {
    return delay;
}

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

public Class<T> getBlockClass() {
    return blockClass;
}

public void setBlockClass(Class<T> blockClass) {
    this.blockClass = blockClass;
}
```

```
public int getPercent() {
    return percent;
}

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

public Set<String> getAddressSet() {
    return addressSet;
}

public void setAddressSet(Set<String> addressSet) {
    this.addressSet = addressSet;
}

public int getStatus() {
    return status;
}

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

public long getCurrentDelay() {
    return currentDelay;
}

public void setCurrentDelay(long currentDelay) {
    this.currentDelay = currentDelay;
}

public long getRoundIndex() {
    return roundIndex;
}

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

public Long getEffectiveHeight() {
    return effectiveHeight;
```

```
    }

    public void setEffectiveHeight(Long effectiveHeight) {
        this.effectiveHeight = effectiveHeight;
    }

    public String getProtocolKey() {
        return version + "-" + percent + "-" + delay;
    }

    public void reset() {
        this.currentDelay = 0;
        this.roundIndex = 0;
        this.status = ProtocolContainer.INVALID;
        this.effectiveHeight = null;
        this.addressSet.clear();
    }

    public int getCurrentPercent() {
        return currentPercent;
    }

    public void setCurrentPercent(int currentPercent) {
        this.currentPercent = currentPercent;
    }

    public int getPrePercent() {
        return prePercent;
    }

    public void setPrePercent(int prePercent) {
        this.prePercent = prePercent;
    }

}

133:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\main\java\io\nuls\protocol\base\version\ProtocolVersionHandler.java
 */
package io.nuls.protocol.base.version;

import io.nuls.core.tools.log.Log;
```

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

/**
 * @author: Charlie
 * @date: 2018/8/15
 */
public class ProtocolVersionHandler extends DefaultHandler {

    private static final String TX_PROTOCOL = "tx";
    private static final String MESSAGE_PROTOCOL = "message";
    private static final String PROTOCOL_ID = "id";
    private static final String PROTOCOL_CLASS = "class";

    @Override
    public void startElement(String uri, String localName, String qName, Attributes attributes)
throws SAXException {
        if(TX_PROTOCOL.equals(qName)){
            String id = attributes.getValue(PROTOCOL_ID);
            String className = attributes.getValue(PROTOCOL_CLASS);
            try {
                Class txClass = Class.forName(className);
                NulsVersionManager.putTxProtocol(id, txClass);
            } catch (ClassNotFoundException e) {
                Log.error(e);
            }
        }
        if(MESSAGE_PROTOCOL.equals(qName)){
            String id = attributes.getValue(PROTOCOL_ID);
            String className = attributes.getValue(PROTOCOL_CLASS);
            try {
                Class messageClass = Class.forName(className);
                NulsVersionManager.putMessageProtocol(id, messageClass);
            } catch (ClassNotFoundException e) {
                Log.error(e);
            }
        }
    }
}

134:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
```

```
base\src\test\java\io\nuls\protocol\base\service\BlockServiceImplTest.java
 */

package io.nuls.protocol.base.service;

import io.nuls.db.module.impl.LevelDbModuleBootstrap;
import io.nuls.kernel.MicroKernelBootstrap;
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.script.BlockSignature;
import io.nuls.ledger.module.impl.UtxoLedgerModuleBootstrap;
import io.nuls.protocol.service.BlockService;
import org.junit.Before;
import org.junit.Test;

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

import static org.junit.Assert.assertTrue;

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

    private BlockService service;

    @Before
    public void init() {
        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();

        LevelDbModuleBootstrap bootstrap = new LevelDbModuleBootstrap();
        bootstrap.init();
        bootstrap.start();

        UtxoLedgerModuleBootstrap ledgerModuleBootstrap = new UtxoLedgerModuleBootstrap();
        ledgerModuleBootstrap.init();
```

```
        ledgerModuleBootstrap.start();


        service = NulsContext.getServiceBean(BlockService.class);
        Block block = new Block();
        BlockHeader blockHeader = new BlockHeader();
        blockHeader.setHash(NulsDigestData.calcDigestData("hashhash".getBytes()));
        blockHeader.setHeight(1286L);
        blockHeader.setExtend("extends".getBytes());
        blockHeader.setMerkleHash(NulsDigestData.calcDigestData("merkleHash".getBytes()));
        blockHeader.setPreHash(NulsDigestData.calcDigestData("prehash".getBytes()));
        try {
            blockHeader.setPackingAddress("address".getBytes());
        } catch (Exception e) {
            e.printStackTrace();
            assertTrue(false);
        }
        blockHeader.setBlockSignature(new BlockSignature());
        blockHeader.setTime(12345678901L);
        blockHeader.setTxCount(3);
        List<NulsDigestData> txHashList = new ArrayList<>();
        txHashList.add(NulsDigestData.calcDigestData("first-tx-hash".getBytes()));
        txHashList.add(NulsDigestData.calcDigestData("second-tx-hash".getBytes()));
        txHashList.add(NulsDigestData.calcDigestData("third-tx-hash".getBytes()));
//        block.setTxHashList(txHashList);
//        this.model = blockHeader;
    }

    @Test
    public void test() {

    }
//
//    @Test
//    public void getGengsisBlock() {
//    }
//
//    @Test
//    public void getBestBlock() {
//    }
//
//    @Test
```

```
//    public void getBestBlockHeader() {
//    }
//
//    @Test
//    public void getBlockHeader() {
//    }
//
//    @Test
//    public void getBlockHeader1() {
//    }
//
//    @Test
//    public void getBlock() {
//    }
//
//    @Test
//    public void getBlock1() {
//    }
//
//    @Test
//    public void saveBlock() {
//    }
//
//    @Test
//    public void rollbackBlock() {
//    }
//
//    @Test
//    public void forwardBlock() {
//    }
//
//    @Test
//    public void broadcastBlock() {
//    }
}

135:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\test\java\io\nuls\protocol\base\service\TransactionServiceImplTest.java
 */

package io.nuls.protocol.base.service;
```

```
import io.nuls.consensus.poc.protocol.entity.*;
import io.nuls.consensus.poc.protocol.tx.*;
import io.nuls.core.tools.crypto.ECKey;
import io.nuls.core.tools.log.Log;
import io.nuls.db.module.impl.LevelDbModuleBootstrap;
import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.core.SpringLiteContext;
import io.nuls.kernel.model.*;

import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.ledger.module.impl.UtxoLedgerModuleBootstrap;
import io.nuls.network.module.impl.NettyNetworkModuleBootstrap;
import io.nuls.protocol.model.tx.CoinBaseTransaction;
import io.nuls.protocol.model.tx.TransferTransaction;
import io.nuls.protocol.service.TransactionService;
import org.junit.Before;
import org.junit.Test;

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

import static org.junit.Assert.*;

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

    private List<Transaction> allList;
    private List<Transaction> txList;

    private TransactionService transactionService;


    @Before
    public void init() {
```

```
        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();

        LevelDbModuleBootstrap bootstrap = new LevelDbModuleBootstrap();
        bootstrap.init();
        bootstrap.start();

        UtxoLedgerModuleBootstrap ledgerModuleBootstrap = new UtxoLedgerModuleBootstrap();
        ledgerModuleBootstrap.init();
        ledgerModuleBootstrap.start();

        NettyNetworkModuleBootstrap networkModuleBootstrap = new
NettyNetworkModuleBootstrap();
        networkModuleBootstrap.init();
        networkModuleBootstrap.start();

        transactionService = SpringLiteContext.getBean(TransactionService.class);
        initTxList();
    }

    @Test
    public void test() {
        this.conflictDetect();
        this.commitTx();
        this.rollback();
        this.broadcastTx();
        this.forwardTx();
    }

    private void commitTx() {
        for (Transaction tx : txList) {
            Result result = this.transactionService.commitTx(tx, null);
            assertTrue(result.isSuccess());
        }
    }

    private void rollback() {
        for (int i = txList.size() - 1; i >= 0; i--) {
            Transaction tx = txList.get(i);
            Result result = this.transactionService.rollbackTx(tx, null);
            assertTrue(result.isSuccess());
```

```
        }
        Result result = this.transactionService.rollbackTx(txList.get(0), null);
        assertFalse(result.isSuccess());
    }

    private void forwardTx() {
        for (int i = txList.size() - 1; i >= 0; i--) {
            Transaction tx = txList.get(i);
            Result result = this.transactionService.forwardTx(tx, null);
            assertTrue(result.isSuccess());
        }
    }

    private void broadcastTx() {
        for (int i = txList.size() - 1; i >= 0; i--) {
            Transaction tx = txList.get(i);
            Result result = this.transactionService.forwardTx(tx, null);
            assertTrue(result.isSuccess());
        }
    }

    private void conflictDetect() {
        ValidateResult result = transactionService.conflictDetect(allList);
        this.txList = (List<Transaction>) result.getData();
        assertNotNull(txList);
        //
        assertEquals(2, 2);
    }

    private void initTxList() {
        List<Transaction> list = new ArrayList<>();
        ECKey ecKey1 = new ECKey();
        ECKey ecKey2 = new ECKey();
        ECKey ecKey3 = new ECKey();
        ECKey ecKey4 = new ECKey();
        ECKey ecKey5 = new ECKey();
        ECKey ecKey6 = new ECKey();


        Transaction tx = createCoinBaseTransaction(ecKey1, ecKey2, ecKey3, ecKey4, ecKey5,
ecKey6);
        list.add(tx);
```

```
        Transaction yellowPunishTx = createYellowPunishTx(ecKey1, ecKey2, ecKey3, ecKey4,
ecKey5, ecKey6);
        list.add(yellowPunishTx);

//        RedPunishTransaction redPunishTransaction = createRedPunishTx(ecKey1, ecKey4,
ecKey5, ecKey6);
//        list.add(redPunishTransaction);

        TransferTransaction transferTransaction1 = createTransferTransaction(ecKey1, null, ecKey2,
Na.ZERO);
        TransferTransaction transferTransaction2 = createTransferTransaction(ecKey1, null, ecKey3,
Na.ZERO);
        list.add(transferTransaction1);
        list.add(transferTransaction2);

        createSetAliasTransaction(ecKey1, "alias");
//        createSetAliasTransaction(ecKey1, "alias1");
//        createSetAliasTransaction(ecKey2, "alias");

        CreateAgentTransaction tx1 = createRegisterAgentTransaction(ecKey1, ecKey2,
"agentName");
        CreateAgentTransaction tx2 = createRegisterAgentTransaction(ecKey2, ecKey3,
"agentName");
        CreateAgentTransaction tx3 = createRegisterAgentTransaction(ecKey4, ecKey5,
"agentName2");
        CreateAgentTransaction tx4 = createRegisterAgentTransaction(ecKey1, ecKey3,
"agentName3");
        list.add(tx1);
        list.add(tx2);
        list.add(tx3);
        list.add(tx4);

        DepositTransaction join1 = createDepositTransaction(ecKey1, tx1.getHash(),
Na.parseNuls(200000));
        DepositTransaction join2 = createDepositTransaction(ecKey1, tx2.getHash(),
Na.parseNuls(200000));
        DepositTransaction join3 = createDepositTransaction(ecKey1, tx3.getHash(),
Na.parseNuls(200000));
        DepositTransaction join4 = createDepositTransaction(ecKey1, tx4.getHash(),
Na.parseNuls(200000));
        DepositTransaction join5 = createDepositTransaction(ecKey1, tx3.getHash(),
```

```
Na.parseNuls(200000));
        DepositTransaction join6 = createDepositTransaction(ecKey1, tx3.getHash(),
Na.parseNuls(200000));
        DepositTransaction join7 = createDepositTransaction(ecKey1, tx3.getHash(),
Na.parseNuls(200000));
        list.add(join1);
        list.add(join3);
        list.add(join2);
        list.add(join4);
        list.add(join5);
        list.add(join6);
        list.add(join7);

        try {
            createCancelDepositTransaction(ecKey1, NulsDigestData.fromDigestHex("txHash"));
        } catch (NulsException e) {
            Log.error(e);
        }

        StopAgentTransaction stop1 = createStopAgentTransaction(ecKey1, tx1.getHash());
        StopAgentTransaction stop2 = createStopAgentTransaction(ecKey1, tx2.getHash());
        StopAgentTransaction stop3 = createStopAgentTransaction(ecKey4, tx3.getHash());
        StopAgentTransaction stop4 = createStopAgentTransaction(ecKey1, tx4.getHash());
        list.add(stop1);
        list.add(stop2);
        list.add(stop3);
        list.add(stop4);
        this.allList = list;
    }

//    private RedPunishTransaction createRedPunishTx(ECKey ecKey, ECKey... ecKeys) {
//        RedPunishTransaction tx = new RedPunishTransaction();
//        setCommonFields(tx);
//        RedPunishData data = new RedPunishData();
//        data.setAddress(AddressTool.getAddress(ecKeys[0].getPubKey()));
//        data.setEvidence("for test".getBytes());
//        data.setReasonCode(PunishReasonEnum.BIFURCATION.getCode());
//        tx.setTxData(data);
//        return tx;
//    }

    private YellowPunishTransaction createYellowPunishTx(ECKey ecKey, ECKey... ecKeys) {
```

```
        YellowPunishTransaction tx = new YellowPunishTransaction();
        setCommonFields(tx);
        YellowPunishData data = new YellowPunishData();
        List<byte[]> addressList = new ArrayList<>();
        for (ECKey ecKey1 : ecKeys) {
            addressList.add(AddressTool.getAddress(ecKey1.getPubKey()));
        }
        data.setAddressList(addressList);
        tx.setTxData(data);
        return tx;
    }

    private CancelDepositTransaction createCancelDepositTransaction(ECKey ecKey,
NulsDigestData txHash) {
        CancelDepositTransaction tx = new CancelDepositTransaction();
        setCommonFields(tx);
        CancelDeposit cd = new CancelDeposit();
        cd.setAddress(AddressTool.getAddress(ecKey.getPubKey()));
        cd.setJoinTxHash(txHash);
        tx.setTxData(cd);
        signTransaction(tx, ecKey);
        return tx;
    }

    private StopAgentTransaction createStopAgentTransaction(ECKey ecKey, NulsDigestData
agentTxHash) {
        StopAgentTransaction tx = new StopAgentTransaction();
        setCommonFields(tx);
        StopAgent txData = new StopAgent();
        txData.setAddress(AddressTool.getAddress(ecKey.getPubKey()));
        txData.setCreateTxHash(agentTxHash);
        tx.setTxData(txData);
        signTransaction(tx, ecKey);
        return tx;

    }

    private DepositTransaction createDepositTransaction(ECKey ecKey, NulsDigestData
agentTxHash, Na na) {
        DepositTransaction tx = new DepositTransaction();
        setCommonFields(tx);
        Deposit deposit = new Deposit();
```

```
        deposit.setDelHeight(0L);
        deposit.setBlockHeight(1);
        deposit.setTime(System.currentTimeMillis());
        deposit.setAddress(AddressTool.getAddress(ecKey.getPubKey()));
        deposit.setAgentHash(agentTxHash);
        deposit.setDeposit(na);
        tx.setTxData(deposit);
        signTransaction(tx, ecKey);
        return tx;
    }

    private CreateAgentTransaction createRegisterAgentTransaction(ECKey ecKey1, ECKey
ecKey2, String agentName) {
        CreateAgentTransaction tx = new CreateAgentTransaction();
        setCommonFields(tx);
        Agent agent = new Agent();
        agent.setBlockHeight(1);
        agent.setDelHeight(0);
        agent.setTime(System.currentTimeMillis());
        agent.setAgentAddress(AddressTool.getAddress(ecKey1.getPubKey()));
        agent.setCommissionRate(10);
        agent.setDeposit(Na.parseNuls(20000));
        agent.setPackingAddress(AddressTool.getAddress(ecKey2.getPubKey()));
        agent.setRewardAddress(agent.getAgentAddress());
        tx.setTxData(agent);
        signTransaction(tx, ecKey1);
        return tx;
    }

    private Transaction createSetAliasTransaction(ECKey ecKey, String alias) {
        return null;
    }

    private TransferTransaction createTransferTransaction(ECKey ecKey1, byte[] coinKey, ECKey
ecKey2, Na na) {
        TransferTransaction tx = new TransferTransaction();
        setCommonFields(tx);
        CoinData coinData = new CoinData();
        List<Coin> fromList = new ArrayList<>();
        fromList.add(new Coin(coinKey, Na.parseNuls(10001), 0));
        coinData.setFrom(fromList);
        List<Coin> toList = new ArrayList<>();
```

```
        toList.add(new Coin(AddressTool.getAddress(ecKey2.getPubKey()), Na.parseNuls(10000),
1000));
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        signTransaction(tx, ecKey1);
        return tx;
    }

    private CoinBaseTransaction createCoinBaseTransaction(ECKey ecKey, ECKey... ecKeys) {
        CoinBaseTransaction tx = new CoinBaseTransaction();
        setCommonFields(tx);
        CoinData coinData = new CoinData();
        List<Coin> toList = new ArrayList<>();
        toList.add(new Coin(AddressTool.getAddress(ecKey.getPubKey()), Na.parseNuls(10000),
1000));
        toList.add(new Coin(AddressTool.getAddress(ecKey.getPubKey()), Na.parseNuls(10000),
0));
        for (ECKey ecKey1 : ecKeys) {
            Coin coin = new Coin(AddressTool.getAddress(ecKey1.getPubKey()),
Na.parseNuls(10000), 0);
            toList.add(coin);
        }
        coinData.setTo(toList);
        tx.setCoinData(coinData);
        signTransaction(tx, ecKey);
        return tx;
    }

    private void setCommonFields(Transaction tx) {
        tx.setTime(System.currentTimeMillis());
        tx.setBlockHeight(1);
        tx.setRemark("for test".getBytes());
    }

    private 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);
        }
    }
}

136:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
base\src\test\java\io\nuls\protocol\base\utils\filter\InventoryFilterTest.java
 */

package io.nuls.protocol.base.utils.filter;

import com.google.common.hash.BloomFilter;
import com.google.common.hash.Funnels;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.protocol.model.tx.TransferTransaction;
import org.junit.Test;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * @author: Niels Wang
 * @date: 2018/7/8
 */
public class InventoryFilterTest {

    private AtomicInteger count = new AtomicInteger(0);

    @Test
    public void test1() {
```

```
        BloomFilter<byte[]> filter = BloomFilter.create(Funnels.byteArrayFunnel(), 1000000,
0.00001);
        ArrayList<Transaction> txList = new ArrayList<>();
        for (int i = 1000000; i < 2000000; i++) {
            Transaction tx = new TransferTransaction();
            tx.setTime(i);
tx.setRemark("sdfsdfsdfsdfsdfsdfaaadsfasdfsadfsdfasdfasdfasdfasdfasdfsadfaaaaaaaaaaaaaaaaaa
aaaaabsdsadfsadfsdfsdfsdfsdfsdfsdfsdfaaadsfasdfsadfsdfasdfasdfasdfasdfasdfsadfaaaaaaaaaaa
aaaaaaaaaaabsdsadfsadfsdfsdfsdfsdfsdfsdfsdfaa".getBytes());
            txList.add(tx);
        }
        System.out.println("start....");
        long start = System.currentTimeMillis();
        for (Transaction tx : txList) {
            NulsDigestData hash = tx.getHash();
            if (!filter.mightContain(hash.getDigestBytes())) {
                filter.put(hash.getDigestBytes());
                int num = count.incrementAndGet();
                if (num % 100 == 0) {
                    System.out.println("count::::::" + num);
                }
            }
        }
        System.out.println("use time::" + (System.currentTimeMillis() - start));
        System.out.println(count.get());

    }

    @Test
    public void test() throws IOException {

        BloomFilter<byte[]> filter = BloomFilter.create(Funnels.byteArrayFunnel(), 1000000,
0.00001);
        List<String> list = new ArrayList<>();
        Set<NulsDigestData> set = new HashSet<>();
        ArrayList<Transaction> txList = new ArrayList<>();
        for (int i = 0; i < 1000000; i++) {
            Transaction tx = new TransferTransaction();
            tx.setTime(i);
tx.setRemark("sdfsdfsdfsdfsdfsdfaaadsfasdfsadfsdfasdfasdfasdfasdfasdfsadfaaaaaaaaaaaaaaaaaa
aaaaabsdsadfsadfsdfsdfsdfsdfsdfsdfsdfaaadsfasdfsadfsdfasdfasdfasdfasdfasdfsadfaaaaaaaaaaa
aaaaaaaaaaabsdsadfsadfsdfsdfsdfsdfsdfsdfsdfaa".getBytes());
```

```
            tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
            txList.add(tx);
        }
        for (int i = 0; i < 2; i++) {
            Thread t = new Thread(new Runnable() {
                @Override
                public void run() {
                    for (Transaction tx : txList) {
                        NulsDigestData hash = tx.getHash();
                        if (!filter.mightContain(hash.getDigestBytes())) {
                            filter.put(hash.getDigestBytes());
                            set.add(hash);
                            int num = count.incrementAndGet();
                            if (num % 1000 == 0) {
                                System.out.println("count::::::" + num);
                            }
                        }
                    }
                    list.add("done");
                }
            });
            t.start();
        }
        while (list.size() < 5) {
            try {
                Thread.sleep(1000L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("count====" + count.get());
        System.out.println("real-size====" + set.size());

    }
}

137:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-rpc\src\main\java\io\nuls\protocol\rpc\cmd\GetBestBlockHeaderProcessor.java
 */

package io.nuls.protocol.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.model.CommandResult;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.CommandHelper;
import io.nuls.kernel.utils.RestFulUtils;

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

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription());
        return builder.toString();
    }

    @Override
    public String getCommandDescription() {
        return "getbestblockheader --get the best block header";
    }

    @Override
    public boolean argsValidate(String[] args) {
        int length = args.length;
        if(length > 1) {
            return false;
        }
        return true;
```

```
    }

    @Override
    public CommandResult execute(String[] args) {
        RpcClientResult result = restFul.get("/block/newest/",null);
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> map = (Map) result.getData();
        map.put("reward", CommandHelper.naToNuls(map.get("reward")));
        map.put("fee", CommandHelper.naToNuls(map.get("fee")));
        map.put("time", DateUtil.convertDate(new Date((Long) map.get("time"))));
        map.put("roundStartTime", DateUtil.convertDate(new Date((Long)
map.get("roundStartTime"))));
        result.setData(map);
        return CommandResult.getResult(result);
    }
}

138:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\cmd\GetBlockHeaderListProcessor.java
 */

package io.nuls.protocol.rpc.cmd;

import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.CommandBuilder;
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;

/**
 * RPC
 * @author: Charlie
 */
public class GetBlockHeaderListProcessor implements CommandProcessor {

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

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

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

@Override
public String getCommandDescription() {
    return "getblockheaderlist <pageNumber> <pageSize> --get block header list";
}

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if(length != 3) {
        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("", parameters);
    if (result.isFailed()) {
        return CommandResult.getFailed(result);
```

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

139:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\cmd\GetBlockHeaderProcessor.java
 */

package io.nuls.protocol.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.constant.KernelErrorCode;
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.Map;


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

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

    @Override
    public String getHelp() {
        CommandBuilder builder = new CommandBuilder();
        builder.newLine(getCommandDescription())
                .newLine("\t<hash> | <height> get block header by hash or block height - Required");
```

```
    return builder.toString();
}

@Override
public String getCommandDescription() {
    return "getblockheader <hash> | <height>--get the block header with hash or height";
}

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

@Override
public CommandResult execute(String[] args) {
    String hash = null;
    long height = 0;

    if (StringUtils.isBlank(args[1])) {
        return CommandResult.getFailed(KernelErrorCode.PARAMETER_ERROR.getMsg());
    }

    try {
        height = Long.parseLong(args[1]);
    } catch (Exception e) {
        hash = args[1];
    }

    RpcClientResult result = null;
    if (hash != null) {
        result = restFul.get("/block/header/hash/" + hash, null);
    } else {
        result = restFul.get("/block/header/height/" + height, null);
    }
    if(result.isFailed()){
```

```
            return CommandResult.getFailed(result);
        }
        Map<String, Object> map = (Map) result.getData();
        map.put("reward", CommandHelper.naToNuls(map.get("reward")));
        map.put("fee", CommandHelper.naToNuls(map.get("fee")));
        map.put("time", DateUtil.convertDate(new Date((Long) map.get("time"))));
        map.put("roundStartTime", DateUtil.convertDate(new Date((Long)
map.get("roundStartTime"))));
        result.setData(map);
        return CommandResult.getResult(result);
    }
}

140:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\cmd\GetBlockProcessor.java
 */

package io.nuls.protocol.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.Result;
import io.nuls.kernel.processor.CommandProcessor;
import io.nuls.kernel.utils.RestFulUtils;

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

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

    private RestFulUtils restFul = RestFulUtils.getInstance();

    @Override
```

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

@Override
public String getHelp() {
    CommandBuilder builder = new CommandBuilder();
    builder.newLine(getCommandDescription())
            .newLine("\t<hash> | <height> get block by hash or block height - Required");
    return builder.toString();
}

@Override
public String getCommandDescription() {
    return "getblock <hash> | <height> --get the block with hash or height";
}

@Override
public boolean argsValidate(String[] args) {
    int length = args.length;
    if(length != 2) {
        return false;
    }
    if(!CommandHelper.checkArgsIsNull(args)) {
        return false;
    }
    if(!StringUtils.isNumeric(args[1])){
        if(!NulsDigestData.validHash(args[1])){
            return false;
        }
    }
    return true;
}

@Override
public CommandResult execute(String[] args) {
    String arg = args[1];
    RpcClientResult result = null;
    if(StringUtils.isNumeric(arg)){
        result = restFul.get("/block/height/" + arg, null);
    }else{
        result = restFul.get("/block/hash/" + arg, null);
```

```
        }
        if (result.isFailed()) {
            return CommandResult.getFailed(result);
        }
        Map<String, Object> map = (Map) result.getData();
        map.put("reward", CommandHelper.naToNuls(map.get("reward")));
        map.put("fee", CommandHelper.naToNuls(map.get("fee")));
        map.put("time", DateUtil.convertDate(new Date((Long) map.get("time"))));
        map.put("roundStartTime", DateUtil.convertDate(new Date((Long)
map.get("roundStartTime"))));

        List<Map<String, Object>> txList = (List<Map<String, Object>>)map.get("txList");
        for(Map<String, Object> tx : txList){
            tx.put("type", CommandHelper.txTypeExplain((Integer)tx.get("type")));
            tx.put("value", CommandHelper.naToNuls(tx.get("value")));
            tx.put("status", CommandHelper.statusConfirmExplain((Integer)tx.get("status")));
            tx.put("fee", CommandHelper.naToNuls(tx.get("fee")));
            tx.put("time", DateUtil.convertDate(new Date((Long) tx.get("time"))));
        }
        map.put("txList", txList);

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

}

141:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\model\BlockDto.java
 */

package io.nuls.protocol.rpc.model;

import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.constant.TxStatusEnum;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.*;
import io.nuls.protocol.constant.ProtocolConstant;
```

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

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

/**
 * @author: Niels Wang
 */
@ApiModel(value = "blockJSON (, ), ")
public class BlockDto {

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

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

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

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

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

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

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

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

    @ApiModelProperty(name = "scriptSign", value = "Hex.encode(byte[])")
    private String scriptSign;

    @ApiModelProperty(name = "extend", value = "Hex.encode(byte[])")
    private String extend;
```

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

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

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

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

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

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

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

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

@ApiModelProperty(name = "txList", value = "transactionsJSON")
private List<TransactionDto> txList;

public BlockDto(Block block) throws IOException {
    this(block.getHeader());
    this.size = block.size();
    this.txList = new ArrayList<>();
    Na fee = Na.ZERO;
    for (Transaction tx : block.getTxs()) {
        this.txList.add(new TransactionDto(tx));
        fee = fee.add(tx.getFee());
        if (tx.getType() == ProtocolConstant.TX_TYPE_COINBASE) {
            setBlockReward(tx);
        }
        tx.setStatus(TxStatusEnum.CONFIRMED);
    }
    this.fee = fee.getValue();
```

```
}

private void setBlockReward(Transaction tx) {
    CoinData coinData = tx.getCoinData();
    if (null == coinData) {
        throw new NulsRuntimeException(TransactionErrorCode.COINDATA_NOT_FOUND);
    }
    Na rewardNa = Na.ZERO;
    for (Coin coin : coinData.getTo()) {
        rewardNa = rewardNa.add(coin.getNa());
    }
    this.reward = rewardNa.getValue();
}

public BlockDto(BlockHeader header) throws IOException {
    long bestBlockHeight = NulsContext.getInstance().getBestBlock().getHeader().getHeight();
    this.hash = header.getHash().getDigestHex();
    this.preHash = header.getPreHash().getDigestHex();
    this.merkleHash = header.getMerkleHash().getDigestHex();
    this.time = header.getTime();
    this.height = header.getHeight();
    this.txCount = header.getTxCount();
    this.packingAddress = Address.fromHashs(header.getPackingAddress()).getBase58();
    this.scriptSign = Hex.encode(header.getBlockSignature().serialize());
    this.extend = Hex.encode(header.getExtend());
    this.confirmCount = bestBlockHeight - this.height;
    try {
        BlockExtendsData roundData = new BlockExtendsData(header.getExtend());
        this.roundIndex = roundData.getRoundIndex();
        this.roundStartTime = roundData.getRoundStartTime();
        this.consensusMemberCount = roundData.getConsensusMemberCount();
        this.packingIndexOfRound = roundData.getPackingIndexOfRound();
        if(roundData.getStateRoot() != null) {
            this.stateRoot = Hex.encode(roundData.getStateRoot());
        }
    } catch (Exception e) {
        Log.error(e);
    }
}

public String getHash() {
    return hash;
```

```
}

public void setHash(String hash) {
    this.hash = hash;
}

public String getPreHash() {
    return preHash;
}

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

public String getMerkleHash() {
    return merkleHash;
}

public void setMerkleHash(String merkleHash) {
    this.merkleHash = merkleHash;
}

public Long getTime() {
    return time;
}

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

public Long getHeight() {
    return height;
}

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

public Long getTxCount() {
    return txCount;
}
```

```
public void setTxCount(Long txCount) {
    this.txCount = txCount;
}

public String getPackingAddress() {
    return packingAddress;
}

public void setPackingAddress(String packingAddress) {
    this.packingAddress = packingAddress;
}

public String getScriptSig() {
    return scriptSign;
}

public void setScriptSig(String scriptSig) {
    this.scriptSign = scriptSig;
}

public Long getRoundIndex() {
    return roundIndex;
}

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

public Integer getConsensusMemberCount() {
    return consensusMemberCount;
}

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

public Long getRoundStartTime() {
    return roundStartTime;
}

public void setRoundStartTime(Long roundStartTime) {
    this.roundStartTime = roundStartTime;
```

```
}

public Integer getPackingIndexOfRound() {
    return packingIndexOfRound;
}

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

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

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

public Long getReward() {
    return reward;
}

public void setReward(Long reward) {
    this.reward = reward;
}

public Long getFee() {
    return fee;
}

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

public Long getConfirmCount() {
    return confirmCount;
}

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

```
    public int getSize() {
        return size;
    }

    public void setSize(int size) {
        this.size = size;
    }

    public String getStateRoot() {
        return stateRoot;
    }

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

    public String getExtend() {
        return extend;
    }

    public void setExtend(String extend) {
        this.extend = extend;
    }
}

142:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\model\BlockHeaderDto.java
 */

package io.nuls.protocol.rpc.model;

import io.nuls.consensus.poc.model.BlockExtendsData;
import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.*;
import io.nuls.protocol.constant.ProtocolConstant;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
import java.io.IOException;

/**
 * @author: Niels Wang
 */
@ApiModel(value = "blockJSON (, ), ")
public class BlockHeaderDto {

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

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

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

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

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

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

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

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

    @ApiModelProperty(name = "scriptSign", value = "Hex.encode(byte[])")
    private String scriptSign;

    @ApiModelProperty(name = "extend", value = "Hex.encode(byte[])")
    private String extend;

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

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

```
private Integer consensusMemberCount;

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

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

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

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

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

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

public BlockHeaderDto(Block block) throws IOException {
    this(block.getHeader());
    this.size = block.getHeader().size();
    Na fee = Na.ZERO;
    for (Transaction tx : block.getTxs()) {
        fee = fee.add(tx.getFee());
        if (tx.getType() == ProtocolConstant.TX_TYPE_COINBASE) {
            setBlockReward(tx);
        }
    }
    this.fee = fee.getValue();
}

private void setBlockReward(Transaction tx) {
    CoinData coinData = tx.getCoinData();
    if (null == coinData) {
        throw new NulsRuntimeException(TransactionErrorCode.COINDATA_NOT_FOUND);
    }
    Na rewardNa = Na.ZERO;
    for (Coin coin : coinData.getTo()) {
        rewardNa = rewardNa.add(coin.getNa());
    }
```

```
    this.reward = rewardNa.getValue();
}

public BlockHeaderDto(BlockHeader header) throws IOException {
    long bestBlockHeight = NulsContext.getInstance().getBestBlock().getHeader().getHeight();
    this.hash = header.getHash().getDigestHex();
    this.preHash = header.getPreHash().getDigestHex();
    this.merkleHash = header.getMerkleHash().getDigestHex();
    this.time = header.getTime();
    this.height = header.getHeight();
    this.txCount = header.getTxCount();
    this.packingAddress = Address.fromHashs(header.getPackingAddress()).getBase58();
    this.scriptSign = Hex.encode(header.getBlockSignature().serialize());
    this.confirmCount = bestBlockHeight - this.height;
    this.extend = Hex.encode(header.getExtend());
    try {
        BlockExtendsData roundData = new BlockExtendsData(header.getExtend());
        this.roundIndex = roundData.getRoundIndex();
        this.roundStartTime = roundData.getRoundStartTime();
        this.consensusMemberCount = roundData.getConsensusMemberCount();
        this.packingIndexOfRound = roundData.getPackingIndexOfRound();
        if(roundData.getStateRoot() != null) {
            this.stateRoot = Hex.encode(roundData.getStateRoot());
        }
    } catch (Exception e) {
        Log.error(e);
    }
}

public String getHash() {
    return hash;
}

public void setHash(String hash) {
    this.hash = hash;
}

public String getPreHash() {
    return preHash;
}

public void setPreHash(String preHash) {
```

```
    this.preHash = preHash;
}

public String getMerkleHash() {
    return merkleHash;
}

public void setMerkleHash(String merkleHash) {
    this.merkleHash = merkleHash;
}

public Long getTime() {
    return time;
}

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

public Long getHeight() {
    return height;
}

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

public Long getTxCount() {
    return txCount;
}

public void setTxCount(Long txCount) {
    this.txCount = txCount;
}

public String getPackingAddress() {
    return packingAddress;
}

public void setPackingAddress(String packingAddress) {
    this.packingAddress = packingAddress;
}
```

```
public String getScriptSig() {
    return scriptSign;
}

public void setScriptSig(String scriptSig) {
    this.scriptSign = scriptSig;
}

public Long getRoundIndex() {
    return roundIndex;
}

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

public Integer getConsensusMemberCount() {
    return consensusMemberCount;
}

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

public Long getRoundStartTime() {
    return roundStartTime;
}

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

public Integer getPackingIndexOfRound() {
    return packingIndexOfRound;
}

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

public Long getReward() {
```

```
    return reward;
}

public void setReward(Long reward) {
    this.reward = reward;
}

public Long getFee() {
    return fee;
}

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

public Long getConfirmCount() {
    return confirmCount;
}

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

public int getSize() {
    return size;
}

public void setSize(int size) {
    this.size = size;
}

public String getStateRoot() {
    return stateRoot;
}

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

public String getExtend() {
    return extend;
}
```

```
    public void setExtend(String extend) {
        this.extend = extend;
    }
}

143:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\model\BlockInfoDto.java
 */

package io.nuls.protocol.rpc.model;

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

/**
 * @author: Niels Wang
 */
@ApiModel(value = "blockJSON (), ")
public class BlockInfoDto {

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

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

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

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

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

    public Long getHeight() {
```

```
        return height;
    }

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

    public Long getTxCount() {
        return txCount;
    }

    public void setTxCount(Long txCount) {
        this.txCount = txCount;
    }

    public String getPackingAddress() {
        return packingAddress;
    }

    public void setPackingAddress(String packingAddress) {
        this.packingAddress = packingAddress;
    }

}

144:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\model\InputDto.java
 */

package io.nuls.protocol.rpc.model;

import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

/**
 * @author Niels
 */
@ApiModel(value = "inputJSON")
public class InputDto {
```

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

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

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

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

public InputDto(Coin input) {
    NulsByteBuffer byteBuffer = new NulsByteBuffer(input.getOwner());
    try {
        this.fromHash = byteBuffer.readHash().getDigestHex();
        this.fromIndex = (int) byteBuffer.readVarInt();
    } catch (Exception e) {
        throw new NulsRuntimeException(e);
    }
    this.value = input.getNa().getValue();
}

public String getAddress() {
    return address;
}

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

public Long getValue() {
    return value;
}

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

public String getFromHash() {
    return fromHash;
```

```
    }

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

    public Integer getFromIndex() {
        return fromIndex;
    }

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


}

145:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\model\OutputDto.java
 */

package io.nuls.protocol.rpc.model;

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

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

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

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

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

    public OutputDto(Coin output) {
        //this.address = AddressTool.getStringAddressByBytes(output.());
```

```
        this.address = AddressTool.getStringAddressByBytes(output.getAddress());
        this.value = output.getNa().getValue();
        this.lockTime = output.getLockTime();
    }

    public String getAddress() {
        return address;
    }

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

    public Long getValue() {
        return value;
    }

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

    public Long getLockTime() {
        return lockTime;
    }

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

146:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\model\TransactionDto.java
 */

package io.nuls.protocol.rpc.model;

import io.nuls.core.tools.crypto.Hex;
import io.nuls.kernel.cfg.NulsConfig;
import io.nuls.kernel.constant.TxStatusEnum;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.CoinData;
```

```
import io.nuls.kernel.model.Transaction;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;

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

@ApiModel(value = "transactionJSON")
public class TransactionDto {

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

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

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

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

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

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

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

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

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

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

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

```
private int size;

@ApiModelProperty(name = "inputs", value = "")
private List<InputDto> inputs;

@ApiModelProperty(name = "outputs", value = "")
private List<OutputDto> outputs;

public TransactionDto(Transaction tx) {
    long bestBlockHeight = NulsContext.getInstance().getBestBlock().getHeader().getHeight();
    this.hash = tx.getHash().getDigestHex();
    this.type = tx.getType();
    this.time = tx.getTime();
    this.blockHeight = tx.getBlockHeight();
    this.fee = tx.getFee().getValue();
    this.size = tx.getSize();
    if (this.blockHeight > 0 || TxStatusEnum.CONFIRMED.equals(tx.getStatus())) {
        this.confirmCount = bestBlockHeight - this.blockHeight;
    } else {
        this.confirmCount = 0L;
    }
    if (TxStatusEnum.CONFIRMED.equals(tx.getStatus())) {
        this.status = 1;
    } else {
        this.status = 0;
    }

    if (tx.getRemark() != null) {
        try {
            this.setRemark(new String(tx.getRemark(), NulsConfig.DEFAULT_ENCODING));
        } catch (UnsupportedEncodingException e) {
            this.setRemark(Hex.encode(tx.getRemark()));
        }
    }
    if (tx.getTransactionSignature() != null) {
        this.setScriptSig(Hex.encode(tx.getTransactionSignature()));
    }

    CoinData coinData = tx.getCoinData();
    List<InputDto> inputs = new ArrayList<>();
    List<OutputDto> outputs = new ArrayList<>();
    if (coinData != null) {
```

```
            List<Coin> froms = coinData.getFrom();
            for (Coin from : froms) {
                inputs.add(new InputDto(from));
            }
            List<Coin> tos = coinData.getTo();
            for (Coin coin : tos) {
                outputs.add(new OutputDto(coin));
            }
        }
        this.inputs = inputs;
        this.outputs = outputs;


    }

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

    public Integer getType() {
        return type;
    }

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

    public Long getTime() {
        return time;
    }

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

    public Long getBlockHeight() {
        return blockHeight;
    }
```

```
public void setBlockHeight(Long blockHeight) {
    this.blockHeight = blockHeight;
}

public Long getFee() {
    return fee;
}

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

public Long getValue() {
    return value;
}

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

public List<InputDto> getInputs() {
    return inputs;
}

public void setInputs(List<InputDto> inputs) {
    this.inputs = inputs;
}

public List<OutputDto> getOutputs() {
    return outputs;
}

public void setOutputs(List<OutputDto> outputs) {
    this.outputs = outputs;
}

public String getRemark() {
    return remark;
}

public void setRemark(String remark) {
```

```
        this.remark = remark;
    }

    public String getScriptSig() {
        return scriptSig;
    }

    public void setScriptSig(String scriptSig) {
        this.scriptSig = scriptSig;
    }

    public Integer getStatus() {
        return status;
    }

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

    public Long getConfirmCount() {
        return confirmCount;
    }

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

    public int getSize() {
        return size;
    }

    public void setSize(int size) {
        this.size = size;
    }


}

147:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
rpc\src\main\java\io\nuls\protocol\rpc\resources\BlockResource.java
 */
```

```
package io.nuls.protocol.rpc.resources;

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.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.constant.ProtocolErroeCode;
import io.nuls.protocol.rpc.model.*;
import io.nuls.protocol.service.BlockService;
import io.swagger.annotations.*;

import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.io.IOException;
import java.util.*;

/**
 * @author: Niels Wang
 */
@Path("/block")
@Api(value = "/block", description = "Block")
@Component
public class BlockResource {

    @Autowired
    private BlockService blockService;

    @Autowired
    private LedgerService ledgerService;

    @GET
    @Path("/header/height/{height}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "", notes = "result.data: blockHeaderJson ")
    @ApiResponses(value = {
```

```
        @ApiResponse(code = 200, message = "success", response = BlockDto.class)
    })
    public RpcClientResult getHeaderByHeight(@ApiParam(name = "height", value = "", required =
true)
                                             @PathParam("height") Integer height) {
        AssertUtil.canNotEmpty(height);
        Result<Block> blockResult = blockService.getBlock(height);
        if (blockResult.isFailed()) {
            return blockResult.toRpcClientResult();
        }
        BlockHeaderDto dto = null;
        try {
            dto = new BlockHeaderDto(blockResult.getData());
        } catch (IOException e) {
            Log.error(e);
            return Result.getFailed(KernelErrorCode.IO_ERROR).toRpcClientResult();
        }
        return Result.getSuccess().setData(dto).toRpcClientResult();
    }



    @GET
    @Path("/header/hash/{hash}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "hash", notes = "result.data: blockHeaderJson ")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = BlockDto.class)
    })
    public RpcClientResult getHeader(@ApiParam(name = "hash", value = "hash", required = true)
                                     @PathParam("hash") String hash) {
        AssertUtil.canNotEmpty(hash);
        hash = StringUtils.formatStringPara(hash);
        if (!NulsDigestData.validHash(hash)) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        Result result = Result.getSuccess();
        Block block = null;
        try {
            block = blockService.getBlock(NulsDigestData.fromDigestHex(hash)).getData();
        } catch (NulsException e) {
            Log.error(e);
        }
```

```
    if (block == null) {
        return Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL).toRpcClientResult();
    }
    try {
        result.setData(new BlockHeaderDto(block));
    } catch (IOException e) {
        Log.error(e);
        return Result.getFailed(KernelErrorCode.IO_ERROR).toRpcClientResult();
    }
    return result.toRpcClientResult();
}

@GET
@Path("/hash/{hash}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("hash")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = BlockDto.class)
})
public RpcClientResult loadBlock(@ApiParam(name = "hash", value = "hash", required = true)
                                 @PathParam("hash") String hash) throws IOException {
    AssertUtil.canNotEmpty(hash);
    Result result;
    if (!NulsDigestData.validHash(hash)) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }
    Block block = null;
    try {
        block = blockService.getBlock(NulsDigestData.fromDigestHex(hash)).getData();
    } catch (NulsException e) {
        Log.error(e);
    }
    if (block == null) {
        result = Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
    } else {
        result = Result.getSuccess();
        BlockDto dto = new BlockDto(block);
        fillBlockTxInputAddress(dto);
        calTransactionValue(dto);
        result.setData(dto);

    }
```

```
    return result.toRpcClientResult();
}

@GET
@Path("/height/{height}")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation("")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = BlockDto.class)
})
public RpcClientResult getBlock(@ApiParam(name = "height", value = "", required = true)
                                @PathParam("height") Long height) throws IOException {
    AssertUtil.canNotEmpty(height);
    Result result = Result.getSuccess();
    if (height < 0) {
        return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
    }

    Block block = blockService.getBlock(height).getData();
    if (block == null) {
        result = Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
    } else {
        BlockDto dto = new BlockDto(block);
        fillBlockTxInputAddress(dto);
        calTransactionValue(dto);
        result.setData(dto);
    }
    return result.toRpcClientResult();
}

@GET
@Path("/newest")
@Produces(MediaType.APPLICATION_JSON)
@ApiOperation(value = "", notes = "result.data: blockHeaderJson ")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "success", response = BlockDto.class)
})
public RpcClientResult getBestBlockHeader() throws IOException {
    Result result = Result.getSuccess();
    result.setData(new BlockHeaderDto(NulsContext.getInstance().getBestBlock()));
    return result.toRpcClientResult();
}
```

```
    @GET
    @Path("/newest/height")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult getBestBlockHight() throws IOException {
        long hight = NulsContext.getInstance().getBestBlock().getHeader().getHeight();
        Map<String, Long> map = new HashMap<>();
        map.put("value", hight);
        return Result.getSuccess().setData(map).toRpcClientResult();
    }

    @GET
    @Path("/newest/hash")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation(value = "hash")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult getBestBlockHash() throws IOException {
        String hash =
NulsContext.getInstance().getBestBlock().getHeader().getHash().getDigestHex();
        Map<String, String> map = new HashMap<>();
        map.put("value", hash);
        return Result.getSuccess().setData(map).toRpcClientResult();
    }

    @GET
    @Path("/bytes")
    @Produces(MediaType.APPLICATION_JSON)
    public RpcClientResult getBlockBytes(@QueryParam("hash") String hash) throws IOException {
        Result result;
        if (!NulsDigestData.validHash(hash)) {
            return Result.getFailed(KernelErrorCode.PARAMETER_ERROR).toRpcClientResult();
        }
        Block block = null;
        try {
            // ()
            block = blockService.getBlock(NulsDigestData.fromDigestHex(hash), true).getData();
```

```
        } catch (NulsException e) {
            Log.error(e);
        }
        if (block == null) {
            result = Result.getFailed(ProtocolErroeCode.BLOCK_IS_NULL);
        } else {
            result = Result.getSuccess();
            Map<String, String> map = new HashMap<>();
            map.put("value", Base64.getEncoder().encodeToString(block.serialize()));
            result.setData(map);
        }
        return result.toRpcClientResult();
    }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = BlockDto.class)
    })
    public RpcClientResult getBlockList(@QueryParam("startHeight") Long startHeight,
@QueryParam("size") Long size) throws IOException {
        if (size > 100) {
            return RpcClientResult.getFailed("the size is too big");
        }
        long bestHeight = NulsContext.getInstance().getBestHeight();
        if (startHeight > bestHeight) {
            return RpcClientResult.getFailed("The start height is to high!");
        }
        List<BlockDto> list = new ArrayList<>();

        for (int i = 0; i < size && (startHeight + i) <= bestHeight; i++) {
            Block block = blockService.getBlock(startHeight + i).getData();
            if (null == block) {
                break;
            }
            BlockDto dto = new BlockDto(block);
            fillBlockTxInputAddress(dto);
            calTransactionValue(dto);
            list.add(dto);
        }
        Map<String, List<BlockDto>> map = new HashMap<>();
```

```
    map.put("list", list);
    return Result.getSuccess().setData(map).toRpcClientResult();

}

private void fillBlockTxInputAddress(BlockDto dto) {
    for (TransactionDto transaction : dto.getTxList()) {
        if (transaction.getInputs() == null || transaction.getInputs().isEmpty()) {
            continue;
        }
        for (InputDto inputDto : transaction.getInputs()) {
            Transaction tx;
            try {
                tx = ledgerService.getTx(NulsDigestData.fromDigestHex(inputDto.getFromHash()));
            } catch (NulsException e) {
                Log.error(e);
                continue;
            }
            Coin coin = tx.getCoinData().getTo().get(inputDto.getFromIndex());
            //inputDto.setAddress(AddressTool.getStringAddressByBytes(coin.()));
            inputDto.setAddress(AddressTool.getStringAddressByBytes(coin.getAddress()));
        }
    }

}

/**
 * ()
 * Calculate the actual amount of the transaction.
 */
private void calTransactionValue(BlockDto dto) {
    for (TransactionDto txDto : dto.getTxList()) {
        if (txDto == null) {
            break;
        }
        Set<String> inputAdressSet = null;
        if (txDto.getInputs() != null && !txDto.getInputs().isEmpty()) {
            List<InputDto> inputDtoList = txDto.getInputs();
            inputAdressSet = new HashSet<>(inputDtoList.size());
            for (InputDto inputDto : inputDtoList) {
                inputAdressSet.add(inputDto.getAddress());
            }
```

```
            }
            Na value = Na.ZERO;
            List<OutputDto> outputDtoList = txDto.getOutputs();
            for (OutputDto outputDto : outputDtoList) {
                if (null != inputAdressSet && inputAdressSet.contains(outputDto.getAddress())) {
                    continue;
                }
                value = value.add(Na.valueOf(outputDto.getValue()));
            }
            txDto.setValue(value.getValue());
        }
    }


    @GET()
    @Path("/info")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = BlockInfoDto.class)
    })
    public RpcClientResult getBlockInfoList() throws IOException {
        Block bestBlock = NulsContext.getInstance().getBestBlock();
        List<BlockInfoDto> list = new ArrayList<>();
        int count = 0;
        NulsDigestData preHash = null;
        while (count < 5) {
            count++;
            BlockHeader header = null;
            if (null != preHash) {
                header = blockService.getBlockHeader(preHash).getData();
            } else {
                header = bestBlock.getHeader();
            }
            if (null == header) {
                return RpcClientResult.getFailed(KernelErrorCode.DATA_ERROR);
            }
            BlockInfoDto dto = new BlockInfoDto();
            dto.setHeight(header.getHeight());
            dto.setHash(header.getHash().getDigestHex());
dto.setPackingAddress(AddressTool.getStringAddressByBytes(header.getPackingAddress()));
        dto.setTxCount(header.getTxCount());
```

```
            list.add(dto);
            preHash = header.getPreHash();
        }
        Map<String, List<BlockInfoDto>> map = new HashMap<>();
        map.put("list", list);
        return Result.getSuccess().setData(map).toRpcClientResult();

    }
}
```

148:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-rpc\src\test\java\io\nuls\protocol\rpc\resources\BlockResourceTest.java
*/

```
package io.nuls.protocol.rpc.resources;

import org.junit.Test;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;

import static javax.ws.rs.core.MediaType.APPLICATION_JSON;
import static org.junit.Assert.assertTrue;

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

    @Test
    public void getBlockBytes() {
        assertTrue(true);
//
//        Client client = ClientBuilder.newClient();
//        WebTarget target = client.target("http://127.0.0.1:8001/").path("/block/bytes");
//        target = target.queryParam("height", 1);
//        byte[] response = target.request(APPLICATION_JSON).get(byte[].class);
//        System.out.println(response.length);
    }
```

```
}

149:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-storage\src\main\java\io\nuls\protocol\storage\constant\ProtocolStorageConstant.java
 */

package io.nuls.protocol.storage.constant;

import io.nuls.core.tools.crypto.Hex;
import io.nuls.core.tools.str.StringUtils;

/**
 *
 * The protocol module stores a collection of related constants.
 *
 * @author: Niels Wang
 */
public interface ProtocolStorageConstant {
    /**
     *
     * Block header height index table.
     */
    String DB_NAME_BLOCK_HEADER_INDEX = "block_header_index";

    /**
     *
     * Block header table name.
     */
    String DB_NAME_BLOCK_HEADER = "block_header";

    /**
     * hash
     * The index value of the latest block hash stored in the database.
     */
    String BEST_BLOCK_HASH_INDEX = "best_block_hash_index";
    /**
     *
     */
    String NULS_VERSION_AREA = "nuls_version_area";
    /**
     *
     */
```

```
    String NULS_PROTOCOL_AREA = "nuls_protocol_area";
    /**
     *
     */
    String PROTOCOL_TEMP_AREA = "protocol_temp_area";

    String BLOCK_PROTOCOL_AREA = "block_protocol_area";

    String BLOCK_TEMP_PROTOCOL_AREA = "block_temp_protocol_area";

    String BLOCK_PROTOCOL_INDEX = "block_protocol_index";

    String BLOCK_TEMP_PROTOCOL_INDEX = "block_temp_protocol_index";

    String CONSENSUS_VERSION_AREA = "consensus_version_area";

    String BLOCK_PROTOCOL_HEIGHT = "block_protocol_height";
    /**
     * key
     */
    byte[] MAIN_VERSION_KEY = StringUtils.bytes("mainVersion");

    /**
     * key
     */
    byte[] CHANGE_HASH_HEIGHT_KEY = StringUtils.bytes("changeHashHeight");
}

150:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
storage\src\main\java\io\nuls\protocol\storage\po\BlockHeaderPo.java
 */

package io.nuls.protocol.storage.po;

import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;

import io.nuls.kernel.script.BlockSignature;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
```

```
import io.nuls.kernel.utils.SerializeUtils;

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

/**
 * @author: Niels Wang
 */
public class BlockHeaderPo extends BaseNulsData {

    private transient NulsDigestData hash;

    private NulsDigestData preHash;

    private NulsDigestData merkleHash;

    private long time;

    private long height = -1L;

    private long txCount;

    private byte[] packingAddress;

    private BlockSignature scriptSign;

    private byte[] extend;

    private byte[] stateRoot;

    private List<NulsDigestData> txHashList;


    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfNulsData(preHash);
        size += SerializeUtils.sizeOfNulsData(merkleHash);
        size += SerializeUtils.sizeOfVarInt(time);
        size += SerializeUtils.sizeOfVarInt(height);
        size += SerializeUtils.sizeOfVarInt(txCount);
```

```
    size += SerializeUtils.sizeOfBytes(extend);
    size += SerializeUtils.sizeOfNulsData(scriptSign);
    for (NulsDigestData hash : txHashList) {
        size += SerializeUtils.sizeOfNulsData(hash);
    }
    if (NulsContext.MAIN_NET_VERSION > 1) {
        size += SerializeUtils.sizeOfBytes(stateRoot);
    }
    return size;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeNulsData(preHash);
    stream.writeNulsData(merkleHash);
    stream.writeVarInt(time);
    stream.writeVarInt(height);
    stream.writeVarInt(txCount);
    stream.writeBytesWithLength(extend);
    stream.writeNulsData(scriptSign);
    for (NulsDigestData hash : txHashList) {
        stream.writeNulsData(hash);
    }
    if (NulsContext.MAIN_NET_VERSION > 1) {
        stream.writeBytesWithLength(stateRoot);
    }
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.preHash = byteBuffer.readHash();
    this.merkleHash = byteBuffer.readHash();
    this.time = byteBuffer.readVarInt();
    this.height = byteBuffer.readVarInt();
    this.txCount = byteBuffer.readVarInt();
    this.extend = byteBuffer.readByLengthByte();
    this.scriptSign = byteBuffer.readNulsData(new BlockSignature());
    this.txHashList = new ArrayList<>();
    for (int i = 0; i < txCount; i++) {
        this.txHashList.add(byteBuffer.readHash());
    }
    if (!byteBuffer.isFinished()) {
```

```
        this.stateRoot = byteBuffer.readByLengthByte();
    }
}

public BlockHeaderPo() {
}

public NulsDigestData getHash() {
    return hash;
}

public void setHash(NulsDigestData hash) {
    this.hash = hash;
}

public NulsDigestData getPreHash() {
    return preHash;
}

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

public NulsDigestData getMerkleHash() {
    return merkleHash;
}

public void setMerkleHash(NulsDigestData merkleHash) {
    this.merkleHash = merkleHash;
}

public long getTime() {
    return time;
}

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

public long getHeight() {
    return height;
}
```

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

public long getTxCount() {
    return txCount;
}

public void setTxCount(long txCount) {
    this.txCount = txCount;
}

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

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

public BlockSignature getScriptSign() {
    return scriptSign;
}

public void setScriptSign(BlockSignature scriptSign) {
    this.scriptSign = scriptSign;
}

public byte[] getExtend() {
    return extend;
}

public void setExtend(byte[] extend) {
    this.extend = extend;
}

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

public void setStateRoot(byte[] stateRoot) {
```

```
        this.stateRoot = stateRoot;
    }

    public List<NulsDigestData> getTxHashList() {
        return txHashList;
    }

    public void setTxHashList(List<NulsDigestData> txHashList) {
        this.txHashList = txHashList;
    }

}
```

151:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-storage\src\main\java\io\nuls\protocol\storage\po\BlockProtocolInfoPo.java

```
 */
package io.nuls.protocol.storage.po;

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

/**
 * @author: Charlie
 * @date: 2018/8/17
 */
public class BlockProtocolInfoPo {
    /**
     *
     */
    private long blockHeight;
    /**
     *
     */
    private int version;
    /**
     *
     */
    private long currentDelay;
    /**
     *
     */
    private Set<String> addressSet;
```

```
/****/
private long roundIndex;
/**
 *
 */
private int status;
/**
 *
 */
private Long effectiveHeight;

private int prePercent;

public BlockProtocolInfoPo() {
    addressSet = new HashSet<>();
}

public BlockProtocolInfoPo(ProtocolTempInfoPo tempInfoPo) {
    this.version = tempInfoPo.getVersion();
    this.currentDelay = tempInfoPo.getCurrentDelay();
    this.addressSet = tempInfoPo.getAddressSet();
    this.roundIndex = tempInfoPo.getRoundIndex();
    this.status = tempInfoPo.getStatus();
    this.effectiveHeight = tempInfoPo.getEffectiveHeight();
}

public int getVersion() {
    return version;
}

public void setVersion(int version) {
    this.version = version;
}

public long getCurrentDelay() {
    return currentDelay;
}

public void setCurrentDelay(long currentDelay) {
    this.currentDelay = currentDelay;
}
```

```
public Set<String> getAddressSet() {
    return addressSet;
}

public void setAddressSet(Set<String> addressSet) {
    this.addressSet = addressSet;
}

public int getStatus() {
    return status;
}

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

public long getRoundIndex() {
    return roundIndex;
}

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

public Long getEffectiveHeight() {
    return effectiveHeight;
}

public void setEffectiveHeight(Long effectiveHeight) {
    this.effectiveHeight = effectiveHeight;
}

public long getBlockHeight() {
    return blockHeight;
}

public void setBlockHeight(long blockHeight) {
    this.blockHeight = blockHeight;
}

public int getPrePercent() {
```

```
        return prePercent;
    }

    public void setPrePercent(int prePercent) {
        this.prePercent = prePercent;
    }

}

152:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
storage\src\main\java\io\nuls\protocol\storage\po\ProtocolInfoPo.java
 */
package io.nuls.protocol.storage.po;

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

/**
 * @author: Charlie
 * @date: 2018/8/17
 */
public class ProtocolInfoPo {
    /**
     *
     */
    private int version;
    /****/
    private int percent;
    /****/
    private long delay;
    /**
     *
     */
    private long currentDelay;

    private int currentPercent;
    /**
     *
     */
    private Set<String> addressSet;
    /****/
    private long roundIndex;
```

```
/**
 *
 */
private int status;
/**
 *
 */
private Long effectiveHeight;

private int prePercent;

public ProtocolInfoPo() {
    addressSet = new HashSet<>();
}

public ProtocolInfoPo(ProtocolTempInfoPo tempInfoPo) {
    this.version = tempInfoPo.getVersion();
    this.percent = tempInfoPo.getPercent();
    this.delay = tempInfoPo.getDelay();
    this.currentDelay = tempInfoPo.getCurrentDelay();
    this.addressSet = tempInfoPo.getAddressSet();
    this.roundIndex = tempInfoPo.getRoundIndex();
    this.status = tempInfoPo.getStatus();
    this.effectiveHeight = tempInfoPo.getEffectiveHeight();
    this.setCurrentPercent(tempInfoPo.getCurrentPercent());
    this.prePercent = tempInfoPo.getPrePercent();
}

public int getVersion() {
    return version;
}

public void setVersion(int version) {
    this.version = version;
}

public long getCurrentDelay() {
    return currentDelay;
}

public void setCurrentDelay(long currentDelay) {
    this.currentDelay = currentDelay;
```

```
}

public Set<String> getAddressSet() {
    return addressSet;
}

public void setAddressSet(Set<String> addressSet) {
    this.addressSet = addressSet;
}

public int getStatus() {
    return status;
}

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

public long getRoundIndex() {
    return roundIndex;
}

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

public int getPercent() {
    return percent;
}

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

public long getDelay() {
    return delay;
}

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

```
    public Long getEffectiveHeight() {
        return effectiveHeight;
    }

    public void setEffectiveHeight(Long effectiveHeight) {
        this.effectiveHeight = effectiveHeight;
    }

    public int getCurrentPercent() {
        return currentPercent;
    }

    public void setCurrentPercent(int currentPercent) {
        this.currentPercent = currentPercent;
    }

    public int getPrePercent() {
        return prePercent;
    }

    public void setPrePercent(int prePercent) {
        this.prePercent = prePercent;
    }

}

153:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
storage\src\main\java\io\nuls\protocol\storage\po\ProtocolTempInfoPo.java
 */
package io.nuls.protocol.storage.po;

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

public class ProtocolTempInfoPo {

    /***/
    private int version;
    /****/
    private int percent;
    /****/
    private long delay;
```

```
/**
 *
 */
private long currentDelay;

private int currentPercent;

private int prePercent;

/**
 *
 */
private Set<String> addressSet;
/****/
private long roundIndex;
/**
 *
 */
private int status;

/**
 *
 */
private Long effectiveHeight;

public ProtocolTempInfoPo(){
    addressSet = new HashSet<>();
}

public int getVersion() {
    return version;
}

public void setVersion(int version) {
    this.version = version;
}

public long getCurrentDelay() {
    return currentDelay;
}

public void setCurrentDelay(long currentDelay) {
```

```
    this.currentDelay = currentDelay;
}

public Set<String> getAddressSet() {
    return addressSet;
}

public void setAddressSet(Set<String> addressSet) {
    this.addressSet = addressSet;
}

public int getStatus() {
    return status;
}

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

public long getRoundIndex() {
    return roundIndex;
}

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

public int getPercent() {
    return percent;
}

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

public long getDelay() {
    return delay;
}

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

```
    public Long getEffectiveHeight() {
        return effectiveHeight;
    }

    public void setEffectiveHeight(Long effectiveHeight) {
        this.effectiveHeight = effectiveHeight;
    }

    public String getProtocolKey() {
        return version + "-" + percent + "-" + delay;
    }

    public void reset() {
        this.currentDelay = 0;
        this.roundIndex = 0;
        this.status = 0;
        this.effectiveHeight = null;
        this.addressSet.clear();
    }

    public int getCurrentPercent() {
        return currentPercent;
    }

    public void setCurrentPercent(int currentPercent) {
        this.currentPercent = currentPercent;
    }

    public int getPrePercent() {
        return prePercent;
    }

    public void setPrePercent(int prePercent) {
        this.prePercent = prePercent;
    }

}

154:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
storage\src\main\java\io\nuls\protocol\storage\service\BlockHeaderStorageService.java
 */
```

```
package io.nuls.protocol.storage.service;

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.protocol.storage.po.BlockHeaderPo;

/**
 *
 * Block header data storage service interface.
 *
 * @author: Niels Wang
 */
public interface BlockHeaderStorageService {

    /**
     *
     * Query block header data according to block height.
     *
     * @param height /block height
     * @return BlockHeaderPo
     */
    BlockHeaderPo getBlockHeaderPo(long height);

    /**
     * hash
     * Query block header data according to block hash.
     *
     * @param hash /block hash
     * @return BlockHeaderPo
     */
    BlockHeaderPo getBlockHeaderPo(NulsDigestData hash);

    /**
     *
     * Save the block header data to the storage.
     *
     * @param po /block header data
     * @return /operating result
     */
    Result saveBlockHeader(BlockHeaderPo po);
```

```
    /**
     *
     * Remove block header data from storage.
     *
     * @param po /block header data
     * @return /operating result
     */
    Result removeBlockHerader(BlockHeaderPo po);

    /**
     *
     * Get the latest block header.
     */
    BlockHeaderPo getBestBlockHeaderPo();
}

155:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-storage\src\main\java\io\nuls\protocol\storage\service\impl\BlockHeaderStorageServiceImpl.java
 */

package io.nuls.protocol.storage.service.impl;

import io.nuls.core.tools.log.Log;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.utils.VarInt;
import io.nuls.protocol.storage.constant.ProtocolStorageConstant;
import io.nuls.protocol.storage.po.BlockHeaderPo;
import io.nuls.protocol.storage.service.BlockHeaderStorageService;

import java.io.IOException;

/**
 *
```

```
 * Block header data storage service implementation class.
 *
 * @author: Niels Wang
 */

@Service
public class BlockHeaderStorageServiceImpl implements BlockHeaderStorageService,
InitializingBean {

    private byte[] bestBlockKey;

    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    /**
     *
     * Create a storage table, or throw an exception if it is normal if it is already existing.
     */
    @Override
    public void afterPropertiesSet() {
        Result result =
this.dbService.createArea(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER_INDEX);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
        result = this.dbService.createArea(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
        try {
            bestBlockKey =
NulsDigestData.calcDigestData(ProtocolStorageConstant.BEST_BLOCK_HASH_INDEX.getBytes
()).serialize();
        } catch (IOException e) {
            throw new NulsRuntimeException(e.getCause());
        }
    }
```

```
    /**
     *
     * Query block header data according to block height.
     *
     * @param height /block height
     * @return BlockHeaderPo
     */
    @Override
    public BlockHeaderPo getBlockHeaderPo(long height) {
        if(height < 0L) {
            return null;
        }
        byte[] hashBytes =
dbService.get(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER_INDEX, new
VarInt(height).encode());
        if (null == hashBytes) {
            return null;
        }
        return getBlockHeaderPo(hashBytes);
    }

    /**
     * hash
     * Query block header data according to block hash.
     *
     * @param hash /block hash
     * @return BlockHeaderPo
     */
    @Override
    public BlockHeaderPo getBlockHeaderPo(NulsDigestData hash) {
        if (null == hash) {
            return null;
        }
        try {
            return getBlockHeaderPo(hash.serialize());
        } catch (IOException e) {
            Log.error(e);
            return null;
        }
    }

    /**
```

```
     * hash
     * Query block header data according to block hash.
     *
     * @param hashBytes /block hash
     * @return BlockHeaderPo
     */
    private BlockHeaderPo getBlockHeaderPo(byte[] hashBytes) {
        byte[] bytes = dbService.get(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER,
hashBytes);
        if (null == bytes) {
            return null;
        }
        BlockHeaderPo po = new BlockHeaderPo();
        try {
            po.parse(bytes,0);
        } catch (NulsException e) {
            Log.error(e);
        }
        NulsDigestData hash = new NulsDigestData();
        try {
            hash.parse(hashBytes,0);
        } catch (NulsException e) {
            Log.error(e);
        }
        po.setHash(hash);
        return po;
    }

    /**
     *
     * Save the block header data to the storage.
     *
     * @param po /block header data
     * @return /operating result
     */
    @Override
    public Result saveBlockHeader(BlockHeaderPo po) {
        if (null == po) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        byte[] hashBytes = null;
        try {
```

```
            hashBytes = po.getHash().serialize();
        } catch (IOException e) {
            Log.error(e);
            return Result.getFailed(KernelErrorCode.IO_ERROR);
        }
        Result result = null;
        try {
            result = dbService.put(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER,
hashBytes, po.serialize());
        } catch (IOException e) {
            Log.error(e);
            return Result.getFailed(KernelErrorCode.IO_ERROR);
        }
        if (result.isFailed()) {
            return result;
        }
        result = dbService.put(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER_INDEX, new
VarInt(po.getHeight()).encode(), hashBytes);
        if (result.isFailed()) {
            this.removeBlockHerader(hashBytes);
            return result;
        }
        dbService.put(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER_INDEX,
bestBlockKey, hashBytes);
        return Result.getSuccess();
    }

    private Result removeBlockHerader(byte[] hashBytes) {
        if (null == hashBytes) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        return dbService.delete(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER,
hashBytes);
    }

    /**
     *
     * Remove block header data from storage.
     *
     * @param po ,/Block heads, abstracts and heights must be available.
     * @return /operating result
     */
```

```
    @Override
    public Result removeBlockHerader(BlockHeaderPo po) {
        if (null == po || po.getHeight() < 0 || po.getHash() == null || po.getPreHash() == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        dbService.delete(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER_INDEX, new
VarInt(po.getHeight()).encode());
        try {
            dbService.put(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER_INDEX,
bestBlockKey, po.getPreHash().serialize());
        } catch (IOException e) {
            Log.error(e);
        }
        try {
            return removeBlockHerader(po.getHash().serialize());
        } catch (IOException e) {
            Log.error(e);
            return Result.getFailed();
        }
    }

    /**
     *
     * Gets the latest block header data.
     */
    @Override
    public BlockHeaderPo getBestBlockHeaderPo() {
        byte[] hashBytes =
dbService.get(ProtocolStorageConstant.DB_NAME_BLOCK_HEADER_INDEX, bestBlockKey);
        if (null == hashBytes) {
            return null;
        }
        return getBlockHeaderPo(hashBytes);
    }

}

156:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
storage\src\main\java\io\nuls\protocol\storage\service\impl\VersionManagerStorageServiceImpl.jav
a
 */
package io.nuls.protocol.storage.service.impl;
```

```
import io.nuls.core.tools.crypto.Util;
import io.nuls.db.constant.DBErrorCode;
import io.nuls.db.service.DBService;
import io.nuls.kernel.exception.NulsRuntimeException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.utils.VarInt;
import io.nuls.protocol.storage.constant.ProtocolStorageConstant;
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 org.checkerframework.checker.units.qual.A;

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

/**
 * @author: Charlie
 * @date: 2018/8/17
 */
@Service
public class VersionManagerStorageServiceImpl implements VersionManagerStorageService,
InitializingBean {
    /**
     *
     * Universal data storage services.
     */
    @Autowired
    private DBService dbService;

    /**
     *
     * Create a storage table, or throw an exception if it is normal if it is already existing.
     */
    @Override
```

```
    public void afterPropertiesSet() {
        Result result =
this.dbService.createArea(ProtocolStorageConstant.NULS_VERSION_AREA);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result = this.dbService.createArea(ProtocolStorageConstant.NULS_PROTOCOL_AREA);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result = this.dbService.createArea(ProtocolStorageConstant.PROTOCOL_TEMP_AREA);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result = this.dbService.createArea(ProtocolStorageConstant.BLOCK_PROTOCOL_INDEX);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result = this.dbService.createArea(ProtocolStorageConstant.BLOCK_PROTOCOL_AREA);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result =
this.dbService.createArea(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_INDEX);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result =
this.dbService.createArea(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_AREA);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result =
this.dbService.createArea(ProtocolStorageConstant.CONSENSUS_VERSION_AREA);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
```

```
            throw new NulsRuntimeException(result.getErrorCode());
        }

        result = this.dbService.createArea(ProtocolStorageConstant.BLOCK_PROTOCOL_HEIGHT);
        if (result.isFailed() && !DBErrorCode.DB_AREA_EXIST.equals(result.getErrorCode())) {
            throw new NulsRuntimeException(result.getErrorCode());
        }
    }

    @Override
    public Result saveMainVersion(int version) {
        return dbService.put(ProtocolStorageConstant.NULS_VERSION_AREA,
ProtocolStorageConstant.MAIN_VERSION_KEY, Util.intToBytes(version));
    }

    @Override
    public Integer getMainVersion() {
        byte[] mainVersion = dbService.get(ProtocolStorageConstant.NULS_VERSION_AREA,
ProtocolStorageConstant.MAIN_VERSION_KEY);
        return null == mainVersion ? null : Util.byteToInt(mainVersion);
    }

    @Override
    public Result saveProtocolInfoPo(ProtocolInfoPo upgradeInfoPo) {
        return dbService.putModel(ProtocolStorageConstant.NULS_PROTOCOL_AREA,
Util.intToBytes(upgradeInfoPo.getVersion()), upgradeInfoPo);
    }

    @Override
    public ProtocolInfoPo getProtocolInfoPo(int version) {
        return (ProtocolInfoPo)
dbService.getModel(ProtocolStorageConstant.NULS_PROTOCOL_AREA,
Util.intToBytes(version));
    }

    @Override
    public Result saveProtocolTempInfoPo(ProtocolTempInfoPo tempInfoPo) {
        try {
            return dbService.putModel(ProtocolStorageConstant.PROTOCOL_TEMP_AREA,
tempInfoPo.getProtocolKey().getBytes("utf-8"), tempInfoPo);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
```

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

    @Override
    public ProtocolTempInfoPo getProtocolTempInfoPo(String key) {
        try {
            return dbService.getModel(ProtocolStorageConstant.PROTOCOL_TEMP_AREA,
key.getBytes("utf-8"), ProtocolTempInfoPo.class);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    public Result saveBlockProtocolInfoPo(BlockProtocolInfoPo protocolInfoPo) {
        List<Long> blockHeightIndex =
dbService.getModel(ProtocolStorageConstant.BLOCK_PROTOCOL_INDEX,
Util.intToBytes(protocolInfoPo.getVersion()), List.class);
        if (blockHeightIndex == null) {
            blockHeightIndex = new ArrayList<>();
        }
        blockHeightIndex.add(protocolInfoPo.getBlockHeight());
        dbService.putModel(ProtocolStorageConstant.BLOCK_PROTOCOL_INDEX,
Util.intToBytes(protocolInfoPo.getVersion()), blockHeightIndex);
        return dbService.putModel(ProtocolStorageConstant.BLOCK_PROTOCOL_AREA, new
VarInt(protocolInfoPo.getBlockHeight()).encode(), protocolInfoPo);
    }

    @Override
    public List<Long> getBlockProtocolIndex(int version) {
        return dbService.getModel(ProtocolStorageConstant.BLOCK_PROTOCOL_INDEX,
Util.intToBytes(version), List.class);
    }

    @Override
    public List<Long> getBlockTempProtocolIndex(int version) {
        return dbService.getModel(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_INDEX,
Util.intToBytes(version), List.class);
    }
```

```
    @Override
    public void saveBlockProtocolIndex(int version, List<Long> list) {
        dbService.putModel(ProtocolStorageConstant.BLOCK_PROTOCOL_INDEX,
Util.intToBytes(version), list);
    }

    @Override
    public void saveTempBlockProtocolIndex(int version, List<Long> list) {
        dbService.putModel(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_INDEX,
Util.intToBytes(version), list);
    }

    @Override
    public BlockProtocolInfoPo getBlockProtocolInfoPo(long blockHeight) {
        return dbService.getModel(ProtocolStorageConstant.BLOCK_PROTOCOL_AREA, new
VarInt(blockHeight).encode(), BlockProtocolInfoPo.class);
    }

    @Override
    public BlockProtocolInfoPo getBlockTempProtocolInfoPo(long blockHeight) {
        return dbService.getModel(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_AREA,
new VarInt(blockHeight).encode(), BlockProtocolInfoPo.class);
    }

    @Override
    public void clearBlockProtocol(long blockHeight, int version) {
        dbService.delete(ProtocolStorageConstant.BLOCK_PROTOCOL_INDEX,
Util.intToBytes(version));
        dbService.delete(ProtocolStorageConstant.BLOCK_PROTOCOL_AREA, new
VarInt(blockHeight).encode());
    }

    @Override
    public void clearTempBlockProtocol(long blockHeight, int version) {
        dbService.delete(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_INDEX,
Util.intToBytes(version));
        dbService.delete(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_AREA, new
VarInt(blockHeight).encode());
    }

    @Override
    public Result saveBlockProtocolTempInfoPo(BlockProtocolInfoPo protocolInfoPo) {
```

```
        List<Long> blockHeightIndex =
dbService.getModel(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_INDEX,
Util.intToBytes(protocolInfoPo.getVersion()), List.class);
        if (blockHeightIndex == null) {
            blockHeightIndex = new ArrayList<>();
        }
        blockHeightIndex.add(protocolInfoPo.getBlockHeight());
        dbService.putModel(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_INDEX,
Util.intToBytes(protocolInfoPo.getVersion()), blockHeightIndex);
        return dbService.putModel(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_AREA,
new VarInt(protocolInfoPo.getBlockHeight()).encode(), protocolInfoPo);
    }


    @Override
    public void deleteBlockProtocol(long blockHeight) {
        dbService.delete(ProtocolStorageConstant.BLOCK_PROTOCOL_AREA, new
VarInt(blockHeight).encode());
    }

    @Override
    public void deleteBlockTempProtocol(long blockHeight) {
        dbService.delete(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_AREA, new
VarInt(blockHeight).encode());
    }

    @Override
    public Result saveConsensusVersionMap(Map<String, Integer> versionMap) {
        return dbService.putModel(ProtocolStorageConstant.CONSENSUS_VERSION_AREA,
ProtocolStorageConstant.CONSENSUS_VERSION_AREA.getBytes(), versionMap);
    }

    @Override
    public Map<String, Integer> getConsensusVersionMap() {
        return dbService.getModel(ProtocolStorageConstant.BLOCK_TEMP_PROTOCOL_AREA,
ProtocolStorageConstant.CONSENSUS_VERSION_AREA.getBytes(), HashMap.class);
    }

    @Override
    public Result saveConsensusVersionHeight(Long blockHeight) {
        return dbService.putModel(ProtocolStorageConstant.BLOCK_PROTOCOL_HEIGHT,
ProtocolStorageConstant.BLOCK_PROTOCOL_HEIGHT.getBytes(), blockHeight);
```

```
    }

    @Override
    public Long getConsensusVersionHeight() {
        return dbService.getModel(ProtocolStorageConstant.BLOCK_PROTOCOL_HEIGHT,
ProtocolStorageConstant.BLOCK_PROTOCOL_HEIGHT.getBytes(), Long.class);
    }

    @Override
    public Map<String, ProtocolTempInfoPo> getProtocolTempMap() {
        List<ProtocolTempInfoPo> list =
dbService.values(ProtocolStorageConstant.PROTOCOL_TEMP_AREA,
ProtocolTempInfoPo.class);
        Map<String, ProtocolTempInfoPo> map = new HashMap<>();
        if (null == list) {
            return map;
        }
        for (ProtocolTempInfoPo protocolTempInfoPo : list) {
            map.put((protocolTempInfoPo.getProtocolKey()), protocolTempInfoPo);
        }
        return map;
    }

    @Override
    public void removeProtocolTempInfo(String key) {
        dbService.delete(ProtocolStorageConstant.PROTOCOL_TEMP_AREA, key.getBytes());
    }

    @Override
    public Result saveChangeTxHashBlockHeight(Long effectiveHeight) {
        return dbService.put(ProtocolStorageConstant.NULS_VERSION_AREA,
ProtocolStorageConstant.CHANGE_HASH_HEIGHT_KEY, Util.longToBytes(effectiveHeight));

    }

    @Override
    public Long getChangeTxHashBlockHeight() {
        byte[] height = dbService.get(ProtocolStorageConstant.NULS_VERSION_AREA,
ProtocolStorageConstant.CHANGE_HASH_HEIGHT_KEY);
        return height == null ? null : Long.valueOf(Util.byteToInt(height));
    }
```

```
    @Override
    public void deleteChangeTxHashBlockHeight() {
        dbService.delete(ProtocolStorageConstant.NULS_VERSION_AREA,
ProtocolStorageConstant.CHANGE_HASH_HEIGHT_KEY);
    }

}

157:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
storage\src\main\java\io\nuls\protocol\storage\service\VersionManagerStorageService.java
 */
package io.nuls.protocol.storage.service;

import io.nuls.kernel.model.Result;
import io.nuls.protocol.storage.po.BlockProtocolInfoPo;
import io.nuls.protocol.storage.po.ProtocolInfoPo;
import io.nuls.protocol.storage.po.ProtocolTempInfoPo;

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

/**
 * @author: Charlie
 * @date: 2018/8/17
 */
public interface VersionManagerStorageService {

    /**
     *
     * Save the version currently running on the main net
     *
     * @param version
     * @return Result
     */
    Result saveMainVersion(int version);

    /**
     *
     * Gets the version currently running on the main net
     *
     * @return version
     */
```

```
Integer getMainVersion();

/**
 *
 * @param protocolInfoPo
 * @return Result
 */
Result saveProtocolInfoPo(ProtocolInfoPo protocolInfoPo);

/**
 *
 * @param version
 * @return ProtocolInfoPo
 */
ProtocolInfoPo getProtocolInfoPo(int version);

/**
 *
 * Save the number of nodes that have upgraded the new version of the program
 *
 * @param tempInfoPo
 * @return Result
 */
Result saveProtocolTempInfoPo(ProtocolTempInfoPo tempInfoPo);

/**
 *
 * Gets the number of nodes that have upgraded the new version of the program
 *
 * @param key
 * @return ProtocolTempInfoPo
 */
ProtocolTempInfoPo getProtocolTempInfoPo(String key);

Result saveBlockProtocolInfoPo(BlockProtocolInfoPo protocolInfoPo);

List<Long> getBlockProtocolIndex(int version);

List<Long> getBlockTempProtocolIndex(int version);

void saveBlockProtocolIndex(int version, List<Long> list);
```

```
    void saveTempBlockProtocolIndex(int version, List<Long> list);

    BlockProtocolInfoPo getBlockProtocolInfoPo(long blockHeight);

    BlockProtocolInfoPo getBlockTempProtocolInfoPo(long blockHeight);

    void clearBlockProtocol(long blockHeight, int version);

    void clearTempBlockProtocol(long blockHeight, int version);

    Result saveBlockProtocolTempInfoPo(BlockProtocolInfoPo protocolInfoPo);
    /**
     *
     * @return
     */
    Map<String,ProtocolTempInfoPo> getProtocolTempMap();

    void removeProtocolTempInfo(String key);

    Result saveChangeTxHashBlockHeight(Long effectiveHeight);

    Long getChangeTxHashBlockHeight();

    void deleteChangeTxHashBlockHeight();

    void deleteBlockProtocol(long blockHeight);

    void deleteBlockTempProtocol(long blockHeight);

    Result saveConsensusVersionMap(Map<String, Integer> versionMap);

    Map<String, Integer> getConsensusVersionMap();

    Result saveConsensusVersionHeight(Long blockHeight);

    Long getConsensusVersionHeight();
}

158:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-
storage\src\test\java\io\nuls\protocol\storage\po\BlockHeaderPoTest.java
 */
```

```
package io.nuls.protocol.storage.po;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;

import io.nuls.kernel.script.BlockSignature;
import org.junit.Test;

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

import static org.junit.Assert.*;

/**
 *
 * Block header model unit test tool class.
 *
 * @author: Niels Wang
 */
public class BlockHeaderPoTest {

    /**
     *
     * Verify the correctness of serialization and deserialization of block header entities.
     */
    @Test
    public void serializeAndParse() {
        BlockHeaderPo po = new BlockHeaderPo();
        po.setHeight(1286L);
        po.setExtend("extends".getBytes());
        po.setMerkleHash(NulsDigestData.calcDigestData("merkleHash".getBytes()));
        try {
            po.setPackingAddress("address".getBytes());
        } catch (Exception e) {
            e.printStackTrace();
            assertTrue(false);
        }
        po.setScriptSign(new BlockSignature());
        po.setTime(12345678901L);
```

```
        po.setTxCount(3);
        List<NulsDigestData> txHashList = new ArrayList<>();
        txHashList.add(NulsDigestData.calcDigestData("first-tx-hash".getBytes()));
        txHashList.add(NulsDigestData.calcDigestData("second-tx-hash".getBytes()));
        txHashList.add(NulsDigestData.calcDigestData("third-tx-hash".getBytes()));
        po.setTxHashList(txHashList);

        byte[] bytes = new byte[0];
        try {
            bytes = po.serialize();
        } catch (IOException e) {
            Log.error(e);
        }

        BlockHeaderPo newPo = new BlockHeaderPo();
        try {
            newPo.parse(bytes,0);
        } catch (NulsException e) {
            Log.error(e);
        }
        assertNull(newPo.getHash());
        assertEquals(po.getHeight(), newPo.getHeight());
        assertEquals(po.getPreHash(), newPo.getPreHash());
        assertEquals(po.getMerkleHash(), newPo.getMerkleHash());
        assertTrue(Arrays.equals(po.getExtend(), newPo.getExtend()));
        assertTrue(Arrays.equals(po.getPackingAddress(), newPo.getPackingAddress()));
        assertEquals(po.getScriptSign().getPublicKey(), newPo.getScriptSign().getPublicKey());
        assertEquals(po.getScriptSign().getSignData(), newPo.getScriptSign().getSignData());
        assertEquals(po.getTime(), newPo.getTime());
        assertEquals(po.getTxCount(), newPo.getTxCount());
        assertEquals(po.getTxHashList().get(0), newPo.getTxHashList().get(0));
        assertEquals(po.getTxHashList().get(1), newPo.getTxHashList().get(1));
        assertEquals(po.getTxHashList().get(2), newPo.getTxHashList().get(2));
    }

}

159:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\base\protocol-storage\src\test\java\io\nuls\protocol\storage\service\impl\BlockHeaderStorageServiceImplTest.java
 */
```

```
package io.nuls.protocol.storage.service.impl;

import io.nuls.db.module.impl.LevelDbModuleBootstrap;
import io.nuls.kernel.MicroKernelBootstrap;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;

import io.nuls.kernel.script.BlockSignature;
import io.nuls.protocol.storage.po.BlockHeaderPo;
import io.nuls.protocol.storage.service.BlockHeaderStorageService;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

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

import static org.junit.Assert.*;

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

    private BlockHeaderStorageService service;

    private BlockHeaderPo entity;

    @Before
    public void init() {
        MicroKernelBootstrap mk = MicroKernelBootstrap.getInstance();
        mk.init();
        mk.start();

        LevelDbModuleBootstrap bootstrap = new LevelDbModuleBootstrap();
        bootstrap.init();
        bootstrap.start();

        service = NulsContext.getServiceBean(BlockHeaderStorageService.class);
        BlockHeaderPo po = new BlockHeaderPo();
```

```
    po.setHash(NulsDigestData.calcDigestData("hashhash".getBytes()));
    po.setHeight(1286L);
    po.setExtend("extends".getBytes());
    po.setMerkleHash(NulsDigestData.calcDigestData("merkleHash".getBytes()));
    po.setPreHash(NulsDigestData.calcDigestData("prehash".getBytes()));
    try {
        po.setPackingAddress("address".getBytes());
    } catch (Exception e) {
        e.printStackTrace();
        assertTrue(false);
    }
    po.setScriptSign(new BlockSignature());
    po.setTime(12345678901L);
    po.setTxCount(3);
    List<NulsDigestData> txHashList = new ArrayList<>();
    txHashList.add(NulsDigestData.calcDigestData("first-tx-hash".getBytes()));
    txHashList.add(NulsDigestData.calcDigestData("second-tx-hash".getBytes()));
    txHashList.add(NulsDigestData.calcDigestData("third-tx-hash".getBytes()));
    po.setTxHashList(txHashList);
    this.entity = po;
}

@Test
public void test() {
    assertNotNull(service);
    this.saveBlockHeader();

    this.getBlockPo();

    this.getBlockPo1();

    this.removeBlockHerader();
}

public void getBlockPo() {
    BlockHeaderPo po = this.service.getBlockHeaderPo(entity.getHeight());
    this.testEquals(po, entity);
}

public void getBlockPo1() {
    BlockHeaderPo po = this.service.getBlockHeaderPo(entity.getHash());
    this.testEquals(po, entity);
```

```
    }

    public void saveBlockHeader() {
        Result result = service.saveBlockHeader(entity);
        assertTrue(result.isSuccess());
    }

    public void removeBlockHerader() {
        service.removeBlockHerader(entity);
        BlockHeaderPo po = this.service.getBlockHeaderPo(entity.getHash());
        assertNull(po);
    }


    private void testEquals(BlockHeaderPo po, BlockHeaderPo entity) {
        assertEquals(po.getHash(),entity.getHash());
        assertEquals(po.getHeight(), entity.getHeight());
        assertEquals(po.getPreHash(), entity.getPreHash());
        assertEquals(po.getMerkleHash(), entity.getMerkleHash());
        assertTrue(Arrays.equals(po.getExtend(), entity.getExtend()));
        assertTrue(Arrays.equals(po.getPackingAddress(), entity.getPackingAddress()));
        assertEquals(po.getScriptSign().getPublicKey(), entity.getScriptSign().getPublicKey());
        assertEquals(po.getScriptSign().getSignData(), entity.getScriptSign().getSignData());
        assertEquals(po.getTime(), entity.getTime());
        assertEquals(po.getTxCount(), entity.getTxCount());
        assertEquals(po.getTxHashList().get(0), entity.getTxHashList().get(0));
        assertEquals(po.getTxHashList().get(1), entity.getTxHashList().get(1));
        assertEquals(po.getTxHashList().get(2), entity.getTxHashList().get(2));
    }

}

160:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\cache\TemporaryCacheManager.java
 */
package io.nuls.protocol.cache;

import io.nuls.cache.LimitHashMap;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.protocol.model.SmallBlock;
```

```
/**
 * Used for sharing temporary data between multiple hander.
 * handler
 *
 * @author Niels
 */
public class TemporaryCacheManager {
    private static final TemporaryCacheManager INSTANCE = new TemporaryCacheManager();

    private LimitHashMap<NulsDigestData, SmallBlock> smallBlockCacheMap = new
LimitHashMap<>(100);
    private LimitHashMap<NulsDigestData, NulsDigestData> smallBlockHashCacheMap = new
LimitHashMap<>(100);
//    private CacheMap<NulsDigestData, Transaction> txCacheMap = new CacheMap<>("temp-tx-
cache", 128, NulsDigestData.class, Transaction.class, 0, 3600);

    private LimitHashMap<NulsDigestData, Transaction> txCacheMap = new
LimitHashMap<>(100000);

    private TemporaryCacheManager() {

    }

    public static TemporaryCacheManager getInstance() {
        return INSTANCE;
    }

    /**
     * SmallBlock1000
     * Store a SmallBlock in memory, cache it full or exist for over 1000 seconds, and clean it
automatically.
     *
     * @param smallBlock
     */
    public void cacheSmallBlock(SmallBlock smallBlock) {
        smallBlockCacheMap.put(smallBlock.getHeader().getHash(), smallBlock);
    }

    public void cacheSmallBlockWithRequest(NulsDigestData requestHash, SmallBlock smallBlock)
{
        NulsDigestData blockHash = smallBlock.getHeader().getHash();
        smallBlockHashCacheMap.put(requestHash, blockHash);
```

```
        smallBlockCacheMap.put(blockHash, smallBlock);
    }

    /**
     * hashSmallBlock
     * get SmallBlock by block header digest data
     *
     * @param requestHash getTxGroupRequestHash
     * @return SmallBlock
     */
    public SmallBlock getSmallBlockByRequest(NulsDigestData requestHash) {

        return getSmallBlockByHash(smallBlockHashCacheMap.get(requestHash));
    }
    public SmallBlock getSmallBlockByHash(NulsDigestData blockHash) {

        return smallBlockCacheMap.get(blockHash);
    }

    /**
     * hash1000
     * Cache a transaction where the identity of the cache is the hash object of the transaction,
     * which exists in memory until the memory size is limited or survived for more than 1000
seconds.
     *
     * @param tx transaction
     */
    public boolean cacheTx(Transaction tx) {
        return txCacheMap.put(tx.getHash(), tx);
    }

    /**
     * hash
     * get whole transaction from cache by transaction digest data
     *
     * @param hash transaction digest data
     * @return whole transaction
     */
    public Transaction getTx(NulsDigestData hash) {
        if (null == txCacheMap) {
            return null;
        }
```

```
        return txCacheMap.get(hash);
    }

    /**
     * SmallBlocknull
     * A SmallBlock is removed from the cache based on the block summary object, and null is returned when it is removed.
     *
     * @param hash transaction digest data
     */
    public void removeSmallBlock(NulsDigestData hash) {
        if (null == smallBlockCacheMap) {
            return;
        }
        smallBlockCacheMap.remove(hash);
    }


    /**
     *
     * Empty all cached data.
     */
    public void clear() {
        this.smallBlockCacheMap.clear();
        this.txCacheMap.clear();
    }

    /**
     *
     * destroy cache
     */
    public void destroy() {
        this.smallBlockCacheMap.clear();
        this.txCacheMap.clear();
    }

    public boolean containsTx(NulsDigestData txHash) {
        return txCacheMap.containsKey(txHash);
    }

    public int getSmallBlockCount() {
        return smallBlockCacheMap.size();
```

```
    }

    public int getTxCount() {
        return txCacheMap.size();
    }
}

161:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\constant\MessageDataType.java
 */

package io.nuls.protocol.constant;

/**
 *  {@link io.nuls.protocol.model.NotFound}
 * NotFound
 * <p>
 * For network data acquisition, data obtained when peer node cannot be found, returns {@link
io.nuls.protocol.model.NotFound}  as a result,
 * NotFound results are divided into several categories according to the data obtained, and specific
categories are defined in this class.
 *
 * @author: Niels Wang
 */
public enum MessageDataType {
    /**
     * Not Found
     * When the block cannot be Found, the returned Not Found type.
     */
    BLOCK(1),

    /**
     * Not Found
     * When the block cannot be Found, the returned Not Found type.
     */
    BLOCKS(2),
    /**
     * Not Found
     * When the transactions cannot be Found, the returned Not Found type.
     */
//    TRANSACTIONS(3),
    /**
```

```
     * Not Found
     * When the block header digest data cannot be Found, the returned Not Found type.
     */
    HASHES(4),

//    SMALL_BLOCK(5),
//
//    TRANSACTION(6),

    REQUEST(7),

    ;

    /**
     * code{@link io.nuls.protocol.model.NotFound}
     * type codefor serialize of {@link io.nuls.protocol.model.NotFound}
     */
    private final int code;

    MessageDataType(int code) {
        this.code = code;
    }

    /**
     *
     * Gets the code that the type corresponds to.
     *
     * @return int
     */
    public int getCode() {
        return code;
    }

    /**
     * Get Enum by type code
     *
     *
     * @param code int type code
     * @return {@link MessageDataType}
     */
    public static MessageDataType getType(int code) {
        switch (code) {
```

```
            case 1:
                return BLOCK;
            case 2:
                return BLOCKS;
//              case 3:
//                  return TRANSACTIONS;
            case 4:
                return HASHES;
            default:
                return null;
        }
    }
}
```

162:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\constant\ProtocolConstant.java
*/

```
package io.nuls.protocol.constant;

import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.model.Na;

/**
 *
 * The relevant constants of the protocol and some general constants are defined here.
 *
 * @author: Niels Wang
 */
public interface ProtocolConstant extends NulsConstant {

    /**
     *
     * Block interval time.
     * unit:second
     */
    long BLOCK_TIME_INTERVAL_SECOND = 10;

    /**
     *
     * Block interval time.
     * unit:millis
```

```
 */
long BLOCK_TIME_INTERVAL_MILLIS = BLOCK_TIME_INTERVAL_SECOND * 1000L;

/**
 * id
 * module id of the protocol module
 */
short MODULE_ID_PROTOCOL = 3;

/**
 *
 * The number of minimum connection nodes that the system runs.
 */
int ALIVE_MIN_NODE_COUNT = 1;

/**
 *
 * Maximum block size (excluding block headers)
 */
long MAX_BLOCK_SIZE = 2 * 1024 * 1024L;

/**
 *
 * All message type definitions for the protocol module.
 * ==========================================================================
 */
/**
 * ""
 * The data cannot find the type of answer.
 */
short PROTOCOL_NOT_FOUND = 1;
/**
 *
 * The type of message that the new transaction sends and forwards.
 */
short PROTOCOL_NEW_TX = 2;
/**
 *
 * Gets the type of message for the block.
 */
short PROTOCOL_GET_BLOCK = 3;
/**
```

```
 *
 * The type of message to send the block.
 */
short PROTOCOL_BLOCK = 4;
/**
 * hash
 * The type of message to get the blocks by hash.
 */
short PROTOCOL_GET_BLOCKS_BY_HASH = 5;

/**
 *
 * The type of message to get the blocks by height.
 */
short PROTOCOL_GET_BLOCKS_BY_HEIGHT = 6;
/**
 *
 * Gets the type of message for the block-header.
 */
short PROTOCOL_GET_BLOCK_HEADER = 7;
/**
 *
 * The type of message to send the block-header.
 */
short PROTOCOL_BLOCK_HEADER = 8;
/**
 *
 * Gets the type of message for the transactions.
 */
short PROTOCOL_GET_TX_GROUP = 9;
/**
 *
 * The type of message to send the transactions.
 */
short PROTOCOL_TX_GROUP = 10;
/**
 *
 * The type of message that the new SmallBlock sends and forwards.
 */
short PROTOCOL_NEW_BLOCK = 11;
/**
 * hashhash
```

```
     * Gets the type of message for the Blocks hashes.
     */
    short PROTOCOL_GET_BLOCKS_HASH = 12;
    /**
     * hash
     * The type of message to send the Blocks hashes.
     */
    short PROTOCOL_BLOCKS_HASH = 13;
    /**
     *
     * The type of message that is sent to a peer.
     */
    short PROTOCOL_STRING = 14;
    /**
     *
     * task complete message
     */
    short PROTOCOL_COMPLETE = 15;
    /**
     *
     * Request reply message type, used to immediately know if the target node received this
request
     */
    short PROTOCOL_REQUEST_REACT = 16;
    short PROTOCOL_FORWARD_NEW_TX = 17;
    short PROTOCOL_FORWARD_NEW_BLOCK = 18;
    short PROTOCOL_GET_SMALL_BLOCK = 19;
    short PROTOCOL_GET_TRANSACTION = 20;
    //
    int MIN_PROTOCOL_UPGRADE_RATE = 60;
    //
    int MIN_PROTOCOL_UPGRADE_DELAY = 1000;
    // pierre test comment out

    /**
     *
     * Minimum transfer amount
     */
    Na MININUM_TRANSFER_AMOUNT = Na.parseNuls(0.0001);

}
```

163:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\constant\ProtocolErroeCode.java

```
 */
package io.nuls.protocol.constant;

import io.nuls.kernel.constant.ErrorCode;
import io.nuls.kernel.constant.KernelErrorCode;

/**
 * @author: Charlie
 * @date: 2018/8/9
 */
public interface ProtocolErroeCode extends KernelErrorCode {

    ErrorCode BLOCK_HEADER_SIGN_CHECK_FAILED= ErrorCode.init("30001");
    ErrorCode BLOCK_HEADER_FIELD_CHECK_FAILED= ErrorCode.init("30002");
    ErrorCode BLOCK_FIELD_CHECK_FAILED= ErrorCode.init("30003");
    ErrorCode BLOCK_TOO_BIG= ErrorCode.init("30004");
    ErrorCode MERKLE_HASH_WRONG= ErrorCode.init("30005");

}
```

164:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\base\BaseMessage.java

```
package io.nuls.protocol.message.base;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;

import java.io.IOException;

/**
 *
 * The base class for all messages transmitted over the network defines the basic format of the network message.
 *
 * @author Niels
 */
```

```
public abstract class BaseMessage<T extends BaseNulsData> extends BaseNulsData {

    private transient NulsDigestData hash;

    private MessageHeader header;

    private T msgBody;

    public BaseMessage() {

    }

//    /**
//     *
//     */
    public BaseMessage(short moduleId, short msgType) {
        this.header = new MessageHeader(moduleId, msgType);
    }


//    /**
//     * serialize important field
//     */
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.write(header.serialize());
        stream.write(msgBody.serialize());
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        MessageHeader header = new MessageHeader();
        header.parse(byteBuffer);
        this.header = header;
        this.msgBody = parseMessageBody(byteBuffer);
    }

    protected abstract T parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException;

    @Override
    public int size() {
        int s = 0;
```

```
    s += header.size();
    s += msgBody.size();
    return s;
}

/**
 * msgBody
 * The verification value of msgBody is calculated,
 * and the result is put into the message header through a simple difference or result.
 *
 * @return ,Verification value (calculation result)
 */
public byte caculateXor() {
    if (header == null || msgBody == null) {
        return 0x00;
    }
    byte xor = 0x00;
    byte[] data = new byte[0];
    try {
        data = msgBody.serialize();
    } catch (IOException e) {
        Log.error(e);
    }
    for (int i = 0; i < data.length; i++) {
        xor ^= data[i];
    }
    header.setXor(xor);
    return xor;
}

public T getMsgBody() {
    return msgBody;
}

public void setMsgBody(T msgBody) {
    this.msgBody = msgBody;
}

public MessageHeader getHeader() {
    return header;
}
```

```
    public void setHeader(MessageHeader header) {
        this.header = header;
    }

    public NulsDigestData getHash() {
        if (hash == null) {
            try {
                this.hash = NulsDigestData.calcDigestData(this.serialize());
            } catch (IOException e) {
                Log.error(e);
            }
        }
        return hash;
    }

}

165:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\base\CommonStringMessage.java
 */

package io.nuls.protocol.message.base;

import io.nuls.core.tools.str.StringUtils;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.message.BaseProtocolMessage;
import io.nuls.protocol.model.basic.NulsStringData;

/**
 *
 * The message body has only a string of message classes.
 *
 * @author Niels
 */
public class CommonStringMessage extends BaseProtocolMessage<NulsStringData> {

    public CommonStringMessage() {
        super(ProtocolConstant.PROTOCOL_STRING);
    }
```

```
    @Override
    protected NulsStringData parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException
{
        return byteBuffer.readNulsData(new NulsStringData());
    }

    public void setMessage(String message) {
        if (StringUtils.isBlank(message)) {
            return;
        }
        this.setMsgBody(new NulsStringData(message));
    }

    public String getMessage() {
        NulsStringData data = this.getMsgBody();
        if (null == data) {
            return null;
        }
        return data.getVal();
    }
}

166:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\base\MessageHeader.java
 */
package io.nuls.protocol.message.base;

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.protostuff.Tag;

import java.io.IOException;

/**
 * id
 * Network message header, the message header contains
 * magic parameters, message body size, parity, encryption algorithm id, module id, message type
```

```
information.
 *
 * @author Niels
 */
public class MessageHeader extends BaseNulsData {
    /**
     *
     * Magic parameters used in the isolation section.
     */

    private long magicNumber;
    /**
     *
     * the length of the msgBody
     */

    private int length;
    /**
     *
     * Parity bit for the parity of the message body.
     */

    private byte xor;
    /**
     *
     * Encryption algorithm identification
     */

    private byte arithmetic;
    /**
     * id
     */

    private short moduleId;
    /**
     *
     */

    private short msgType;

    public MessageHeader() {
    }
```

```
public MessageHeader(short moduleId, short msgType) {
    this.moduleId = moduleId;
    this.msgType = msgType;
}

/**
 * serialize important field
 */
@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeUint32(magicNumber);
    stream.writeUint32(length);
    stream.write(xor);
    stream.write(arithmetic);
    stream.writeUint16(moduleId);
    stream.writeUint16(msgType);
}

@Override
public void parse(NulsByteBuffer buffer) throws NulsException {
    magicNumber = buffer.readUint32();
    length = (int) buffer.readUint32();
    xor = buffer.readByte();
    arithmetic = buffer.readByte();
    moduleId = (short) buffer.readUint16();
    msgType = (short) buffer.readUint16();
}

@Override
public int size() {
    int s = 0;
    s += SerializeUtils.sizeOfUint32();
    s += SerializeUtils.sizeOfUint32();
    s += 1;
    s += 1;
    s += SerializeUtils.sizeOfUint16();
    s += SerializeUtils.sizeOfUint16();
    return s;
}

public short getMsgType() {
```

```
    return msgType;
}

public void setMsgType(short msgType) {
    this.msgType = msgType;
}

public short getModuleId() {
    return moduleId;
}

public void setModuleId(short moduleId) {
    this.moduleId = moduleId;
}

public long getMagicNumber() {
    return magicNumber;
}

public void setMagicNumber(long magicNumber) {
    this.magicNumber = magicNumber;
}

public int getLength() {
    return length;
}

public void setLength(int length) {
    this.length = length;
}

public byte getXor() {
    return xor;
}

public void setXor(byte xor) {
    this.xor = xor;
}

public byte getArithmetic() {
    return arithmetic;
}
```

```
    public void setArithmetic(byte arithmetic) {
        this.arithmetic = arithmetic;
    }

}
```

167:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\BaseProtocolMessage.java

```
 */
package io.nuls.protocol.message;

import io.nuls.kernel.model.BaseNulsData;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.message.base.BaseMessage;

/**
 *
 * The protocol module message base class is used to normalize all messages for this module.
 *
 * @author Niels
 */
public abstract class BaseProtocolMessage<T extends BaseNulsData> extends BaseMessage<T> {

    /**
     * id
     * The constructor defaults to the module id, and the implementer only needs to focus on the message type.
     *
     * @param messageType
     */
    public BaseProtocolMessage(short messageType) {
        super(ProtocolConstant.MODULE_ID_PROTOCOL, messageType);
    }

}
```

168:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\BlockHeaderMessage.java

```
 */
package io.nuls.protocol.message;
```

```
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 *
 * The host class of the block header in the network message.
 *
 * @author Niels
 */
public class BlockHeaderMessage extends BaseProtocolMessage<BlockHeader> {
    public BlockHeaderMessage() {
        super(ProtocolConstant.PROTOCOL_BLOCK_HEADER);
    }

    @Override
    protected BlockHeader parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new BlockHeader());
    }
}
```

169:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\BlockMessage.java

```
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 *
 * The host class of the block in the network message.
 *
 * @author Niels
 */
public class BlockMessage extends BaseProtocolMessage<Block> {
    public BlockMessage() {
        super(ProtocolConstant.PROTOCOL_BLOCK);
    }
```

```
    @Override
    protected Block parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new Block());
    }
}

170:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\BlocksHashMessage.java
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.BlockHashResponse;

/**
 * @author Niels
 */
public class BlocksHashMessage extends BaseProtocolMessage<BlockHashResponse> {

    public BlocksHashMessage() {
        super(ProtocolConstant.PROTOCOL_BLOCKS_HASH);
    }

    @Override
    protected BlockHashResponse parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new BlockHashResponse());
    }
}

171:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\CompleteMessage.java
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.CompleteParam;
```

```
/**
 * @author ln
 */
public class CompleteMessage extends BaseProtocolMessage<CompleteParam> {

    public CompleteMessage() {
        super(ProtocolConstant.PROTOCOL_COMPLETE);
    }

    @Override
    protected CompleteParam parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new CompleteParam());
    }
}

172:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\ForwardSmallBlockMessage.java
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 *
 * The message for send a new transaction
 *
 * @author Niels
 */
public class ForwardSmallBlockMessage extends BaseProtocolMessage<NulsDigestData> {

    public ForwardSmallBlockMessage() {
        super(ProtocolConstant.PROTOCOL_FORWARD_NEW_BLOCK);
    }

    @Override
    protected NulsDigestData parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
```

```
        return byteBuffer.readHash();
    }
}
```

173:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\ForwardTxMessage.java

```
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 *
 * The message for send a new transaction
 *
 * @author Niels
 */
public class ForwardTxMessage extends BaseProtocolMessage<NulsDigestData> {

    public ForwardTxMessage() {
        super(ProtocolConstant.PROTOCOL_FORWARD_NEW_TX);
    }

    @Override
    protected NulsDigestData parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readHash();
    }
}
```

174:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\GetBlockHeaderRequest.java

```
// */
//package io.nuls.protocol.message;
//
//import io.nuls.kernel.exception.NulsException;
//import io.nuls.kernel.model.NulsDigestData;
//import io.nuls.kernel.utils.NulsByteBuffer;
//import io.nuls.protocol.constant.ProtocolConstant;
```

```
//import io.nuls.protocol.model.GetBlocksByHashParam;
//
///**
// *
// * The message for get block-header of block-headers
// *
// * @author Niels
// */
//public class GetBlockHeaderRequest extends BaseProtocolMessage<GetBlocksByHashParam>
{
//
//    public GetBlockHeaderRequest() {
//        super(ProtocolConstant.MESSAGE_TYPE_GET_BLOCK_HEADER);
//    }
//
//    @Override
//    protected GetBlocksByHashParam parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
//        return byteBuffer.readNulsData(new GetBlocksByHashParam());
//    }
//
//    public GetBlockHeaderRequest(long start, long size) {
//        this();
//        GetBlocksByHashParam param = new GetBlocksByHashParam();
//        param.setSize(size);
//        param.setStart(start);
//        this.setMsgBody(param);
//    }
//
//    public GetBlockHeaderRequest(long start, long size, NulsDigestData startHash,
NulsDigestData endHash) {
//        this();
//        GetBlocksByHashParam param = new GetBlocksByHashParam();
//        param.setSize(size);
//        param.setStart(start);
//        param.setStartHash(startHash);
//        param.setEndHash(endHash);
//        this.setMsgBody(param);
//    }
//
//    public long getStart() {
//        if (null == this.getMsgBody()) {
```

```
//            return -1;
//        }
//        return this.getMsgBody().getStart();
//    }
//
//    public long getSize() {
//        if (null == this.getMsgBody()) {
//            return -1;
//        }
//        return this.getMsgBody().getSize();
//    }
//
//
//    public NulsDigestData getStartHash() {
//        if (null == this.getMsgBody()) {
//            return null;
//        }
//        return this.getMsgBody().getStartHash();
//    }
//
//    public NulsDigestData getEndHash() {
//        if (null == this.getMsgBody()) {
//            return null;
//        }
//        return this.getMsgBody().getEndHash();
//    }
//
//}
```

175:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\GetBlockMessage.java

```
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.GetBlockParam;

/**
 *
 * The message for get block or blocks
```

```
 *
 * @author Niels
 */
public class GetBlockMessage extends BaseProtocolMessage<GetBlockParam> {

    public GetBlockMessage() {
        super(ProtocolConstant.PROTOCOL_GET_BLOCK);
    }

    @Override
    protected GetBlockParam parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new GetBlockParam());
    }

    public GetBlockMessage(NulsDigestData hash) {
        this();
        GetBlockParam param = new GetBlockParam();
        param.setBlockHash(hash);
        this.setMsgBody(param);
    }

    public NulsDigestData getBlockHash() {
        if (null == this.getMsgBody()) {
            return null;
        }
        return this.getMsgBody().getBlockHash();
    }
}
```

176:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\GetBlocksByHashMessage.java

```
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.GetBlocksByHashParam;

/**
 *
```

```
 * The message for get block or blocks
 *
 * @author Niels
 */
public class GetBlocksByHashMessage extends
BaseProtocolMessage<GetBlocksByHashParam> {

    public GetBlocksByHashMessage() {
        super(ProtocolConstant.PROTOCOL_GET_BLOCKS_BY_HASH);
    }

    public GetBlocksByHashMessage(NulsDigestData startHash, NulsDigestData endHash) {
        this();
        GetBlocksByHashParam param = new GetBlocksByHashParam(startHash, endHash);
        setMsgBody(param);
    }

    @Override
    protected GetBlocksByHashParam parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new GetBlocksByHashParam());
    }
}
```

177:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\GetBlocksByHeightMessage.java

```
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.GetBlocksByHeightParam;

/**
 *
 * The message for get block or blocks
 *
 * @author Niels
 */
public class GetBlocksByHeightMessage extends
BaseProtocolMessage<GetBlocksByHeightParam> {
```

```
    public GetBlocksByHeightMessage() {
        super(ProtocolConstant.PROTOCOL_GET_BLOCKS_BY_HEIGHT);
    }

    public GetBlocksByHeightMessage(long startHeight, long endHeight) {
        this();
        GetBlocksByHeightParam param = new GetBlocksByHeightParam(startHeight, endHeight);
        setMsgBody(param);
    }

    @Override
    protected GetBlocksByHeightParam parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new GetBlocksByHeightParam());
    }
}

178:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\GetBlocksHashMessage.java
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.GetBlocksHashParam;

/**
 *
 * The message of gets the block header summary list from the peer node.
 *
 * @author Niels
 */
public class GetBlocksHashMessage extends BaseProtocolMessage<GetBlocksHashParam> {

    public GetBlocksHashMessage() {
        super(ProtocolConstant.PROTOCOL_GET_BLOCKS_HASH);
    }

    @Override
    protected GetBlocksHashParam parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
```

```
        return byteBuffer.readNulsData(new GetBlocksHashParam());
    }

    public GetBlocksHashMessage(long startHeight, long endHeight) {
        this();
        GetBlocksHashParam param = new GetBlocksHashParam(startHeight, endHeight);
        this.setMsgBody(param);
    }
}
```

179:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\GetSmallBlockMessage.java

```
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.GetBlockParam;

/**
 *
 * The message for get block or blocks
 *
 * @author Niels
 */
public class GetSmallBlockMessage extends BaseProtocolMessage<NulsDigestData> {

    public GetSmallBlockMessage() {
        super(ProtocolConstant.PROTOCOL_GET_SMALL_BLOCK);
    }

    @Override
    protected NulsDigestData parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readHash();
    }

}
```

180:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-

```
module\protocol\src\main\java\io\nuls\protocol\message\GetTxGroupRequest.java
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.GetTxGroupParam;

/**
 *
 * The message of gets the transaction list from the peer node.
 *
 * @author Niels
 */
public class GetTxGroupRequest extends BaseProtocolMessage<GetTxGroupParam> {

    public GetTxGroupRequest() {
        super(ProtocolConstant.PROTOCOL_GET_TX_GROUP);
    }

    @Override
    protected GetTxGroupParam parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readNulsData(new GetTxGroupParam());
    }

}

181:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\GetTxMessage.java
 */
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 *
 * The message for get block or blocks
```

```
 *
 * @author Niels
 */
public class GetTxMessage extends BaseProtocolMessage<NulsDigestData> {

    public GetTxMessage() {
        super(ProtocolConstant.PROTOCOL_GET_TRANSACTION);
    }

    @Override
    protected NulsDigestData parseMessageBody(NulsByteBuffer byteBuffer) throws
NulsException {
        return byteBuffer.readHash();
    }

}

182:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\NotFoundMessage.java
 */

package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.NotFound;

/**
 * ""
 * "Unable to find" feedback on the host class in the network message.
 *
 * @author: Niels Wang
 */
public class NotFoundMessage extends BaseProtocolMessage<NotFound> {
    public NotFoundMessage() {
        super(ProtocolConstant.PROTOCOL_NOT_FOUND);
    }

    @Override
    protected NotFound parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new NotFound());
```

```
    }
}
```

183:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\ReactMessage.java

```
 */

package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.ReactParam;

/**
 * @author ln
 */
public class ReactMessage extends BaseProtocolMessage<ReactParam> {

    public ReactMessage() {
        super(ProtocolConstant.PROTOCOL_REQUEST_REACT);
    }

    public ReactMessage(NulsDigestData requestId) {
        super(ProtocolConstant.PROTOCOL_REQUEST_REACT);
        setMsgBody(new ReactParam(requestId));
    }

    @Override
    protected ReactParam parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new ReactParam());
    }
}
```

184:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\SmallBlockMessage.java

```
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
```

```
import io.nuls.protocol.model.SmallBlock;

/**
 *
 * The message for send new SmallBlock;
 *
 * @author Niels
 */
public class SmallBlockMessage extends BaseProtocolMessage<SmallBlock> {
    public SmallBlockMessage() {
        super(ProtocolConstant.PROTOCOL_NEW_BLOCK);
    }

    @Override
    protected SmallBlock parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new SmallBlock());
    }

    public SmallBlockMessage(SmallBlock newBlock) {
        this();
        this.setMsgBody(newBlock);
    }


}

185:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\message\TransactionMessage.java
 */
package io.nuls.protocol.message;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.TransactionManager;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 *
 * The message for send a new transaction
 *
```

```
 * @author Niels
 */
public class TransactionMessage extends BaseProtocolMessage<Transaction> {

    public TransactionMessage() {
        super(ProtocolConstant.PROTOCOL_NEW_TX);
    }

    @Override
    protected Transaction parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        try {
            return TransactionManager.getInstance(byteBuffer);
        } catch (Exception e) {
            Log.error(e);
            return null;
        }
    }
}
```

186:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\TxGroupMessage.java

```
package io.nuls.protocol.message;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.model.TxGroup;

/**
 *
 * When a peer requests a transaction, the message is answered, and the content is one or more transactions.
 *
 * @author Niels
 */
public class TxGroupMessage extends BaseProtocolMessage<TxGroup> {

    public TxGroupMessage() {
        super(ProtocolConstant.PROTOCOL_TX_GROUP);
    }

    @Override
```

```
    protected TxGroup parseMessageBody(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new TxGroup());
    }

}
```

187:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\message\validator\NulsMessageValidator.java

```
 */
package io.nuls.protocol.message.validator;

import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.network.constant.NetworkErrorCode;
import io.nuls.protocol.message.base.BaseMessage;

/**
 *
 * network message validator
 *
 * @author Niels
 */
@Component
public class NulsMessageValidator implements NulsDataValidator<BaseMessage> {

    /**
     *
     * Verify that the message body is not empty, the message body length is correct, and the check value is correct.
     *
     * @param data network message model
     * @return
     */
    @Override
    public ValidateResult validate(BaseMessage data) {
        if (data.getHeader() == null || data.getMsgBody() == null) {
            return ValidateResult.getFailedResult(this.getClass().getName(), NetworkErrorCode.NET_MESSAGE_ERROR);
        }

        if (data.getHeader().getLength() != data.getMsgBody().size()) {
```

```
            return ValidateResult.getFailedResult(this.getClass().getName(),
NetworkErrorCode.NET_MESSAGE_LENGTH_ERROR);
        }

        if (data.getHeader().getXor() != data.caculateXor()) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
NetworkErrorCode.NET_MESSAGE_XOR_ERROR);
        }
        return ValidateResult.getSuccessResult();
    }
}

188:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\basic\NulsBytesData.java
 */

package io.nuls.protocol.model.basic;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.model.BasicTypeData;

import java.io.IOException;

/**
 *
 * Controlled, byte array type encapsulation.
 *
 * @author Niels
 */
public class NulsBytesData extends BasicTypeData<byte[]> {

    public NulsBytesData() {
        this(null);
    }

    public NulsBytesData(byte[] val) {
        super(val);
    }
```

```
    @Override
    public int size() {
        return SerializeUtils.sizeOfBytes(getVal());
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeBytesWithLength(getVal());
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.setVal(byteBuffer.readByLengthByte());
    }
}

189:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\basic\NulsDoubleData.java
 */

package io.nuls.protocol.model.basic;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.model.BasicTypeData;

import java.io.IOException;

/**
 *
 * Controlled, double type encapsulation.
 *
 * @author Niels
 */
public class NulsDoubleData extends BasicTypeData<Double> {

    public NulsDoubleData() {
        this(null);
    }
```

```
    public NulsDoubleData(Double val) {
        super(val);
    }

    @Override
    public int size() {
        return SerializeUtils.sizeOfDouble(getVal());
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeDouble(getVal());
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.setVal(byteBuffer.readDouble());
    }
}

190:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\basic\NulsIntegerData.java
 */

package io.nuls.protocol.model.basic;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.model.BasicTypeData;

import java.io.IOException;

/**
 *
 * Controlled, int type encapsulation.
 *
 * @author Niels
 */
public class NulsIntegerData extends BasicTypeData<Integer> {
    public NulsIntegerData() {
```

```
        this(null);
    }

    public NulsIntegerData(Integer val) {
        super(val);
    }
    @Override
    public int size() {
        return SerializeUtils.sizeOfVarInt(getVal());
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeVarInt(getVal());
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.setVal((int) byteBuffer.readVarInt());
    }
}

191:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\basic\NulsLongData.java
 */

package io.nuls.protocol.model.basic;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.model.BasicTypeData;

import java.io.IOException;

/**
 *
 * Controlled, long type encapsulation.
 *
 * @author Niels
 */
```

```
public class NulsLongData extends BasicTypeData<Long> {
    public NulsLongData() {
        this(null);
    }

    public NulsLongData(Long val) {
        super(val);
    }

    @Override
    public int size() {
        return SerializeUtils.sizeOfVarInt(getVal());
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeVarInt(getVal());
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.setVal(byteBuffer.readVarInt());
    }
}

192:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\basic\NulsStringData.java
 */

package io.nuls.protocol.model.basic;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.model.BasicTypeData;

import java.io.IOException;

/**
 *
 * Controlled, string type encapsulation.
```

```
 *
 * @author Niels
 */
public class NulsStringData extends BasicTypeData<String> {

    public NulsStringData() {
        this(null);
    }

    public NulsStringData(String val) {
        super(val);
    }

    @Override
    public int size() {
        return SerializeUtils.sizeOfString(getVal());
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeString(getVal());
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.setVal(byteBuffer.readString());
    }
}

193:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\BasicTypeData.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.model.BaseNulsData;
import io.protostuff.Tag;

/**
 *
 * Controlled, Basic type encapsulation.
 *
 * @author Niels
```

```
 */
public abstract class BasicTypeData<T> extends BaseNulsData {

    private T val;

    public BasicTypeData(T data) {
        this.val = data;
    }

    public T getVal() {
        return val;
    }

    public void setVal(T val) {
        this.val = val;
    }
}
```

194:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\model\BlockHashResponse.java

```
 */
package io.nuls.protocol.model;

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

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

/**
 * hash
 * Block hash table reply data encapsulation.
 *
 * @author Niels
 */
public class BlockHashResponse extends BaseNulsData {
```

```
/**
 * hash
 * the digest data of the request message
 */
private NulsDigestData requestMessageHash;

/**
 * hash
 * Returns a list of hashes.
 */
private List<NulsDigestData> hashList = new ArrayList<>();

@Override
public int size() {
    int size = 0;
    size += SerializeUtils.sizeOfNulsData(requestMessageHash);
    size += SerializeUtils.sizeOfVarInt(hashList.size());
    for (NulsDigestData hash : hashList) {
        size += SerializeUtils.sizeOfNulsData(hash);
    }
    return size;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeNulsData(requestMessageHash);
    stream.writeVarInt(hashList.size());
    for (NulsDigestData hash : hashList) {
        stream.writeNulsData(hash);
    }
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.requestMessageHash = byteBuffer.readHash();
    long hashListSize = byteBuffer.readVarInt();
    if (hashListSize <= 0) {
        return;
    }
    this.hashList = new ArrayList<>();
    for (int i = 0; i < hashListSize; i++) {
        hashList.add(byteBuffer.readHash());
```

```
        }
    }


//    /**
//     * hash
//     * Returns a list of hashes.
//     */
    public List<NulsDigestData> getHashList() {
        return hashList;
    }

    public NulsDigestData getHash() {
        try {
            return NulsDigestData.calcDigestData(this.serialize());
        } catch (IOException e) {
            Log.error(e);
            return null;
        }
    }

    public void put(NulsDigestData hash) {
        hashList.add(hash);
    }

    public void putFront(NulsDigestData hash) {
        hashList.add(0, hash);
    }

    public NulsDigestData getRequestMessageHash() {
        return requestMessageHash;
    }

    public void setRequestMessageHash(NulsDigestData requestMessageHash) {
        this.requestMessageHash = requestMessageHash;
    }
}

195:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\CompleteParam.java
 */
package io.nuls.protocol.model;
```

```
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
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 CompleteParam extends BaseNulsData {

    private NulsDigestData requestHash;
    private boolean success;

    public CompleteParam() {
    }

    public CompleteParam(NulsDigestData requestHash, boolean success) {
        this.requestHash = requestHash;
        this.success = success;
    }

    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfNulsData(requestHash);
        size += SerializeUtils.sizeOfBoolean(success);
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeNulsData(requestHash);
        stream.writeBoolean(success);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
```

```
        this.requestHash = byteBuffer.readHash();
        this.success = byteBuffer.readBoolean();
    }

    public NulsDigestData getRequestHash() {
        return requestHash;
    }

    public void setRequestHash(NulsDigestData requestHash) {
        this.requestHash = requestHash;
    }

    public boolean isSuccess() {
        return success;
    }

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

196:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\GetBlockParam.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;

/**
 *
 * Request block or request block header data encapsulation.
 *
 * @author Niels
 */
public class GetBlockParam extends BaseNulsData {
```

```
    private NulsDigestData blockHash;

    public GetBlockParam() {
    }
    public GetBlockParam(NulsDigestData blockHash) {
        this.blockHash = blockHash;
    }

    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfNulsData(blockHash);
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeNulsData(blockHash);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.blockHash = byteBuffer.readHash();
    }

    public NulsDigestData getBlockHash() {
        return blockHash;
    }

    public void setBlockHash(NulsDigestData blockHash) {
        this.blockHash = blockHash;
    }
}

197:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\GetBlocksByHashParam.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
```

```
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;

/**
 *
 * Request block or request block header data encapsulation.
 *
 * @author Niels
 */
public class GetBlocksByHashParam extends BaseNulsData {
    /**
     *
     * The initial hash, when only one request is requested,
     * is the target hash, and when multiple requests are requested, it is the previous of the first
hash.
     */

    private NulsDigestData startHash;
    /**
     *
     * the last hash of request
     */

    private NulsDigestData endHash;

    public GetBlocksByHashParam() {
    }

    public GetBlocksByHashParam(NulsDigestData startHash, NulsDigestData endHash) {
        this.startHash = startHash;
        this.endHash = endHash;
    }

    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfNulsData(startHash);
        size += SerializeUtils.sizeOfNulsData(endHash);
```

```
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeNulsData(startHash);
        stream.writeNulsData(endHash);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.startHash = byteBuffer.readHash();
        this.endHash = byteBuffer.readHash();
    }

//    /**
//     *
//     * The initial hash, when only one request is requested,
//     * is the target hash, and when multiple requests are requested, it is the previous of the first
hash.
//     */
    public NulsDigestData getStartHash() {
        return startHash;
    }

    public void setStartHash(NulsDigestData startHash) {
        this.startHash = startHash;
    }

    /**
     *
     * the last hash of request
     *
     * @return
     */
    public NulsDigestData getEndHash() {
        return endHash;
    }

    public void setEndHash(NulsDigestData endHash) {
        this.endHash = endHash;
    }
```

```
}

198:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\GetBlocksByHeightParam.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;

/**
 *
 * Request block or request block header data encapsulation.
 *
 * @author Niels
 */
public class GetBlocksByHeightParam extends BaseNulsData {
    private long startHeight;
    private long endHeight;

    public GetBlocksByHeightParam() {
    }

    public GetBlocksByHeightParam(long startHeight, long endHeight) {
        this.startHeight = startHeight;
        this.endHeight = endHeight;
    }

    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfUint32();
        size += SerializeUtils.sizeOfUint32();
        return size;
    }
```

```
    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeUint32(startHeight);
        stream.writeUint32(endHeight);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.startHeight = byteBuffer.readUint32();
        this.endHeight = byteBuffer.readUint32();
    }

    public long getStartHeight() {
        return startHeight;
    }

    public void setStartHeight(long startHeight) {
        this.startHeight = startHeight;
    }

    public long getEndHeight() {
        return endHeight;
    }

    public void setEndHeight(long endHeight) {
        this.endHeight = endHeight;
    }
}

199:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\GetBlocksHashParam.java
 */

package io.nuls.protocol.model;

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 java.io.IOException;
```

```
/**
 *
 * Request block hashes data encapsulation.
 *
 * @author Niels
 */
public class GetBlocksHashParam extends BaseNulsData {

    /**
     *
     * The starting height of the request is the first height to be returned.
     */

    private long startHeight;

    /**
     *
     * end of the blocks request
     */

    private long endHeight;

    public GetBlocksHashParam() {
    }

    public GetBlocksHashParam(long startHeight, long endHeight) {
        this.startHeight = startHeight;
        this.endHeight = endHeight;
    }

    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfUint32();
        size += SerializeUtils.sizeOfUint32();
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeUint32(startHeight);
```

```
        stream.writeUint32(endHeight);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        startHeight = byteBuffer.readUint32();
        endHeight = byteBuffer.readUint32();
    }

    public long getStartHeight() {
        return startHeight;
    }

    public void setStartHeight(long startHeight) {
        this.startHeight = startHeight;
    }

    public long getEndHeight() {
        return endHeight;
    }

    public void setEndHeight(long endHeight) {
        this.endHeight = endHeight;
    }
}

200:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\GetTxGroupParam.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
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 java.io.IOException;
import java.util.ArrayList;
import java.util.List;
```

```
/**
 *
 * Request transaction list data encapsulation.
 *
 * @author Niels
 */
public class GetTxGroupParam extends BaseNulsData {

    /**
     *
     * the list of transaction digest data
     */

    private List<NulsDigestData> txHashList = new ArrayList<>();

    public GetTxGroupParam() {
    }

    @Override
    public int size() {
        int size = 0;
        size += VarInt.sizeOf(txHashList.size());
        size += this.getTxHashBytesLength();
        return size;
    }

    private int getTxHashBytesLength() {
        int size = 0;
        for (NulsDigestData hash : txHashList) {
            size += SerializeUtils.sizeOfNulsData(hash);
        }
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeVarInt(txHashList.size());
        for (NulsDigestData data : txHashList) {
            stream.writeNulsData(data);
        }
    }
```

```
    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        long txCount = byteBuffer.readVarInt();
        this.txHashList = new ArrayList<>();
        for (int i = 0; i < txCount; i++) {
            this.txHashList.add(byteBuffer.readHash());
        }
    }

//    /**
//     *
//     * the list of transaction digest data
//     */
    public List<NulsDigestData> getTxHashList() {
        return txHashList;
    }

//    /**
//     *
//     * add a tx hash to ask list
//     */
    public void addHash(NulsDigestData hash) {
        this.txHashList.add(hash);
    }

    public void setTxHashList(List<NulsDigestData> txHashList) {
        this.txHashList = txHashList;
    }
}

201:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\NotFound.java
 */

package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
```

```
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.protocol.constant.MessageDataType;

import java.io.IOException;

/**
 * @author: Niels Wang
 */
public class NotFound extends BaseNulsData {
    /**
     *  {@link MessageDataType}
     * data type
     */

    private MessageDataType type;

    /**
     *
     * request hash
     */

    private NulsDigestData hash;

    public NotFound() {
    }

    public NotFound(MessageDataType type, NulsDigestData hash) {
        this.type = type;
        this.hash = hash;
    }

    @Override
    public int size() {
        int size = 1;
        size += SerializeUtils.sizeOfNulsData(hash);
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.write((byte) type.getCode());
        stream.writeNulsData(hash);
```

```
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.type = MessageDataType.getType(byteBuffer.readByte());
        this.hash = byteBuffer.readHash();
    }

    /**
     *  {@link MessageDataType}
     * @return MessageDataType
     */
    public MessageDataType getType() {
        return type;
    }

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

    /**
     *
     * request hash
     * @return NulsDigestData
     */
    public NulsDigestData getHash() {
        return hash;
    }

    public void setHash(NulsDigestData hash) {
        this.hash = hash;
    }
}

202:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\ReactParam.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
```

```
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 ReactParam extends BaseNulsData {

    private NulsDigestData requestId;

    public ReactParam() {
    }

    public ReactParam(NulsDigestData requestId) {
        this.requestId = requestId;
    }

    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfNulsData(requestId);
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeNulsData(requestId);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.requestId = byteBuffer.readHash();
    }

    public NulsDigestData getRequestId() {
        return requestId;
    }

    public void setRequestId(NulsDigestData requestId) {
```

```
        this.requestId = requestId;
    }
}

203:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\SmallBlock.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

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

/**
 * hash
 * Block block, used for broadcasting after the new block is packaged,
 * and the blocks in the block are included in the block header ,tx hash list of the block
 * and the transaction generated in the packaging process (other transactions that must not be
made by other nodes).
 *
 * @author Niels
 */
public class SmallBlock extends BaseNulsData {
    /**
     *
     * block header
     */

    private BlockHeader header;

    /**
     *
     * transaction hash list
```

```
 */

private List<NulsDigestData> txHashList;

/**
 *
 * Consensus trading list (transactions that no other node must have)
 */

private List<Transaction> subTxList = new ArrayList<>();

public SmallBlock() {
}

@Override
public int size() {
    int size = header.size();
    size += SerializeUtils.sizeOfVarInt(txHashList.size());
    for (NulsDigestData hash : txHashList) {
        size += SerializeUtils.sizeOfNulsData(hash);
    }
    size += SerializeUtils.sizeOfVarInt(subTxList.size());
    for (Transaction tx : subTxList) {
        size += SerializeUtils.sizeOfNulsData(tx);
    }
    return size;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeNulsData(header);
    stream.writeVarInt(txHashList.size());
    for (NulsDigestData hash : txHashList) {
        stream.writeNulsData(hash);
    }
    stream.writeVarInt(subTxList.size());
    for (Transaction tx : subTxList) {
        stream.writeNulsData(tx);
    }
}

@Override
```

```
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.header = byteBuffer.readNulsData(new BlockHeader());

        this.txHashList = new ArrayList<>();
        long hashListSize = byteBuffer.readVarInt();
        for (int i = 0; i < hashListSize; i++) {
            this.txHashList.add(byteBuffer.readHash());
        }

        this.subTxList = new ArrayList<>();
        long subTxListSize = byteBuffer.readVarInt();
        for (int i = 0; i < subTxListSize; i++) {
            Transaction tx = byteBuffer.readTransaction();
            tx.setBlockHeight(header.getHeight());
            this.subTxList.add(tx);
        }
    }

    /**
     *
     * block header
     * @return BlockHeader
     */
    public BlockHeader getHeader() {
        return header;
    }

    public void setHeader(BlockHeader header) {
        this.header = header;
    }

//    /**
//     *
//     * transaction hash list
//     */
    public List<NulsDigestData> getTxHashList() {
        return txHashList;
    }

    public void setTxHashList(List<NulsDigestData> txHashList) {
        this.txHashList = txHashList;
    }
```

```
//    /**
//     *
//     * Consensus trading list (transactions that no other node must have)
//     */
    public List<Transaction> getSubTxList() {
        return subTxList;
    }

    public void addBaseTx(Transaction tx) {
        this.subTxList.add(tx);
    }

}

204:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\tx\CoinBaseTransaction.java
 */
package io.nuls.protocol.model.tx;

import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;

import java.util.Arrays;

/**
 * @author Niels
 */
public class CoinBaseTransaction extends Transaction {

    public CoinBaseTransaction() {
        this(ProtocolConstant.TX_TYPE_COINBASE);
    }

    @Override
    protected TransactionLogicData parseTxData(NulsByteBuffer byteBuffer) throws NulsException
{
        byteBuffer.readBytes(NulsConstant.PLACE_HOLDER.length);
```

```
        return null;
    }

    protected CoinBaseTransaction(int type) {
        super(type);
    }

    @Override
    public String getInfo(byte[] address) {
        Na to = Na.ZERO;
        for (Coin coin : coinData.getTo()) {
            //if (Arrays.equals(address, coin.()))
            if (Arrays.equals(address, coin.getAddress()))
            {
                to = to.add(coin.getNa());
            }
        }
        return "+" + to.toText();
    }

    @Override
    public boolean isSystemTx() {
        return true;
    }

    @Override
    public boolean needVerifySignature() {
        return false;
    }
}

205:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\tx\DataTransaction.java
 */
package io.nuls.protocol.model.tx;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.protocol.constant.ProtocolConstant;

/**
```

```
 * @author Niels
 */
public class DataTransaction extends Transaction<LogicData> {

    public DataTransaction() {
        this(ProtocolConstant.TX_TYPE_DATA);
    }

    protected DataTransaction(int type) {
        super(type);
    }

    @Override
    public String getInfo(byte[] address) {
        return "--";
    }

    @Override
    protected LogicData parseTxData(NulsByteBuffer byteBuffer) throws NulsException {
        return byteBuffer.readNulsData(new LogicData());
    }

}

206:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\tx\LogicData.java
 */
package io.nuls.protocol.model.tx;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;
import java.util.Set;

/**
 * @author: Niels Wang
 * @date: 2018/7/24
 */
```

```
public class LogicData extends TransactionLogicData {

    private byte[] bytes;

    public LogicData() {
    }

    public LogicData(byte[] bytes) {
        this.bytes = bytes;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeBytesWithLength(bytes);
    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        bytes = byteBuffer.readByLengthByte();
    }

    @Override
    public int size() {
        return SerializeUtils.sizeOfBytes(bytes);
    }

    @Override
    public Set<byte[]> getAddresses() {
        return null;
    }

    public void setBytes(byte[] bytes) {
        this.bytes = bytes;
    }

    public byte[] getBytes() {
        return bytes;
    }

}
```

207:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-

```
module\protocol\src\main\java\io\nuls\protocol\model\tx\TransferTransaction.java
 */
package io.nuls.protocol.model.tx;

import io.nuls.account.ledger.service.AccountLedgerService;
import io.nuls.kernel.constant.NulsConstant;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.*;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.ledger.service.LedgerService;
import io.nuls.ledger.util.LedgerUtil;
import io.nuls.protocol.constant.ProtocolConstant;

import java.math.BigDecimal;
import java.util.Arrays;

import static io.nuls.kernel.model.Na.SMALLEST_UNIT_EXPONENT;

/**
 * @author Niels
 */
public class TransferTransaction extends Transaction {

    private LedgerService ledgerService;

    private AccountLedgerService accountLedgerService;

    public TransferTransaction() {
        this(ProtocolConstant.TX_TYPE_TRANSFER);
    }

    protected TransferTransaction(int type) {
        super(type);
    }


    @Override
    public String getInfo(byte[] address) {
        Long value = 0L;

        byte[] fromHash, owner;
```

```
        int fromIndex;
        NulsDigestData fromHashObj;
        Transaction fromTx;
        Coin fromUtxo;
        for (Coin from : coinData.getFrom()) {
            owner = from.getOwner();
            // ownertxHashindex
            fromHash = LedgerUtil.getTxHashBytes(owner);
            fromIndex = LedgerUtil.getIndex(owner);

            // from UTXO
            fromHashObj = new NulsDigestData();
            try {
                fromHashObj.parse(fromHash, 0);
            } catch (NulsException e) {
                return "--";
            }
            Result<Transaction> result =
getAccountLedgerService().getUnconfirmedTransaction(fromHashObj);
            if (result.isSuccess() && result.getData() != null) {
                fromTx = result.getData();
            } else {
                fromTx = getLedgerService().getTx(fromHashObj);
            }
            fromUtxo = fromTx.getCoinData().getTo().get(fromIndex);
            //if (Arrays.equals(address, fromUtxo.()))
            if (Arrays.equals(address, fromUtxo.getAddress())) {
                value = value - fromUtxo.getNa().getValue();
            }
        }

        for (Coin to : coinData.getTo()) {
            if (Arrays.equals(address, to.getAddress())) {
                value = value + to.getNa().getValue();
            }
        }
        long divide = (long) Math.pow(10, SMALLEST_UNIT_EXPONENT);
        BigDecimal decimal = new BigDecimal(value).divide(BigDecimal.valueOf(divide));
        if(decimal.doubleValue() > 0) {
            return "+" + decimal.toPlainString();
        }
        return decimal.toPlainString();
```

```
    }

    @Override
    protected TransactionLogicData parseTxData(NulsByteBuffer byteBuffer) throws NulsException
{
        byteBuffer.readBytes(NulsConstant.PLACE_HOLDER.length);
        return null;
    }

    public LedgerService getLedgerService() {
        if (ledgerService == null) {
            ledgerService = NulsContext.getServiceBean(LedgerService.class);
        }
        return ledgerService;
    }

    public AccountLedgerService getAccountLedgerService() {
        if (accountLedgerService == null) {
            accountLedgerService = NulsContext.getServiceBean(AccountLedgerService.class);
        }
        return accountLedgerService;
    }
}

208:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\TxGroup.java
 */
package io.nuls.protocol.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.kernel.utils.VarInt;

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

```
import java.util.Map;

/**
 *
 * The encapsulation of the transaction list used to return a peer request.
 *
 * @author Niels
 */
public class TxGroup extends BaseNulsData {

    /**
     *
     * transaction list for response
     */

    private NulsDigestData requestHash;

    private List<Transaction> txList;

    /**
     * hashmap
     * The transaction is sorted into a hashmap.
     */
    private transient Map<NulsDigestData, Transaction> txMap;

    @Override
    public int size() {
        int size = 0;
        size += requestHash.size();
        size += VarInt.sizeOf(txList.size());
        size += this.getTxListLength();
        return size;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeNulsData(requestHash);
        stream.writeVarInt(txList.size());
        for (Transaction data : txList) {
            stream.writeNulsData(data);
        }
    }
```

```
    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        requestHash = byteBuffer.readHash();
        long txCount = byteBuffer.readVarInt();
        this.txList = new ArrayList<>();
        for (int i = 0; i < txCount; i++) {
            try {
                this.txList.add(byteBuffer.readTransaction());
            } catch (Exception e) {
                throw new NulsException(e);
            }
        }
        initTxMap();
    }

    private int getTxListLength() {
        int size = 0;
        for (Transaction tx : txList) {
            size += SerializeUtils.sizeOfNulsData(tx);
        }
        return size;
    }

    /**
     * hashmap
     * The transaction is sorted into a hashmap.
     */
    private synchronized void initTxMap() {
        if (null != txMap) {
            return;
        }
        this.txMap = new HashMap<>();
        for (Transaction tx : txList) {
            txMap.put(tx.getHash(), tx);
        }
    }

//    /**
//     *
//     * transaction list for response
//     */
```

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

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

    public Transaction getTx(NulsDigestData hash) {
        return txMap.get(hash);
    }

//    /**
//     * hashmap
//     * The transaction is sorted into a hashmap.
//     */
    public Map<NulsDigestData, Transaction> getTxMap() {
        if (null == txMap) {
            initTxMap();
        }
        return txMap;
    }

    public NulsDigestData getRequestHash() {
        return requestHash;
    }

    public void setRequestHash(NulsDigestData requestHash) {
        this.requestHash = requestHash;
    }
}

209:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\validator\BlockFieldValidator.java
 */
package io.nuls.protocol.model.validator;

import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
```

```
import io.nuls.protocol.constant.ProtocolErroeCode;

/**
 * @author Niels
 */
@Component
public class BlockFieldValidator implements NulsDataValidator<Block> {

    @Override
    public ValidateResult validate(Block data) {
        ValidateResult result = ValidateResult.getSuccessResult();
        boolean failed = false;
        do {
            if (data == null) {
                failed = true;
                break;
            }
            if (data.getHeader() == null) {
                failed = true;
                break;
            }
            if (data.getTxs() == null || data.getTxs().isEmpty()) {
                failed = true;
                break;
            }

            if(data.getHeader().getTxCount() == 0 || data.getTxs().size() !=
data.getHeader().getTxCount()){
                failed = true;
                break;
            }

        } while (false);
        if (failed) {
            result = ValidateResult.getFailedResult(this.getClass().getName(),
ProtocolErroeCode.BLOCK_FIELD_CHECK_FAILED);
        }
        return result;
    }
}

210:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
```

```
module\protocol\src\main\java\io\nuls\protocol\model\validator\BlockHeaderValidator.java
 */
package io.nuls.protocol.model.validator;

import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolErroeCode;

/**
 * @author Niels
 */
@Component
public class BlockHeaderValidator implements NulsDataValidator<Block> {
    @Override
    public ValidateResult validate(Block data) {
        if (null == data || data.getHeader() == null) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
ProtocolErroeCode.BLOCK_IS_NULL);
        }
        return data.getHeader().verify();
    }
}

211:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\validator\BlockMaxSizeValidator.java
 */
package io.nuls.protocol.model.validator;

import io.nuls.contract.constant.ContractConstant;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolConstant;
import io.nuls.protocol.constant.ProtocolErroeCode;

/**
 * @author Niels
```

```
 */
@Component
public class BlockMaxSizeValidator implements NulsDataValidator<Block> {

    @Override
    public ValidateResult validate(Block data) {
        if (data == null) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.NULL_PARAMETER);
        }
        long length = 0L;
        for (Transaction tx : data.getTxs()) {
            // pierre add  - ()
            if (tx.isSystemTx()) {
                continue;
            }
            length += tx.size();
        }
        if (length > ProtocolConstant.MAX_BLOCK_SIZE) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
ProtocolErroeCode.BLOCK_TOO_BIG);
        }
        return ValidateResult.getSuccessResult();
    }
}

212:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\validator\BlockMerkleValidator.java
 */
package io.nuls.protocol.model.validator;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolErroeCode;

/**
 * @author Niels
 */
```

```
@Component
public class BlockMerkleValidator implements NulsDataValidator<Block> {

    @Override
    public ValidateResult validate(Block data) {
        ValidateResult result = ValidateResult.getFailedResult(this.getClass().getName(),
ProtocolErroeCode.MERKLE_HASH_WRONG);
        do {
            if (null == data) {
                result.setMsg("Data is null!");
                break;
            }
            if
(data.getHeader().getMerkleHash().equals(NulsDigestData.calcMerkleDigestData(data.getTxHash
List()))) {
                result = ValidateResult.getSuccessResult();
                break;
            }
        } while (false);
        return result;
    }

}

213:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\validator\HeaderFieldValidator.java
 */
package io.nuls.protocol.model.validator;

import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolErroeCode;

/**
 * @author Niels
 */
@Component
public class HeaderFieldValidator implements NulsDataValidator<BlockHeader> {
```

```
    private static final int OLD_HEADER_EXTENDS_MAS_SIZE = 64;
    private static final int HEADER_EXTENDS_MAS_SIZE = 1024;

    @Override
    public ValidateResult validate(BlockHeader data) {
        ValidateResult result = ValidateResult.getSuccessResult();
        boolean failed = false;
        do {
            if (data.getHash() == null) {
                failed = true;
                break;
            }
            if (data.getHeight() < 0) {
                failed = true;
                break;
            }
            if (data.getMerkleHash() == null) {
                failed = true;
                break;
            }
            if (null == data.getPackingAddress()) {
                failed = true;
                break;
            }
            if (null != data.getExtend()) {
                if (NulsContext.MAIN_NET_VERSION <= 1 && data.getExtend().length >
OLD_HEADER_EXTENDS_MAS_SIZE) {
                    failed = true;
                    break;
                }
                if (data.getExtend().length > HEADER_EXTENDS_MAS_SIZE) {
                    failed = true;
                    break;
                }
            }
        } while (false);
        if (failed) {
            result = ValidateResult.getFailedResult(this.getClass().getName(),
ProtocolErroeCode.BLOCK_HEADER_FIELD_CHECK_FAILED);
        }
        return result;
    }
```

```
}

214:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\validator\HeaderSignValidator.java
 */
package io.nuls.protocol.model.validator;

import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolErroeCode;

/**
 * @author Niels
 */
@Component
public class HeaderSignValidator implements NulsDataValidator<BlockHeader> {

    @Override
    public ValidateResult validate(BlockHeader data) {
        if (data.getBlockSignature() == null) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
ProtocolErroeCode.BLOCK_HEADER_SIGN_CHECK_FAILED);
        }
        return data.getBlockSignature().verifySignature(data.getHash());
    }
}

215:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\validator\TxCoinValidator.java
                return ValidateResult.getSuccessResult();
            }
            List<Coin> toList = tx.getCoinData().getTo();
            if(toList == null || toList.size() == 0){
                return ValidateResult.getSuccessResult();
            }
            for (Coin coin:toList) {
                if(coin.getOwner().length == Address.ADDRESS_LENGTH && coin.getOwner()[2] ==
NulsContext.P2SH_ADDRESS_TYPE){
                    return ValidateResult.getFailedResult(this.getClass().getName(),
```

```
KernelErrorCode.COIN_OWNER_ERROR);
                }
            }
        } catch (Exception e) {
            Log.error(e);
            return ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.DATA_ERROR);
        }
        return ValidateResult.getSuccessResult();
    }
}
```

216:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\model\validator\TxFeeValidator.java

```
package io.nuls.protocol.model.validator;

import io.nuls.contract.constant.ContractConstant;
import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.CoinData;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.TransactionFeeCalculator;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 * @author Niels
 */
@Component
public class TxFeeValidator implements NulsDataValidator<Transaction> {
    @Override
    public ValidateResult validate(Transaction tx) {
        int txType = tx.getType();
        if (tx.isSystemTx()) {
            return ValidateResult.getSuccessResult();
        }
        CoinData coinData = tx.getCoinData();
        if (null == coinData) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.COINDATA_NOT_FOUND);
```

```
        }
        Na realFee = tx.getFee();
        Na fee = null;
        if (txType == ProtocolConstant.TX_TYPE_TRANSFER
                || txType == ProtocolConstant.TX_TYPE_DATA
                || txType == ContractConstant.TX_TYPE_CREATE_CONTRACT
                || txType == ContractConstant.TX_TYPE_CALL_CONTRACT
                || txType == ContractConstant.TX_TYPE_DELETE_CONTRACT) {
            fee = TransactionFeeCalculator.getTransferFee(tx.size());
        } else {
            fee = TransactionFeeCalculator.getMaxFee(tx.size());
        }
        if (realFee.isGreaterOrEquals(fee)) {
            return ValidateResult.getSuccessResult();
        }
        return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.FEE_NOT_RIGHT);
    }
}
```

217:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\model\validator\TxFieldValidator.java

```
package io.nuls.protocol.model.validator;

import io.nuls.kernel.constant.TransactionErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

/**
 * @author Niels
 */
@Component
public class TxFieldValidator implements NulsDataValidator<Transaction> {

    public final static int MAX_REMARK_LEN = 100;
    public final static int MAX_TX_TYPE = 60000;


    @Override
    public ValidateResult validate(Transaction tx) {
```

```
        boolean result = true;
        do {
            if (tx == null) {
                result = false;
                break;
            }
            if (tx.getHash() == null || tx.getHash().size() == 0 || tx.getHash().size() > 70) {
                result = false;
                break;
            }
            if (tx.getType() == 0 || tx.getType() > MAX_TX_TYPE) {
                result = false;
                break;
            }
            if (tx.getTime() == 0) {
                result = false;
                break;
            }
            if (tx.getRemark() != null && tx.getRemark().length > MAX_REMARK_LEN) {
                result = false;
                break;
            }
        } while (false);
        if (!result) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
TransactionErrorCode.TX_DATA_VALIDATION_ERROR);
        }
        return ValidateResult.getSuccessResult();
    }
}
```

218:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\model\validator\TxMaxSizeValidator.java

```
package io.nuls.protocol.model.validator;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

/**
```

```
 * @author Niels
 */
@Component
public class TxMaxSizeValidator implements NulsDataValidator<Transaction> {
    public static final int MAX_TX_BYTES = 300;
    public static final int MAX_TX_SIZE = MAX_TX_BYTES * 1024;

    @Override
    public ValidateResult validate(Transaction data) {
        if (data.size() > MAX_TX_SIZE) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.DATA_SIZE_ERROR);
        }
        return ValidateResult.getSuccessResult();
    }
}

219:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\model\validator\TxRemarkValidator.java
package io.nuls.protocol.model.validator;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

/**
 * @author Niels
 */
@Component
public class TxRemarkValidator implements NulsDataValidator<Transaction> {
    public final static int MAX_REMARK_LEN = 100;

    @Override
    public ValidateResult validate(Transaction data) {
        byte[] remark = data.getRemark();
        if (remark != null && remark.length > MAX_REMARK_LEN) {
            return ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.DATA_SIZE_ERROR);
        }
        return ValidateResult.getSuccessResult();
```

```
    }
}
```

220:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\model\validator\TxSignValidator.java

```
package io.nuls.protocol.model.validator;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

/**
 * @author Niels
 */
@Component
public class TxSignValidator implements NulsDataValidator<Transaction> {

    @Override
    public ValidateResult validate(Transaction tx) {
        try {
            if (SignatureUtil.validateTransactionSignture(tx)) {
                return ValidateResult.getSuccessResult();
            }
        } catch (Exception e) {
            Log.error(e);
        }
        return ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.SIGNATURE_ERROR);
    }
}
```

221:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\model\validator\TxVersionForScriptValidator.java

```
package io.nuls.protocol.model.validator;

import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.lite.annotation.Component;
```

```
import io.nuls.kernel.model.Address;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.kernel.utils.SerializeUtils;
import io.nuls.kernel.validate.NulsDataValidator;
import io.nuls.kernel.validate.ValidateResult;

import java.util.List;

/**
 * @author Niels
 */
@Component
public class TxVersionForScriptValidator implements NulsDataValidator<Transaction> {
    public final static int MAX_REMARK_LEN = 100;


    @Override
    public ValidateResult validate(Transaction tx) {
        if (NulsContext.MAIN_NET_VERSION > 1) {
            return ValidateResult.getSuccessResult();
        }
        if (null == tx.getCoinData() || tx.getCoinData().getTo() == null ||
tx.getCoinData().getTo().isEmpty()) {
            return ValidateResult.getSuccessResult();
        }
        List<Coin> toList = tx.getCoinData().getTo();
        ValidateResult failed = ValidateResult.getFailedResult(this.getClass().getName(),
KernelErrorCode.VERSION_NOT_NEWEST);
        for (Coin coin : toList) {
            if (coin.getOwner().length != Address.ADDRESS_LENGTH) {
                return failed;
            }
            if (coin.getOwner()[2] != NulsContext.DEFAULT_ADDRESS_TYPE) {
                return failed;
            }
        }
        return ValidateResult.getSuccessResult();
    }
}
```

222:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\module\AbstractProtocolModule.java

```
package io.nuls.protocol.module;

import io.nuls.kernel.module.BaseModuleBootstrap;
import io.nuls.protocol.constant.ProtocolConstant;

/**
 * ,id
 * An abstract class of the protocol module launcher used to bind module id.
 *
 * @author Niels
 */
public abstract class AbstractProtocolModule extends BaseModuleBootstrap {
    public AbstractProtocolModule() {
        super(ProtocolConstant.MODULE_ID_PROTOCOL);
    }

}
```

223:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\service\BlockService.java

```
 */
package io.nuls.protocol.service;

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;

/**
 *
 * The block handles the service interface.
 *
 * @author Niels
 */
public interface BlockService {
    /**
     *
```

```
     * Get the creation block (from storage)
     */
    Result<Block>  getGengsisBlock();

    /**
     *
     * Get the highest block (from storage)
     */
    Result<Block> getBestBlock();

    /**
     *
     * Get the highest block header (from storage)
     */
    Result<BlockHeader> getBestBlockHeader();

    /**
     *
     * Get the block head (from storage) according to the block height
     *
     * @param height /block height
     * @return
     */
    Result<BlockHeader> getBlockHeader(long height);

    /**
     *
     * Get the block head (from storage) according to the block hash
     *
     * @param hash /block hash
     * @return /block header
     */
    Result<BlockHeader> getBlockHeader(NulsDigestData hash);

    /**
     *
     * Get the block (from storage) according to the block hash
     *
     * @param hash /block hash
     * @return /block
     */
    Result<Block> getBlock(NulsDigestData hash);
```

```
    /**
     *
     * Get the block (from storage) according to the block hash
     *
     * @param hash /block hash
     * @param isNeedContractTransfer ()/If necessary to add the contract transfer (from the contract) to the block
     * @return /block
     */
    Result<Block> getBlock(NulsDigestData hash, boolean isNeedContractTransfer);

    /**
     *
     * Get the block (from storage) according to the block height
     *
     * @param height /block height
     * @return /block
     */
    Result<Block> getBlock(long height);

    /**
     *
     * Get the block (from storage) according to the block height
     *
     * @param height /block height
     * @param isNeedContractTransfer ()/If necessary to add the contract transfer (from the contract) to the block
     * @return /block
     */
    Result<Block> getBlock(long height, boolean isNeedContractTransfer);

    /**
     *
     * Save the block to the store.
     *
     * @param block /whole block
     * @return /operating result
     * @throws NulsException
     * There may be exceptions to the save block, please handle it carefully after capture.
     */
    Result saveBlock(Block block) throws NulsException;
```

```
    /**
     *
     * roll back the block to the store.
     *
     * @param block /whole block
     * @return /operating result
     * @throws NulsException
     * There may be exceptions to the roll back block, please handle it carefully after capture.
     */
    Result rollbackBlock(Block block) throws NulsException;

    /**
     *
     * Forward block to other peers of the connection, allowing one column (not forward to it)
     *
     * @param hash       /the hash of block
     * @param excludeNode
     *                   The nodes that need to be excluded are generally due to the block received from
the node.
     * @return /forward results
     */
    Result forwardBlock(NulsDigestData hash, Node excludeNode);

    /**
     *
     * The broadcast block gives the connection to other peers.
     *
     * @param block /the whole block
     * @return /Broadcast the results
     */
    Result broadcastBlock(SmallBlock block);
}

224:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\service\DownloadService.java
 */

package io.nuls.protocol.service;

import io.nuls.kernel.model.Block;
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 java.util.List;

/**
 * /
 * Block/transaction download service interface.
 *
 * @author Niels
 */
public interface DownloadService {

    /**
     * hash
     * Download a block according from the node to the hash, and the download process is blocked.
     *
     * @param hash /block hash
     * @param node /Specified node
     * @return / block  results
     */
    Result<Block> downloadBlock(NulsDigestData hash, Node node);

    /**
     *
     * Returns the results of the download.
     * @return Result
     */
    Result isDownloadSuccess();

    /**
     *
     * Recheck whether the current state needs to be resynchronized, and download if necessary.
     * @return Result
     */
    Result reset();

}
```

225:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-module\protocol\src\main\java\io\nuls\protocol\service\TransactionService.java

```
 */

package io.nuls.protocol.service;

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.validate.ValidateResult;
import io.nuls.network.model.Node;

import java.util.List;

/**
 *
 * Transaction operation service interface.
 *
 * @author: Niels Wang
 */
public interface TransactionService {

    /**
     *
     * Identify the method that is invoked during the transaction and submit the transaction related business.
     *
     * @param tx            /The transaction of the operation
     * @param secondaryData /Secondary data (available for null)
     * @return /operating results
     */
    Result commitTx(Transaction tx, Object secondaryData);

    /**
     *
     * The method invoked when the transaction is rolled back and the transaction related business is returned.
     *
     * @param tx            /The transaction of the operation
     * @param secondaryData /Secondary data (available for null)
     * @return /operating results
     */
    Result rollbackTx(Transaction tx, Object secondaryData);
```

```
    /**
     *
     * Forward Transaction to other peers of the connection, allowing one column (not forward to it)
     *
     * @param tx          /the whole transaction
     * @param excludeNode /The nodes that need to be excluded are generally
     *                    due to the transaction received from the node.
     * @return /forward results
     */
    Result forwardTx(Transaction tx, Node excludeNode);

    /**
     *
     * The broadcast transaction gives the connection to other peers.
     *
     * @param tx /the whole transaction
     * @return /Broadcast the results
     */
    Result broadcastTx(Transaction tx);

    /**
     *
     * cache the transaction for consensus
     *
     * @param tx transaction
     * @return Result
     */
    Result newTx(Transaction tx);

    /**
     *
     * <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.
     *
     * @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.
     */
    ValidateResult conflictDetect(List<Transaction> txList);
```

```
    /**
     * hash
     * get transaction by tx hash
     * @param hash
     * @return
     */
    Transaction getTx(NulsDigestData hash);
}

226:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\utils\HashSetDuplicateProcessor.java
 */

package io.nuls.protocol.utils;

import io.nuls.kernel.model.NulsDigestData;

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

/**
 * @author: Niels Wang
 * @date: 2018/7/9
 */
public class HashSetDuplicateProcessor {

    private Set<NulsDigestData> set1 = new HashSet<>();
    private Set<NulsDigestData> set2 = new HashSet<>();
    private final int maxSize;
    private final int percent90;

    public HashSetDuplicateProcessor(int maxSize) {
        this.maxSize = maxSize;
        this.percent90 = maxSize * 9 / 10;
    }

    public boolean insertAndCheck(NulsDigestData hash) {
        boolean result = set1.add(hash);
        if (!result) {
            return result;
        }
```

```
        int size = set1.size();
        if (size >= maxSize) {
            set1.clear();
            set1.addAll(set2);
            set2.clear();
        } else if (size >= percent90) {
            set2.add(hash);
        }
        return result;
    }

    public void remove(NulsDigestData hash) {
        set1.remove(hash);
        set2.remove(hash);
    }
}

227:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\utils\SmallBlockDuplicateRemoval.java
 */

package io.nuls.protocol.utils;

import io.nuls.kernel.model.NulsDigestData;

/**
 *
 *
 * @author: Niels Wang
 * @date: 2018/7/8
 */
public class SmallBlockDuplicateRemoval {

    private static HashSetDuplicateProcessor processorOfSmallBlock = new
HashSetDuplicateProcessor(1000);
    private static HashSetDuplicateProcessor processorOfForward = new
HashSetDuplicateProcessor(1000);

    public static boolean needDownloadSmallBlock(NulsDigestData hash) {
        return processorOfForward.insertAndCheck(hash);
    }
```

```
    public static boolean needProcess(NulsDigestData hash) {
        processorOfForward.insertAndCheck(hash);
        return processorOfSmallBlock.insertAndCheck(hash);
    }

    public static void removeForward(NulsDigestData hash) {
        processorOfForward.remove(hash);
    }
}

228:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\main\java\io\nuls\protocol\utils\TransactionTimeComparator.java
 */
package io.nuls.protocol.utils;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.NulsByteBuffer;

import java.util.Comparator;

/**
 * author Facjas
 * date 2018/5/23.
 */
public class TransactionTimeComparator implements Comparator<Transaction> {
    private static TransactionTimeComparator instance = new TransactionTimeComparator();

    private TransactionTimeComparator() {

    }

    public static TransactionTimeComparator getInstance() {
        return instance;
    }

    @Override
    public int compare(Transaction o1, Transaction o2) {
        if (o1.getHash().equals(o2.getHash())) {
```

```
            return 0;
        }
        if (o1.getTime() < o2.getTime()) {
            return -1;
        } else if (o1.getTime() > o2.getTime()) {
            return 1;
        } else {
            for (Coin coin : o1.getCoinData().getFrom()) {
                NulsByteBuffer buffer = new NulsByteBuffer(coin.getOwner());
                NulsDigestData hash = null;
                try {
                    hash = buffer.readHash();
                } catch (NulsException e) {
                    Log.error(e);
                }
                if (o2.getHash().equals(hash)) {
                    return 1;
                }
            }
            for (Coin coin : o2.getCoinData().getFrom()) {
                NulsByteBuffer buffer = new NulsByteBuffer(coin.getOwner());
                NulsDigestData hash = null;
                try {
                    hash = buffer.readHash();
                } catch (NulsException e) {
                    Log.error(e);
                }
                if (o1.getHash().equals(hash)) {
                    return -1;
                }
            }
        }
        return 0;
    }
}

229:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\test\java\io\nuls\protocol\cache\CacheTestTx.java
 */

package io.nuls.protocol.cache;
```

```
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.model.TransactionLogicData;
import io.nuls.kernel.utils.NulsByteBuffer;

/**
 * @author: Niels Wang
 */
public class CacheTestTx extends Transaction {
    public CacheTestTx() {
        super(1234);
    }

    @Override
    protected TransactionLogicData parseTxData(NulsByteBuffer byteBuffer) {
        return null;
    }

    @Override
    public String getInfo(byte[] address) {
        return null;
    }
}

230:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\test\java\io\nuls\protocol\cache\LimitHashMapTest.java
 */

package io.nuls.protocol.cache;

import io.nuls.cache.CacheMap;
import io.nuls.cache.LimitHashMap;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.protocol.model.tx.TransferTransaction;
import org.junit.Test;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
```

```
import static junit.framework.TestCase.assertTrue;

/**
 * @author: Niels Wang
 * @date: 2018/7/6
 */
public class LimitHashMapTest {

    @Test
    public void test() throws IOException {
        LimitHashMap<NulsDigestData, Transaction> map = new LimitHashMap<>(200000);
        long use = 0;
        List<NulsDigestData> hashList = new ArrayList<>();
        for (int i = 0; i < 200000; i++) {
            Transaction transaction = new TransferTransaction();
            transaction.setTime(System.currentTimeMillis());
            transaction.setHash(NulsDigestData.calcDigestData(transaction.serializeForHash()));
            hashList.add(transaction.getHash());
            long start = System.nanoTime();
            map.put(transaction.getHash(), transaction);
            use += (System.nanoTime() - start);
        }
        System.out.println("20" + use + "");
        long start = System.currentTimeMillis();
        for (int i = 0; i < 100000; i++) {
            map.getQueue().size();
        }
        System.out.println("queue size 100000" + (System.currentTimeMillis() - start) + "ms");
        start = System.currentTimeMillis();
        for (int i = 0; i < 100000; i++) {
            map.getMap().size();
        }
        System.out.println("map size 100000" + (System.currentTimeMillis() - start) + "ms");
        start = System.currentTimeMillis();
        for (NulsDigestData key : hashList) {
            map.get(key);
        }
        System.out.println("200000" + (System.currentTimeMillis() - start) + "ms");
        start = System.currentTimeMillis();
        for (NulsDigestData key : hashList) {
            map.containsKey(key);
        }
```

```
        System.out.println("200000" + (System.currentTimeMillis() - start) + "ms");
        start = System.currentTimeMillis();
        for (NulsDigestData key : hashList) {
            map.remove(key);
        }
        System.out.println("200000" + (System.currentTimeMillis() - start) + "ms");
        assertTrue(true);

    }

    @Test
    public void test1() throws IOException {
        CacheMap<NulsDigestData, Transaction> map = new CacheMap<>("a-test", 128,
NulsDigestData.class, Transaction.class);
        long use = 0;
        List<NulsDigestData> hashList = new ArrayList<>();
        for (int i = 0; i < 200000; i++) {
            Transaction transaction = new TransferTransaction();
            transaction.setTime(System.currentTimeMillis());
            transaction.setHash(NulsDigestData.calcDigestData(transaction.serializeForHash()));
            hashList.add(transaction.getHash());
            long start = System.nanoTime();
            map.put(transaction.getHash(), transaction);
            use += (System.nanoTime() - start);
        }
        System.out.println("20" + use + "");
        long start = System.currentTimeMillis();
        for (NulsDigestData key : hashList) {
            map.get(key);
        }
        System.out.println("200000" + (System.currentTimeMillis() - start) + "ms");
        start = System.currentTimeMillis();
        for (NulsDigestData key : hashList) {
            map.containsKey(key);
        }
        System.out.println("200000" + (System.currentTimeMillis() - start) + "ms");
        start = System.currentTimeMillis();
        for (NulsDigestData key : hashList) {
            map.remove(key);
        }
        System.out.println("200000" + (System.currentTimeMillis() - start) + "ms");
        assertTrue(true);
```

```
    }

    @Test
    public void testSet() {
        List<NulsDigestData> list = new ArrayList<>();
        for (int i = 0; i < 1000000; i++) {
            list.add(NulsDigestData.calcDigestData((i + "").getBytes()));
        }
        Set<NulsDigestData> set = new HashSet<>();
        long start = System.nanoTime();
        for (NulsDigestData hash : list) {
            set.add(hash);
        }
        System.out.println("set100" + (System.nanoTime() - start)/1000000 + "ms");
        Set<NulsDigestData> set2 = new HashSet<>();
        start = System.nanoTime();
        set2.addAll(set);
        System.out.println("addAll 100" + (System.nanoTime() - start)/1000000 + "ms");
        start = System.nanoTime();
        for (NulsDigestData hash : list) {
            set2.remove(hash);
        }
        System.out.println("remove100" + (System.nanoTime() - start)/1000000 + "ms");
        start = System.nanoTime();
        for (NulsDigestData hash : list) {
            set.contains(hash);
        }
        System.out.println("contains100" + (System.nanoTime() - start)/1000000 + "ms");
        start = System.nanoTime();
        set.clear();
        System.out.println("clear100" + (System.nanoTime() - start)/1000000 + "ms");

    }

}

231:F:\git\coin\nuls\nuls-1.1.3\nuls\protocol-
module\protocol\src\test\java\io\nuls\protocol\cache\TemporaryCacheManagerTest.java
 */

package io.nuls.protocol.cache;
```

```
import io.nuls.cache.manager.EhCacheManager;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.model.BlockHeader;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.protocol.model.SmallBlock;
import org.junit.Before;
import org.junit.Test;

import java.io.IOException;

import static org.junit.Assert.*;

/**
 *
 * <p>
 * Temporary cache manager test class, test temporary cache storage, fetch, delete, clean, destroy function.
 *
 * @author: Niels Wang
 */
public class TemporaryCacheManagerTest {

    private TemporaryCacheManager manager;

    /**
     * TemporaryCacheManagerTemporaryCacheManagergetInstance
     * <p>
     * To obtain the TemporaryCacheManager object, TemporaryCacheManager is implemented in singleton mode,
     * and the only instance in the virtual machine can be obtained by calling the getInstance method.
     */
    @Before
    public void init() {
        manager = TemporaryCacheManager.getInstance();
    }

    /**
     *
     * 1.
```

```
     * 2.
     * 3. null
     * 4.
     * test the cache processing process of the smallblock:
     * 1. The first test is put into the cache, and no exceptions are deemed to be successful.
     * 2. Test to get the newly inserted smallblock fast, the results can not be empty and the same content as if it is just as if.
     * 3. Before the test is deleted, the cells should be put into the smallblock quickly. After deleting, it should be null.
     * 4. Reattach the smallblock to the cache, call the cleaning method, and there should be no blocks or transactions in the cache.
     */
    @Test
    public void cacheSmallBlock() {
        SmallBlock smallBlock = new SmallBlock();
        BlockHeader header = new BlockHeader();
        NulsDigestData hash = NulsDigestData.calcDigestData("abcdefg".getBytes());
        header.setHash(hash);
        manager.cacheSmallBlock(smallBlock);
        assertTrue(true);

        this.getSmallBlock(hash, smallBlock);

        this.removeSmallBlock(hash);

        manager.cacheSmallBlock(smallBlock);

        this.clear();
    }

    /**
     *
     * <p>
     * The test gets the small block in the cache.
     *
     * @param hash       A quick summary of the community that has been deposited.
     * @param smallBlock The cached community is fast.
     */
    private void getSmallBlock(NulsDigestData hash, SmallBlock smallBlock) {
        SmallBlock sb =
manager.getSmallBlockByHash(NulsDigestData.calcDigestData("abcdefg".getBytes()));
        assertEquals(sb.getHeader().getHash(), smallBlock.getHeader().getHash());
```

```
    }

    /**
     *
     * 1.
     * 2.
     * 3. null
     * 4.
     * the cache processing flow of test transaction:
     * 1. The first test is put into the cache, and no exceptions are deemed to be successful.
     * 2. Test to obtain the newly placed transaction, the results obtained cannot be empty and the same content as the new one.
     * 3. Delete the transaction that was put in before the test is deleted, and then get null after deleting.
     * 4. Return the transaction to the cache, call the cleaning method, and there should be no blocks or transactions in the cache.
     */
    @Test
    public void cacheTx() {
        Transaction tx = new CacheTestTx();
        tx.setTime(1234567654L);
        try {
            tx.setHash(NulsDigestData.calcDigestData(tx.serializeForHash()));
        } catch (IOException e) {
            Log.error(e);
        }
        manager.cacheTx(tx);
        assertTrue(true);

        getTx(tx.getHash(), tx);
    }

    /**
     *
     * <p>
     * The test gets the transaction in the cache.
     *
     * @param hash A quick summary of the transaction that has been deposited.
     * @param tx   The cached Transaction is fast.
     */
    private void getTx(NulsDigestData hash, Transaction tx) {
        Transaction txGoted = manager.getTx(hash);
```

```
        assertNotNull(tx);
        assertEquals(tx.getTime(), txGoted.getTime());
    }


    /**
     *
     * Remove the corresponding block from the cache according to the block summary object, and determine whether the removal is successful.
     */
    public void removeSmallBlock(NulsDigestData hash) {
        SmallBlock smallBlock = manager.getSmallBlockByHash(hash);
        assertNotNull(smallBlock);
    }


    /**
     *
     * clear the cache and verify the results
     */
    public void clear() {
        manager.clear();
        assertEquals(manager.getSmallBlockCount(), 0);
        assertEquals(manager.getTxCount(), 0);
    }

    /**
     *
     * destroy the cache and verify the results
     */
    @Test
    public void destroy() {
        manager.destroy();
        assertNull(EhCacheManager.getInstance().getCache("temp-small-block-cache"));
        assertNull(EhCacheManager.getInstance().getCache("temp-tx-cache"));
    }
}
```